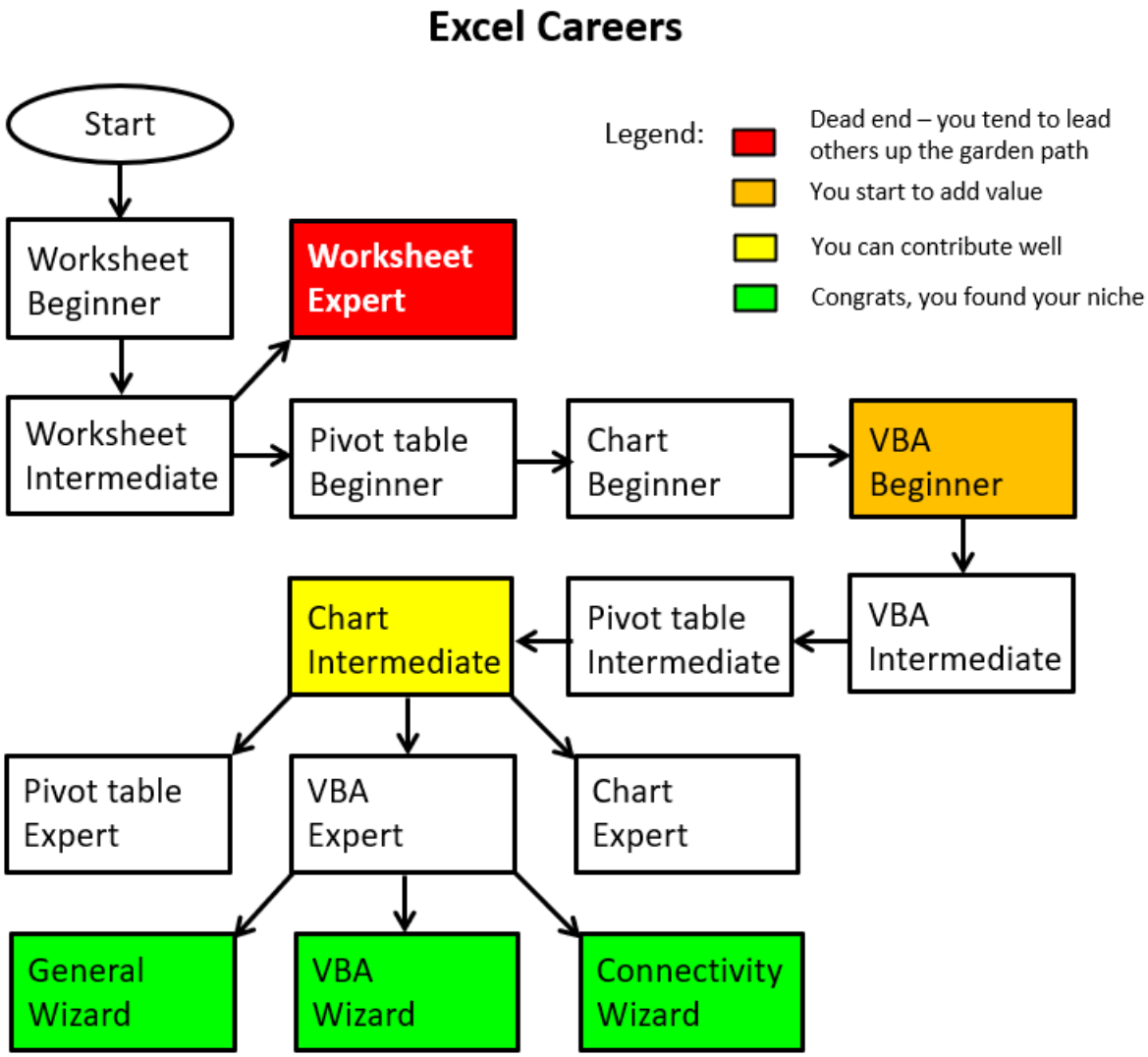
Excel / VBA – A Collection



(C) (P) 2025 Bernd Plumhoff Status: 09 May 2025

Abstract

This is a collection of Excel VBA programs and of some Excel spreadsheet formulas which I found useful.

Bernd Plumhoff, 09 May 2025

Excel Learning Path

Excel Area / Level of expertise	**What you should know**	**Leading others up the garden path**	**Areas or levels you might want to develop into**
Worksheet functions / Muppet	Not to format all cells right aligned: cell format General shows strings left aligned and numbers right aligned; not to have erroneous name definitions		Learn about functions like IF, TEXT, DATE, XLOOKUP
VBA / Muppet	Not to name functions or subs identically to modules		Record macros
Worksheet functions / Beginner	1<>"1", functions: CHAR, DATE, MID, ROUND, TEXT, TODAY, XLOOKUP	Showing off with formulas you do not understand	Learn about functions like SUMPRODUCT, understand volatile functions
VBA / Beginner	How to record macros and to copy others' code fragments, integers can only hold values from -32768 to 32767; use OPTION EXPLICIT		Learn to clean up recorded macro code, use Reddick naming convention or similar
Pivot table / Beginner	Create one-off standard Pivot tables		Learn some VBA to automate Pivot table usage (data feed, refresh)
Charts / Beginner	Create one-off standard charts		Learn from Peltier's website https://peltiertech.com/Excel/Charts/chartvba.html
Worksheet functions / Intermediate	Look up columns to the left with INDEX(MATCH()), when to use Pivot tables or VBA code instead of worksheet formulas, avoid volatile formulas	Use INDIRECT, use formulas which exceed two rows in the formula editor	Learn about proper spreadsheet design, learn some VBA
VBA / Intermediate	Develop special-purpose code, use mainly simple file I/O		Learn when to apply a Pivot table and when worksheet formulas will suffice
Pivot table / Intermediate	Refresh them with VBA		
Charts / Intermediate	Use advanced chart features		Learn even more from Peltier's website https://peltiertech.com/Excel/Charts/chartvba.html
Worksheet functions / Expert	Profile sheets with https://jkp-ads.com/articles/performanceclass.asp	Use worksheet formulas for the sake of it, even if they show quadratic runtime	Learn VBA mate, learn it
VBA / Expert	Add-in usage like Fincad, use classes where reasonable, develop general purpose (re-usable) code		Learn to understand and to develop classes - see SystemState (page 17) and Logging (page 20)
Chart / Expert	Auto-update with worksheet formulas or VBA,		

Excel Area / Level of expertise	What you should know	Leading others up the garden path	Areas or levels you might want to develop into
VBA / Wizard	Program classes - see SystemState (page 17) and Logging (page 20)		Don't get stuck with flat file I/O. Learn to read & to write from / to databases
Excel Connectivity / Wizard	Use database connections efficiently		
General / Wizard	Add-in development	Do not publish your source code, do not care about succession = do not ensure people can use your solutions when you will be gone.	

The graphic on the title page shows an example of a possible Excel learning path.

Table of Contents

Excel / VBA – A Collection 1
Abstract 2
Excel Learning Path 2
The Excel / VBA Programming Environment 11
Abstract 11
Basics 11
Variables 11
Types 11
Scope 12
During Editing 12
During Program Execution 13
Error Handling 14
Good Programming Practices 15
Be a Good Programmer 15
Good Excel and VBA Knowledge 15
Programming Conventions 15
Clean Up Macro Recordings 15
Document Your Program Adequately 15
Test Your Program Thoroughly 16
Log Your Program Execution 16
Optimize Your Program 16
System Status Save and Restore – SystemState Class 17
System Status Variables 17
SystemState Code 18
Documenting Program Flow – *Logging* Class 20
Pros and Cons 20
Parameters 21
Sample Output 22
Modules 22
Class Modules 26
Excursus: Logger for PowerShell – Write-Log 27
Export a Range as Picture – *sbExportRange2Picture* 29
List Named Ranges – *sbNamedRanges* 32
Show Excel Version – *ApplicationVersion* 33
Number of Dimensions of an Array – *ArrayDim* 34

Retrieve Cell Information – *sbGetCell* ... 34
Next Floating Point Number – *sbNextFloat* ... 40
Calling Other Windows Programs Using the Example *sbZip* ... 40
Excel Don't's – What You Should not be Doing with Excel ... 42
A table of what you should avoid ... 42
Number Systems, Formats, and Transformations ... 43
Abstract ... 43
Transformations and Calculations of Numbers ... 43
Spell Numbers in English Words – *sbSpellNumber* ... 43
Convert Decimal into Binary Numbers – sbDec2Bin / sbBin2Dec ... 47
Identify German Bank Holidays – *IstFeiertag* ... 52
Present the Full-Length Number – *sbNum2Str* ... 56
Significant Digits of a Number – *sbNSig* ... 57
Return the Number for a Month's Name – *sbMonthNumber* ... 58
Calculation of the Circle Constant π ... 61
First 1,000 Digits of π ... 63
Calculation of Euler's Number *e* ... 64
First 1,000 Digits of e ... 65
Literature ... 66
Shorten a Number Sequence Representation – sbParseNumSeq ... 66
Rational Numbers = Fractions ... 68
Nearest Rational Number to a Given Floating Point Number – *sbNRN* ... 68
Linear Equations with Rational Coeffizients ... 71
Present Quota Changes as Fractions ... 74
Monthly Fractions ... 75
Linear Combination of Integers ... 76
Extended Euklidean Algorithm – *sbEuklid* ... 76
Time Representations ... 78
Calculate Working Hours Between Two Time Points – *sbTimeDiff* ... 78
Add Working Hours to a Time Point – *sbTimeAdd* ... 81
Convert a Time to a Different Time Zone – *ConvertTime* ... 84
Check Digits ... 84
Calculate or Check a European Article Number – *sbEAN* ... 84
Ordinal Numbers ... 85
Simple Math with Formulas ... 86
How to Analyze Spreadsheet Formulas ... 86

Testing Date Formulas ... 87
First or last Weekday of a Month ... 89
Same Weekday and Calendarweek last year ... 90
Budget Control ... 91
Increment Lowest Significant Digit ... 92
Linear Breakdown ... 93
Merge two columns justified into one ... 96
Minimum Truck Load Problem ... 97
Count Trailing Zeros ... 98
Minimax Interpolation ... 98
Interpolation with PERCENTILE / PERCENTRANK ... 100
Interpolation with TREND ... 102
REFA Time Slicing ... 103
Roles and Rights ... 104
Rounding is Fun ... 105
Tip Distribution ... 106
Smoothing Inner Period Values ... 108
Cell Based Charts ... 109
Simple VBA Programs ... 111
Abstract ... 111
Sum up Numbers with same Number Format – *sbSumMyFormat* ... 111
Count Cells with Certain Color – *sbCountMyColor* ... 112
Allocate Assets to a Portfolio – *sbAllocate* ... 113
Eliminate Outliers – sbORB ... 114
Fair Distribution of a Limited Budget – sbDistBudget ... 116
Compute Collatz Length – sbCollatz ... 118
Rank Item Uniquely – sbUniqRank ... 119
Eliminate Points of a Graph with Small Slope Changes – sbReducePoints ... 121
Birthday List – sbBirthdayList ... 123
Accumulated Trade Blotter – sbAccumulatedTadeBlotter ... 125
Most Frequent Pairs – sbMostFrequentPairs ... 127
Interpolate – sbInterp ... 129
Combinations with Subsets k of n ... 131
Lookup Variants ... 133
Minimal Number of Banknotes and Coins – sbMinCash ... 136
Rebalance AssetClass Weights of a Portfolio – *sbRebalancedReturn* ... 139

Optimal Pitstops 141
Optimal Usage of Vacation Days 143
Create a Round Robin Tournament – sbRondRobin 145
Further Reading 145
Test Access Rights 148
Advanced VBA Programs 150
Abstract 150
Keeping Track of extreme Cell Values – sbCellWatermarks 150
A Task List – *sbTaskList* 152
Accounts Receivable Problem 155
Data Analysis – sbDatastats 158
System Handbook 158
Overview 158
Parameters in Sheet Param 159
User Handbook 159
Summary 159
Config File FileSpecs.csv 160
Numstats Output 160
Numstats Move Output 161
Textstats Output 161
Textstats Move Output 161
Output Limits File 162
Output Limits Move File 163
Mini Calculator 192
Mortality Annuities 199
Complex Array Formula (Worst) 199
Simple user defined function in VBA (Better) 200
Pre-calculated Table plus an NPV Formula (Maybe Best) 200
Financial Mathematics – Options 201
The binomial Method 201
Trinomial Trees and Finite Difference Methods 207
Monte Carlo Simulation 212
Generate all Permutations of an Array – *Quickperm* 222
Weight Calculation 224
Reasonable Extensions and Generalisations 226
A maintenance-free Database 233

Limitations ... 233
Responsibilities ... 233
System Documentation ... 234
User Documentation ... 234
Super User with Read/Write Access ... 234
Normal User with Read-Only Access ... 234
Appendix – SQL Code ... 235
Appendix – VBA Code ... 238
Excursus: Compare Correlation Matrices ... 241
Abstract ... 241
Implemention Approach ... 241
Parameters ... 241
Rounding Values Preserving Their Sum with *RoundToSum* ... 255
Abstract ... 255
Rounding Values Preserving Their Sum ... 255
Percentage Example ... 255
Example with Absolute Values ... 256
The User-Defined VBA Function *RoundToSum* ... 256
RoundToSum Program Code ... 257
Round2Sum Lambda Expression ... 258
Rounding Values Alters Their Sum ... 259
Usage Examples of *RoundToSum* ... 261
Allocation of Overheads ... 261
Example of an Exact Relation of Random Numbers ... 263
The User-Defined VBA Function sbExactRandHistogrm ... 264
Fair Staff Selection Based on Team Size – sbFairStaffSelection ... 266
Distribute a Sample Normally ... 268
Distribution of Budgets Among Remaining Staff ... 273
A Simple Approach ... 273
A Correct Calculation ... 273
Take Vacation When Less is Going on ... 274
Simple Example ... 274
More Complex Example ... 275
Assign Work Units Adjusted by Delivered Output ... 276
RoundToSum Versus Other Methods ... 277
RoundToSum Versus Other "Simple" Methods ... 277

RoundToSum Compared to the D'Hondt Approach 280
Literature 280
Random Number Generation (Excel / VBA) 281
Abstract 281
Random Integers 281
Natural Random Numbers – *UniqRandInt* 281
Random Integers – *sbRandInt* 283
Random Numbers with a Specified Sum 285
Minimum of Random Numbers given – sbLongRandSumN 285
Minimum and Maximum of Random Numbers given – sbRandIntFixSum 286
Usage Examples for Random Integers 288
Krabat, the Satanic Mill – How old can the apprentices become? 288
Generate a Math Test with Random Integer Inputs – Generate_Math_Test 289
Monte Carlo Simulation to Generate Teams Fairly – *sbGenerateTeams* 291
Monte Carlo Simulation for a Regatta Flight Plan – *sbRegattaFlightPlan* 295
Chances at Board Game Risk 299
A Simple Monte Carlo Simulation 303
Random Floating Point Numbers 305
Generate an Ideal Normal Distribution – *sbGenNormDist* 305
Generate Random Numbers with a Sum of 1 – *sbRandSum1* 307
Distributions of Random Floating Point Numbers 309
sbRandGeneral 309
sbRandHistogrm 312
sbRandTriang 315
sbRandTrigen 316
sbRandCauchy 320
sbRandCDFInv 321
sbRandPDF 322
sbRandCumulative 323
Brownian Bridges 325
sbGrowthSeries 325
Fix Sum from Random Corridors 327
Correlated Random Numbers 329
Cholesky Decomposition 329
Iman-Conover Method 331
Practical Applications of General Random Numbers 338

Generating Test Data – *sbGenerateTestData* 338
Random Numbers which do not reappear too soon – sbRandomNoRepeatBeforeN 348
Excursus 350
Calculating Probabilities – Drawing Cards With and Without Replacement 350
Fun without Practical Relevance for Advanced Learners 352
A Simple VBA Pivot Table – *sbMiniPivot* 352
TEXTJOIN 355
Index 356

The Excel / VBA Programming Environment

Abstract

With Visual Basic for Applications (VBA), tasks can be automated in the spreadsheet Excel, and special functionalities can be programmed that are not included in Excel's standard features.

Here, I will show programs that I found useful when I had to plan, implement, and execute business processes.

Basics

Variables

Variables are very useful to store temporary data during execution of your code. You can store different types of data.

Types

Type	Name Examples	Value Examples	Comment
Boolean	bProcessed	True, False	Can only have value False (0, Zero) or True (-1).
Byte	btChar	&H0F, 253	Value range 0 – 255.
Currency	ccyEUR	123.01	Value range -922,337,203,685,477.5808 to 922,337,203,685,477.5807.
Date	dtBirthday	#12/31/1980#	Value range 1 January 1900 to 31 December 9999 and times from 0:00:00 to 23:59:59.
Decimal	decDist	123789.12	With a scale of 0 (no decimal places), the largest possible value is +/-79,228,162,514,264,337,593,543,950,335. With a 28 decimal places, the largest value is +/-7.9228162514264337593543950335 and the smallest, non-zero value is +/-0.0000000000000000000000000001.
Double	dPi	3.14159265	-1.79769313486231E308 to -4.94065645841247E-324 for negative values and from 4.94065645841247E-324 to 1.79769313486232E308 for positive values.
Integer	intAge	123	Value range -32,768 to 32,767.
Long	lIndex	17167	Value range -2,147,483,648 to 2,147,483,647.
LongLong	llBigNum	123152367765	Only available for 64 bit systems: value range -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
LongPtr	lptr	131313123	Signed 32-bit (4-byte) numbers ranging in value from -2,147,483,648 to 2,147,483,647 on 32-bit systems; and signed 64-bit (8-byte) numbers ranging in value from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 on 64-bit systems.
Object	obj	A Range	Use the **Set** keyword to set the object reference to the variable.
Single	sngNum	1.2345	Value range -3.402823E38 to -1.401298E-45 for negative values and from 1.401298E-45 to 3.402823E38 for positive values.
String	sName	"Bernd"	Codes for String characters range from 0–255.
User defined	m_SystemState		Any data type that you define by using the **Type** statement. User-defined data types can contain one or more elements of a data type, an array, or a previously defined user-defined type.
Variant	vA	True, 123.01, #12/31/1980#,	Value range -1.797693134862315E308 to -4.94066E-324 for negative values and from 4.94066E-324 to 1.797693134862315E308 for positive values. Generally, numeric Variant data is maintained in

Type	Name Examples	Value Examples	Comment
		"Bernd", m_SystemState	its original data type within the Variant. For example, if you assign an Integer to a Variant, subsequent operations treat the Variant as an Integer. However, if an arithmetic operation is performed on a Variant containing a Byte, an Integer, a Long, or a Single, and the result exceeds the normal range for the original data type, the result is promoted within the Variant to the next larger data type. A Byte is promoted to an Integer, an Integer is promoted to a Long, and a Long and a Single are promoted to a Double. An error occurs when Variant variables containing Currency, Decimal, and Double values exceed their respective ranges.

Scope

All variables and all constants have a scope which defines the area of code where the variables will be recognised. This scope can be:

Level	Keyword	Location	Description
Global	Public	Top of a module	A public module level variable is visible to all modules in a project.
Module	Private	Top of a module	A private module level variable is visible to the module it is declared in.
Procedure (Sub, Function)	Dim	At the beginning of a Sub or of a Function	A local procedure level variable is visible to the Sub or Function it is declared in.

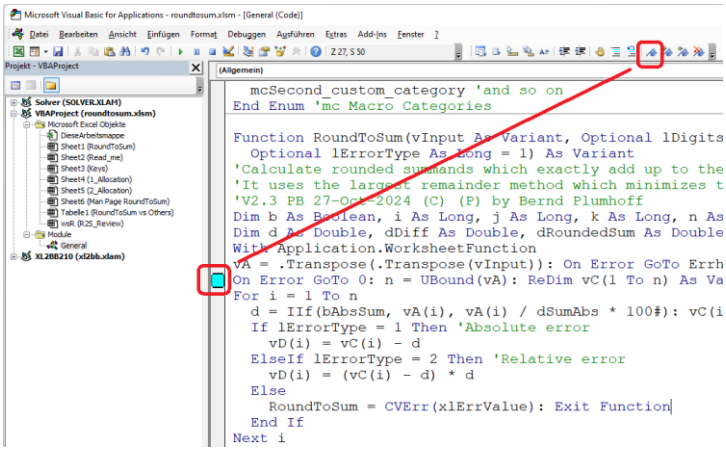
A variable life can be extended by the word STATIC instead of PUBLIC or DIM. In this case the variable value would be stored between calls. For constants the keyword PUBLIC or PRIVATE is followed by CONST or it is replacing the keyword DIM.

During Editing

With `Option Explicit` at the beginning of any VBA module you enforce the declaration of all variables used. If you do not use this you should not program.

When the cursor is located at a variable or at an object during editing you can go to its declaration with t SHIFT plus the function key F2 (SHIFT + F2). This works for Dim, not ReDim. With STRG + SHIFT + F2 you can jump back to where you were coming from.

In the VBA editor you can mark a code line with the "blue flag" symbol. With the adjacent flag pointer symbols you can then jump from one to the next or to the previous flag mark:



The VBA command `Stop` or a breakpoint set to the left of a code line with stop the execution of a program:

```
Sub Logging_Sample()
Dim i As Long

If GLogger Is Nothing Then Start_Log
'Initialize logger for this subroutine
GLogger.SubName = "Logging_Sample"

'Just do something to give log message examples
i = 2
Do While Not IsEmpty(wsData.Cells(i, 1))
  Select Case i
  Case Is < 6
    Stop
    GLogger.info i & " is a number less than 6"
  Case Is < 9
    Call Logging_Warn(i)
  Case Else
    Call Logging_Fatal(i)
  End Select
  i = i + 1
Loop
```

You can then examine the contents of variables or of Excel sheets, for example.

During Program Execution

You can interrupt the execution of a VBA program by pressing the ESC key, by the command `Stop` , by a breakpoint or by defining an interrupt condition such as a variable value. When the execution is interrupted the corresponding code line is marked yellow:

```
Sub Logging_Sample()
Dim i As Long

If GLogger Is Nothing Then Start_Log
'Initialize logger for this subroutine
GLogger.SubName = "Logging_Sample"

'Just do something to give log message examples
i = 2
Do While Not IsEmpty(wsData.Cells(i, 1))
  Select Case i
  Case Is < 6
    GLogger.info i & " is a number less than 6"
  Case Is < 9
    Call Logging_Warn(i)
  Case Else
    Call Logging_Fatal(i)
  End Select
```

Überwachungsausdrücke

Ausdruck	Wert	Typ	Kontext
i	2	Long	General.Logging_Sample

Direktbereich

```
? i
 2
```

You can now move the cursor to the variable *i* at the point "i = 2" (do not click, just hover over it). A small window showing i = 2 will appear. You can also select i, right-click, and add a watch for i. Then, the value of i will be displayed in the Watch Expressions window.

With CTRL + g, after a program break, you can go to the Immediate Window. Here, you can enter individual program commands, such as "Print i" (or simply "? i" — the question mark is shorthand for the Print command).

After a program break, you can continue running the program with the F5 function key. However, you can also step through it one command at a time using F8. If you use SHIFT + F8 instead of F8, the program will not jump into subroutines but will execute them as a single command.

Error Handling

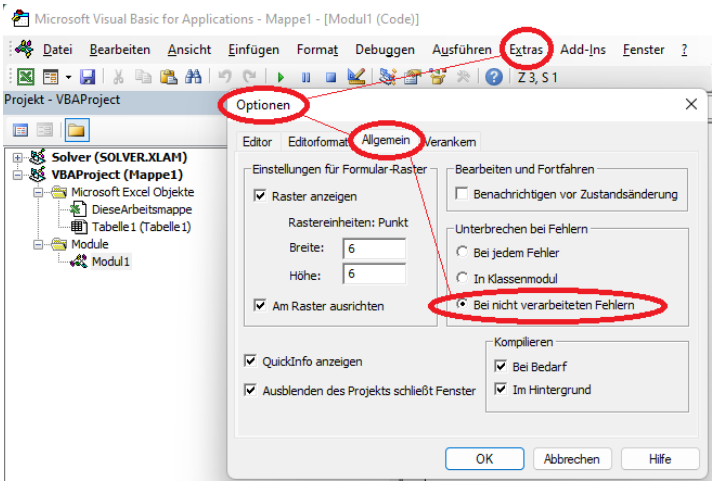
Your Excel VBA program does not necessarily have to stop in case of an error. You can catch errors using "On Error GoTo <Label>" or you can simply ignore all errors with "On Error Resume Next."

A good introduction can be found on Chip Pearson's site:
http://www.cpearson.com/excel/errorhandling.htm

A slightly more complex practical example can be found in my custom function sbMiniPivot. Since we cannot know the exact size of the output range, we start with a smaller value that we increase if necessary and then reduce to the exact required size at the end of the program using `ReDim Preserve` – `ReDim Preserve` allows you to change the last dimension of an array without deleting the existing elements:

```
…
lvdim = UBound(vInput(0))
If lvdim > 100 Then lvdim = 100 'Let us start with small dimension
On Error GoTo ErrHdl
…
'Reduce result array to used area
If k > 0 Then ReDim Preserve vR(0 To UBound(vInput) + liscount, 1 To k)
…
Exit Function

ErrHdl:
If Err.Number = 9 Then
  If i > lvdim Then
    'Here we normally get if we breach Ubound(vR,2)
    'So we need to increase last dimension
    lvdim = 10 * lvdim
    If lvdim > UBound(vInput(0)) Then lvdim = UBound(vInput(0))
    ReDim Preserve vR(0 To UBound(vInput) + liscount, 1 To lvdim)
    Resume 'Back to statement which caused error
  End If
End If
'Other error - terminate
On Error GoTo 0
Resume
```

An important note: Make sure that your options in VBA are set to "Break on Unhandled Errors"! If "Break on All Errors" is selected, your VBA error handling "On Error ..." will simply be ignored – Excel will stop with an error message for every error.



"Break in Class Module" is a reasonable option if you are looking for errors or testing error corrections in your class modules. If this option is not selected, Excel will typically not show an error in the class module but will return with an error message to the calling code. If this option is set, Excel will stop at the error in the class module and show the error message there. Unfortunately, Excel does not have a feature like MS Access's Application.GetOption("Error Trapping"). Mark Lundberg published the trick

```
Application.SendKeys "%{F11}%TO+{TAB}{RIGHT 2}%E~%{F4}"
```

but I recommend simply being mindful and attentive in a natural way.

Good Programming Practices

Be a Good Programmer

The most important aspect of a good VBA program is that it is a good program, not just full of VBA tricks. If you don't know associative arrays or classes, you'll be crawling on the floor and never get your application off the ground.

Good Excel and VBA Knowledge

You should have a strong grasp of Excel and VBA. For example, with the VBA command Enum, you can assign names to spreadsheet columns. This makes it easier to modify the program—unless you want to adjust all the hard-coded references to columns when a new column is inserted or an old one is deleted?

Programming Conventions

Learn naming conventions and coding conventions. By following them, you will make your programs more readable, easier to maintain, and more reliable and efficient.

Example links:
https://learn.microsoft.com/en-us/dotnet/visual-basic/programming-guide/program-structure/naming-conventions
https://learn.microsoft.com/en-us/dotnet/visual-basic/programming-guide/program-structure/coding-conventions

Clean Up Macro Recordings

Of course, I will record a macro if I forget a VBA command. But if you use raw, uncleaned spaghetti code from a recorded macro, someone else will have to clean it up for you.

Document Your Program Adequately

Explain what a knowledgeable third party needs to know, but don't write novels about trivialities. Good documentation comes with the program—not after. Only fools and people trying to make themselves indispensable fail to document their work. I use to indicate the quality of my macros with a version number:

- 0.01 is a play tool prototype
- 0.1 is the first usable version, module tested
- 1.0 is good for general usage

Some simple version control informs your users of what has been changed or corrected, and to quickly decide which of two copies is the newer one. See, for example module LoggerFactory:

```
'This general module is named LoggerFactory. Together with class module Logger it offers
'logging functionality. Requires LibFileTools: https://github.com/cristianbuse/VBA-FileTools
'Version When         Who             What
'      1 Once upon .. Cliff G.        Initial version
'     14 01-Feb-2025  Bernd Plumhoff  sPath corrected
```

Test Your Program Thoroughly

Applications of a certain size require test programs or even a series of regression tests.

Log Your Program Execution

With a Logger class (document the program flow—logging class), you can show an auditor who ran your program, with what parameters, and whether your program ran smoothly.

Optimize Your Program

The quality of your application is largely determined by its design and structure. The more complex the task, the better your expertise and experience should be. By measuring the runtime of individual parts of the program (profiling), you can identify where improvements are still possible.

Jan Karel Pieterse has created a helpful class for profiling:
https://jkp-ads.com/Articles/performanceclass.asp

In his module modTimer, I recommend replacing the function dMicroTimer with:
https://learn.microsoft.com/en-us/previous-versions/office/developer/office-2010/ff700515(v=office.14)?redirectedfrom=MSDN
This function MicroTimer must then be renamed to dMicroTimer.

System Status Save and Restore – SystemState Class

My former colleague Jon T. developed the smallest meaningful VBA class I know: With *SystemState*, you can easily save system status variables and optimize them for your own purposes.

To speed up program execution, you usually write the following at the beginning of a VBA macro:

```
Application.Calculation = xlCalculationManual
Application.ScreenUpdating = False
```

and at the end of the macro:

```
Application.Calculation = xlCalculationAutomatic
Application.ScreenUpdating = True
```

With the *SystemState* class, you only need to write the following at the start:

```
Dim state As SystemState
Set state = New SystemState
'Warning: "Dim state as New SystemState" does NOT create a new instance.
```

and at the end:

```
Set state = Nothing 'Not even necessary - this is done automatically.
```

System Status Variables

The *SystemState* class saves and restores the following system status variables:

Variable	Status	Comment / To Optimize By …
Calculation	xlCalculationAutomatic, xlCalculationManual, xlCalculationSemiautomatic	Determines if recalculation is performed after each cell change. Set to xlCalculationManual.
Cursor	xlDefault, xlBeam, xlNorthwestArrow, xlWait	This is just a display. It's best not to touch it – unless you want to start the debug mode with an hourglass cursor.
DisplayAlerts	True, False	Set to *False* to turn off system prompts.
EnableAnimations	True, False	From Excel 2016 onwards, this disables Excel's screen animations.
EnableEvents	True, False	Set to *False* to prevent event procedures from executing.
Interactive	True, False	Best not to touch – unless you want to prevent all keyboard inputs.
PrintCommunication	True, False	Set to *False* to change page settings without waiting for a response from the printer.
ScreenUpdating	True, False	Set to *False* to turn off screen updates during program execution.
StatusBar	False, "Any helpful user information"	The text is displayed in the status bar (bottom of the screen). Set to *False* to clear the display.

SystemState Code

Please copy the following program code into a class module named *SystemState*, not into a regular module.

```
'This class has been developed by my former colleague Jon T.
'I adapted it to newer Excel versions. Any errors are mine for sure.
'(C) (P) by Jon T., Bernd Plumhoff 3-Nov-2024 PB V1.5
'
'The class is called SystemState.
'It can of course be used in nested subroutines.
'
'This module provides a simple way to save and restore key excel
'system state variables that are commonly changed to speed up VBA code
'during long execution sequences.
'
'Usage:
'    Save() is called automatically on creation and Restore() on destruction
'    To create a new instance:
'        Dim state as SystemState
'        Set state = New SystemState
'    Warning:
'        "Dim state as New SystemState" does NOT create a new instance
'
'
'    Those wanting to do complicated things can use extended API:
'
'    To save state:
'       Call state.Save()
'
'    To restore state and in cleanup code: (can be safely called multiple times)
'       Call state.Restore()
'
'    To restore Excel to its default state (may upset other applications)
'       Call state.SetDefaults()
'       Call state.Restore()
'
'    To clear a saved state (stops it being restored)
'       Call state.Clear()
'
Private Type m_SystemState
  Calculation As XlCalculation
  Cursor As XlMousePointer
  DisplayAlerts As Boolean
  EnableAnimations As Boolean   'From Excel 2016 onwards
  EnableEvents As Boolean
  Interactive As Boolean
  PrintCommunication As Boolean 'From Excel 2010 onwards
  ScreenUpdating As Boolean
  StatusBar As Variant
  m_saved As Boolean
End Type

'
'Instance local copy of m_State?
'
Private m_State As m_SystemState

'
'Reset a saved system state to application defaults
'Warning: restoring a reset state may upset other applications
'
Public Sub SetDefaults()
  m_State.Calculation = xlCalculationAutomatic
  m_State.Cursor = xlDefault
  m_State.DisplayAlerts = True
  m_State.EnableAnimations = True
  m_State.EnableEvents = True
  m_State.Interactive = True
  On Error Resume Next 'In case no printer is installed
  m_State.PrintCommunication = True
  On Error GoTo 0
  m_State.ScreenUpdating = True
  m_State.StatusBar = False
  m_State.m_saved = True 'Effectively we saved a default state
End Sub

'
'Clear a saved system state (stop restore)
'
Public Sub Clear()
    m_State.m_saved = False
End Sub

'
```

```
'Save system state
'
Public Sub Save(Optional SetFavouriteParams As Boolean = False)
  If Not m_State.m_saved Then
    m_State.Calculation = Application.Calculation
    m_State.Cursor = Application.Cursor
    m_State.DisplayAlerts = Application.DisplayAlerts
    m_State.EnableAnimations = Application.EnableAnimations
    m_State.EnableEvents = Application.EnableEvents
    m_State.Interactive = Application.Interactive
    On Error Resume Next 'In case no printer is installed
    m_State.PrintCommunication = Application.PrintCommunication
    On Error GoTo 0
    m_State.ScreenUpdating = Application.ScreenUpdating
    m_State.StatusBar = Application.StatusBar
    m_State.m_saved = True
  End If
  If SetFavouriteParams Then
    Application.Calculation = xlCalculationManual
    Application.DisplayAlerts = False
    Application.EnableAnimations = False
    Application.EnableEvents = False
    On Error Resume Next 'In case no printer is installed
    Application.PrintCommunication = False
    On Error GoTo 0
    Application.ScreenUpdating = False
    Application.StatusBar = False
  End If
End Sub

'
'Restore system state
'
Public Sub Restore()
  If m_State.m_saved Then
    'We check now before setting Calculation because setting
    'Calculation will clear cut/copy buffer
    If Application.Calculation <> m_State.Calculation Then
      Application.Calculation = m_State.Calculation
    End If
    Application.Cursor = m_State.Cursor
    Application.DisplayAlerts = m_State.DisplayAlerts
    Application.EnableAnimations = m_State.EnableAnimations
    Application.EnableEvents = m_State.EnableEvents
    Application.Interactive = m_State.Interactive
    On Error Resume Next 'In case no printer is installed
    Application.PrintCommunication = m_State.PrintCommunication
    On Error GoTo 0
    Application.ScreenUpdating = m_State.ScreenUpdating
    If m_State.StatusBar = "FALSE" Then
      Application.StatusBar = False
    Else
      Application.StatusBar = m_State.StatusBar
    End If
  End If
End Sub

'
'By default save when we are created
'
Private Sub Class_Initialize()
  Call Save(SetFavouriteParams:=True)
End Sub

'
'By default restore when we are destroyed
'
Private Sub Class_Terminate()
  Call Restore
End Sub
```

Documenting Program Flow – *Logging* Class

This Logger class provides logging with the report levels *INFO, WARN, FATAL*, and *EVER*. Program information is logged both in a worksheet and in a file.

The application of this Logger class is not difficult: Simply copy the general module *Logger_Factory* and the class module *Logger* from the example file provided below into your own application, then define the public constant *AppVersion*, for example with the value "My Application Version 1.0" in the main module. After that, you can generate your log messages with the following commands:

```
GLogger.info "Info Meldung ..."
GLogger.warn "Warn Meldung ..."
GLogger.fatal "Fehler Meldung ..."
GLogger.ever "Nichtunterdrückbare Standard Meldung ..."
```

These log messages will be automatically saved in the worksheet Workflow and in the log file "My Application Version 1.0_Logfile_yyyymmdd.log" in the subdirectory *Logs*.

I received the original program code from Cliff G. in 2009 and later extended it. Cliff primarily used this program for debugging. I also find it very useful for logging program executions for revision purposes or to have a program explain its individual execution steps to the user in detail. Additionally, I added version information and system or Excel parameters to quickly identify important differences between different user environments. With this logger, I also typically measure simple runtime durations of SQL queries:

```
'Glogger is declared in module LoggerFactory and set in Sub Start_Log()
Dim dtStamp As Date
'...
dtStamp = Now
'Retrieve data from database here
Glogger.info "SQL xxx ran " & Format(Now - dtStamp, "n:ss") & " [m:ss]"
```

Pros and Cons

In my opinion, this logging program provides the most useful secondary functionality for any VBA application. With it, you can:

- Test in a traceable manner
- Have a program explain all its execution steps in a traceable way
- Easily determine if multiple users are accessing an application simultaneously
- Quickly identify if a user issue is related to a different environment
- Systematically narrow down sporadic application errors
- Convincingly demonstrate to auditors the correct, error-free usage of the program over an extended period (while individual log files can be tampered with, a large number of log files still provides sufficient evidence)
- Roughly determine the runtime of VBA (sub)routines
- Measure the runtime of entire processes

The last point above will raise concerns for works councils and employee representatives:

- When measuring the runtime of entire processes, individual employee performance could be assessed, compared, and potentially used against them.

This would be a clear violation of the GDPR (General Data Protection Regulation), see https://gdpr-info.eu/.

I have never used this logging to measure employee performance or usage, but only for retraining when I identified incorrect usage. However, this should not be taken as an argument for uncritical use.

In my opinion, the approval of works councils or employee representatives can always be obtained by emphasizing the voluntary nature of this self-logging:

- Each user can turn logging on or off before running the program.
- Each user can delete log files at any time afterward.

I have successfully used this logging in multiple countries in Europe (UK, Germany) and with multiple companies (banks, insurance companies, IT providers) without any complaints.

Parameters

Public Constants

AppVersion - This string should contain the program name and its version, e.g.:

```
Public Const AppVersion As String = "... Version x"
```

Then "… Version x" will be logged as version information for this application.

Compiler Constants

Separate_Log_Files_for_each_User - True = Separate daily log files for different users, False = One daily log file for all users

Use_Logger_auto_Open_Close - True will use subs auto_open and auto_close of LoggerFactory, False will not

Logging_on_Screen - Set to True in both LoggerFactory and Logger if you want to log messages also to sheet Workflow (i. e. on screen). Note: sheet Workflow's codename (its internal VBA name) has to be wsW

Logging_cashed - Set to True in both LoggerFactory and Logger if you want to speed up the application by writing log messages in one go to the log file at program end. This requires Logging_on_Screen set to True.

Log_WMI_Info - Set to True in LoggerFactory if you like to log interesting Windows Management Instrumentation (WMI) information such as processor, memory, disk, and operating system data.

Show_Reference_Details - True = Show all details, False = Just show description.

Logging Variables

LogFilePath - Full pathname of log file

SubName - Set at the beginning of each subroutine to enable the logger to report on the right subroutine name

LogLevel - The level for which logging should be performed:

1 - Report all log messages: INFO, WARN, FATAL, and EVER
2 - Report all log messages but not INFOs
3 - Report from FATAL level onwards, i. e. just FATAL and EVER messages
4 - Report only EVER messages
5 - Switch off logging

LogScreenRow - Row from where to start logging in sheet Workflow (usually 3).

Sample Output

Standard output for the logs are sheet Workflow the logfile in subfolder Logs:

```
EVER: BERND-CAPTIVA\earso 12.11.2024 03:57:15 [Start_Log] - Logging started with Logging_Version_13
EVER: BERND-CAPTIVA\earso 12.11.2024 03:57:15 [Start_Log] - Microsoft Windows 11 Home 10.0.22000 (64-Bit)
and Excel 2024 (64-Bit)
INFO: BERND-CAPTIVA\earso 12.11.2024 03:57:15 [Start_Log] - Application ThousandsSeparator '.',
DecimalSeparator ',', use system separators
INFO: BERND-CAPTIVA\earso 12.11.2024 03:57:15 [Start_Log] - App.Internl ThousandsSeparator '.',
DecimalSeparator ',', ListSeparator ';'
INFO: BERND-CAPTIVA\earso 12.11.2024 03:57:15 [Start_Log] - App.Internl xlCountryCode '49',
xlCountrySetting '49'
INFO: BERND-CAPTIVA\earso 12.11.2024 03:57:15 [Start_Log] - VBAProject References: Visual Basic For
Applications, Microsoft Excel 16.0 Object Library, OLE Automation, Microsoft Office 16.0 Object Library
EVER: BERND-CAPTIVA\earso 12.11.2024 03:57:18 [End_Log] - Logging finished with Logging_Version_13
```

Modules

Normal

LoggerFactory contains constants, public variables, default logger settings, and optional autoopen and autoclose subs.

Note: The subroutine Start_Log requires (calls) the subroutines *ApplicationVersion* and *getOperatingSystem* (https://www.devhut.net/vba-determine-the-installed-os/), and the module *LibFileTools* (https://github.com/cristianbuse/VBA-FileTools) to support folder in OneDrive and in Sharepoint.

```
Option Explicit
'This general module is named LoggerFactory. Together with class module Logger it offers logging
functionality.
'Requires LibFileTools: https://github.com/cristianbuse/VBA-FileTools
'Version When         Who             What
'      1 Once upon .. Cliff G.        Initial version
'     14 01-Feb-2025  Bernd Plumhoff  sPath corrected
#Const Separate_Logfiles_for_each_User = False
#Const Use_Logger_auto_Open_Close = True 'Enable auto_open and auto_close subs in here
#Const Logging_on_Screen = True          'IMPORTANT: Also change this constant in class module Logger! We
like to see recent run's loggging messages on screen in tab Workflow
#Const Logging_cashed = False            'IMPORTANT: Also change this constant in class module Logger!
Write logging messages into file at program end to speed this up
#Const Log_WMI_Info = False              'True shows interesting Windows Management Instrumentation (WMI)
data
#Const Show_Reference_Details = False    'True: Show all details; False: Just show description
Public GLogger As Logger                 'Global logfile object - variable scope is across all modules
Public GsThisLogFilePath As String
' Constant log levels
Public Const INFO_LEVEL As Integer = 1
Public Const WARN_LEVEL As Integer = 2
Public Const FATAL_LEVEL As Integer = 3
Public Const EVER_LEVEL As Integer = 4 'For logging messages which cannot be switched off
Public Const DISABLE_LOGGING As Integer = 5
'The application-specific defaults
Const DEFAULT_LOG_FILE_PATH As String = "" 'Force error if not set [Bernd 12-Aug-2009]
Const DEFAULT_LOG_LEVEL As Integer = INFO_LEVEL

Public Function getLogger(sSubName As String) As Logger
  Dim oLogger As New Logger
  oLogger.SubName = sSubName
  'Defaults to the specified values - but may be overridden before used
  oLogger.LogLevel = DEFAULT_LOG_LEVEL
  oLogger.LogFilePath = DEFAULT_LOG_FILE_PATH
  Set getLogger = oLogger
End Function

#If Use_Logger_auto_Open_Close Then
  Sub auto_open()
  'Version Date        Programmer Change
  '9       12-Sep-2021 Bernd      Code outsorced to Start_Log so that user does not need to use auto_open
  Start_Log
  End Sub

  Sub auto_close()
  'Version Date        Programmer Change
  '3       12-Sep-2021 Bernd      Code outsorced to End_Log so that user does not need to use auto_close
  End_Log
  End Sub
#End If '#If Use_Logger_auto_Open_Close

Sub Start_Log()
'Version Date        Programmer Change
'3       02-Nov-2023 Bernd      Log interesting Windows Management Instrumentation (WMI) infos
'4       27-Feb-2024 Bernd      Show_Reference_Details added
'5       12-Nov-2024 Bernd      Using LibFileTools https://github.com/cristianbuse/VBA-FileTools
Dim i As Long
Dim s As String, sDel As String, sPath As String
#If Log_WMI_Info = True Then
  Dim oWMISrvEx As Object 'SWbemServicesEx
  Dim oWMIObjSet As Object 'SWbemServicesObjectSet
  Dim oWMIObjEx As Object 'SWbemObjectEx
  Dim oWMIProp As Object 'SWbemProperty
  Dim sWQL As String 'WQL Statement
  Dim v As Variant
#End If
'LibFileTools are necessary just for the next 5 lines:
sPath = GetLocalPath(ThisWorkbook.Path)
If Right(sPath, 1) <> "\"  Then sPath = sPath & "\"
sPath = sPath & "Logs\"
If Not IsFolder(sPath) Then
  CreateFolder (sPath)
End If
If GLogger Is Nothing Then Set GLogger = New Logger
#If Separate_Logfiles_for_each_User Then
  'If AppVersion is not defined please define it in your main module like:
  'Public Const AppVersion As String = "Application Version ..."
  GLogger.LogFilePath = sPath & Environ("Userdomain") & _
    "_" & Environ("Username") & "_" & AppVersion & "_" & "Logfile_" & _
    Format(Now, "YYYYMMDD") & ".txt"
#Else
  GLogger.LogFilePath = sPath & AppVersion & "_" & _
    "Logfile_" & Format(Now, "YYYYMMDD") & ".txt"
#End If
GLogger.LogLevel = 1
#If Logging_on_Screen Then
  GLogger.LogScreenRow = 3
  wsW.Range("E2:E4").ClearContents
  wsW.Range("5:65535").Delete
#End If
```

```
'Initialize logger for this subroutine
With Application
GLogger.SubName = "Start_Log"
GLogger.ever "Logging started with " & AppVersion
#If Log_WMI_Info = True Then
  Set oWMISrvEx = GetObject("winmgmts:root/CIMV2")
  For Each v In Array("BaseService", "Processor", "PhysicalMemoryArray", "LogicalDisk", "OperatingSystem")
    'Not: "NetworkAdapterConfiguration", "VideoController", "OnBoardDevice", "Printer", "Product"
    Set oWMIObjSet = oWMISrvEx.ExecQuery("Select * From Win32_" & v)
    For Each oWMIObjEx In oWMIObjSet
      s = v & ": "
      For Each oWMIProp In oWMIObjEx.Properties_
        If Not IsNull(oWMIProp.Value) Then
          If Not IsArray(oWMIProp.Value) Then
            Select Case v
            Case "BaseService"
              If InStr("'SystemName'", "'" & oWMIProp.Name & "'") > 0 Then
                GLogger.ever oWMIProp.Name & "='" & Trim(oWMIProp.Value) & "'"
                GoTo Next_v
              End If
            Case "Processor"
              If
InStr("'Name'Description'NumberOfEnabledCore'AddressWidth'DataWidth'CurrentClockSpeed'LoadPercentage'", _
                "'" & oWMIProp.Name & "'") > 0 Then
                If IsNumeric(oWMIProp.Value) Then
                  s = s & oWMIProp.Name & "=" & Format(oWMIProp.Value, "#,##0") & ", "
                Else
                  s = s & oWMIProp.Name & "='" & Trim(oWMIProp.Value) & "', "
                End If
              End If
            Case "PhysicalMemoryArray"
              If InStr("'MaxCapacityEx'", _
                "'" & oWMIProp.Name & "'") > 0 Then s = s & oWMIProp.Name & "=" & Format(oWMIProp.Value,
"#,##0") & ", "
            Case "LogicalDisk"
              If InStr("'DeviceID'ProviderName'Size'FreeSpace'", _
                "'" & oWMIProp.Name & "'") > 0 Then
                If IsNumeric(oWMIProp.Value) Then
                  s = s & oWMIProp.Name & "=" & Format(oWMIProp.Value, "#,##0") & ", "
                Else
                  s = s & oWMIProp.Name & "='" & Trim(oWMIProp.Value) & "', "
                End If
              End If
            Case "OperatingSystem"
              If
InStr("'FreePhysicalMemory'FreeVirtualMemory'FreeSpaceInPagingFiles'MaxProcessMemorySize'InstallDate'", _
                "'" & oWMIProp.Name & "'") > 0 Then s = s & oWMIProp.Name & "=" & Format(oWMIProp.Value,
"#,##0") & ", "
            End Select
          End If
        End If
      Next oWMIProp
      If Len(s) > Len(v & ": ") Then GLogger.ever Left(s, Len(s) - 2)
    Next oWMIObjEx
Next_v:
  Next v
#End If
#If Win64 Then
  s = "64"
#Else
  s = "32"
#End If
GLogger.ever getOperatingSystem() & " and " & ApplicationVersion() & _
  " (" & s & "-Bit)" '& .Version & .Build & " (" & .CalculationVersion & ")"
GLogger.info "Application ThousandsSeparator '" & .ThousandsSeparator & _
  "', DecimalSeparator '" & .DecimalSeparator & "', " & _
  IIf(Not (Application.UseSystemSeparators), "do not ", "") & "use system separators"
GLogger.info "App.Internl ThousandsSeparator '" & .International(xlThousandsSeparator) & _
  "', DecimalSeparator '" & .International(xlDecimalSeparator) & "', ListSeparator '" & _
  .International(xlListSeparator) & "'"
GLogger.info "App.Internl xlCountryCode '" & .International(xlCountryCode) & _
  "', xlCountrySetting '" & .International(xlCountrySetting) & "'"
End With
With ThisWorkbook.VBProject.References 'In case of error tick box Trust access to the VBA project object
  'model under File / Options / Trust Center / Trust Center Settings / Macro Settings
  s = "VBAProject References: "
  On Error Resume Next
  For i = 1 To .Count
    #If Show_Reference_Details Then
      GLogger.info s
      s = ""
      s = s & .Item(i).Description
      s = s & ", FullPath: '" & .Item(i).fullPath & "'"
      s = s & ", Guid: " & .Item(i).GUID
      s = s & ", BuiltIn: " & .Item(i).BuiltIn
      s = s & ", IsBroken: " & .Item(i).IsBroken
      s = s & ", Major: " & .Item(i).Major
      s = s & ", Minor: " & .Item(i).Minor
    #Else
      s = s & sDel & .Item(i).Description
```

```
      sDel = ", "
    #End If
  Next i
  GLogger.info s
End With
'Now two examples of environment variables which might not exist for all Windows / Excel installations.
'Use Sub List_Environ_Variables below to see which variables exist on your system.
s = ""
s = Environ("CRC_VDI-TYPE") 'If this does not exist we will not log anything
If s <> "" Then GLogger.info "CRC_VDI-TYPE: '" & s & "'"
s = ""
s = Environ("ORACLE_HOME_X64") 'If this does not exist we will not log anything
If s <> "" Then GLogger.info "Oracle Client: '" & s & "'"
On Error GoTo 0
End Sub

Sub End_Log()
'Change History:
'Version Date        Programmer Change
'1       12-Sep-2021 Bernd      Initial version so that user does not need to use auto_close. He can
manually call this sub.
If GLogger Is Nothing Then Call auto_open
GLogger.SubName = "End_Log"
'If AppVersion is not defined please define it in your main module like: Public Const AppVersion As String
= "Application Version ..."
GLogger.ever "Logging finished with " & AppVersion
#If Logging_cashed Then
  Set GLogger = Nothing 'Necessary, or Class_Terminate() won't be called for GLogger because it's Public
#End If
End Sub
```

A sample module *General* which show how to use the logger:

```
Option Explicit
'Version When         Who             What
'     11 03-Nov-2023  Bernd Plumhoff  Log interesting Windows Management Instrumentation (WMI) infos
'     12 17-Feb-2024  Bernd Plumhoff  Show_Reference_Details added
'     13 12-Nov-2024  Bernd Plumhoff  Using LibFileTools https://github.com/cristianbuse/VBA-FileTools

Public Const AppVersion As String = "Logging_Version_13"

Sub Logging_Sample()
Dim i As Long

If GLogger Is Nothing Then Start_Log
'Initialize logger for this subroutine
GLogger.SubName = "Logging_Sample"

'Just do something to give log message examples
i = 2
Do While Not IsEmpty(wsData.Cells(i, 1))
    Select Case i
    Case Is < 6
        GLogger.info i & " is a number less than 6"
    Case Is < 9
        Call Logging_Warn(i)
    Case Else
        Call Logging_Fatal(i)
    End Select
    i = i + 1
Loop

#If Logging_cashed Then
Set GLogger = Nothing 'Necessary, or Class_Terminate() won't be called for GLogger since it's Public
#End If

End Sub

'You do not need extra subroutines to log warn messages or fatal messages.
'They are just examples of additional subroutines which do some logging.
Sub Logging_Warn(i As Long)
    'Initialize logger for this subroutine
    GLogger.SubName = "Logging_Warn"
    GLogger.warn i & " is 6, 7, or 8"
End Sub

Sub Logging_Fatal(i As Long)
    'Initialize logger for this subroutine
    GLogger.SubName = "Logging_Fatal"
    GLogger.fatal i & " is greater 8"
End Sub
```

Class Modules

Logger contains the logging functionality:

```
Option Explicit
'This class module is named Logger. Together with class module LoggerFactory it offers logging
functionality.
'Version When         Who             What
'      1 Once upon .. Cliff G.        Initial version
'     14 01-Feb-2025  Bernd Plumhoff  Same version as LoggerFactory
#Const Logging_on_Screen = True 'IMPORTANT: Also change this constant in module LoggerFactory! We like to
see recent run's loggging messages on screen in tab Workflow
#Const Logging_cashed = False   'IMPORTANT: Also change this constant in module LoggerFactory! Write
logging messages into file at program end to speed this up
Const INFO_LEVEL_TEXT As String = "INFO:"
Const WARN_LEVEL_TEXT As String = "#WARN:"
Const FATAL_LEVEL_TEXT As String = "##FATAL:"
Const EVER_LEVEL_TEXT As String = "EVER:"
Private sThisSubName As String
Private iThisLogLevel As Integer
#If Logging_on_Screen Then
  Private iThisLogRow As Integer
  Public Property Let LogScreenRow(iLogRow As Integer)
    iThisLogRow = iLogRow
  End Property

  Public Property Get LogScreenRow() As Integer
    LogScreenRow = iThisLogRow
  End Property
#End If

Public Property Let LogFilePath(sLogFilePath As String)
  GsThisLogFilePath = sLogFilePath
End Property

Public Property Get LogFilePath() As String
  LogFilePath = GsThisLogFilePath
End Property

Public Property Let SubName(sSubName As String)
  sThisSubName = sSubName
End Property

Public Property Get SubName() As String
  SubName = sThisSubName
End Property

Public Property Let LogLevel(iLogLevel As Integer)
  iThisLogLevel = iLogLevel
End Property

Public Property Get LogLevel() As Integer
  LogLevel = iThisLogLevel
End Property

Public Sub info(sLogText As String)
  If Me.LogLevel = LoggerFactory.INFO_LEVEL Then
    Call WriteLog(LoggerFactory.INFO_LEVEL, sLogText)
  End If
End Sub

Public Sub warn(sLogText As String)
  If Me.LogLevel < LoggerFactory.FATAL_LEVEL Then
    Call WriteLog(LoggerFactory.WARN_LEVEL, sLogText)
  End If
End Sub

Public Sub fatal(sLogText As String)
  If Me.LogLevel <= LoggerFactory.FATAL_LEVEL Then
    Call WriteLog(LoggerFactory.FATAL_LEVEL, sLogText)
  End If
End Sub

Public Sub ever(sLogText As String)
  If Me.LogLevel <= LoggerFactory.EVER_LEVEL Then
    Call WriteLog(LoggerFactory.EVER_LEVEL, sLogText)
  End If
End Sub
```

```
Private Sub WriteLog(iLogLevel As Integer, sLogText As String)
  Dim FileNum As Integer, LogMessage As String, sDateTime As String, sLogLevel As String
  Select Case iLogLevel
  Case LoggerFactory.INFO_LEVEL
    sLogLevel = INFO_LEVEL_TEXT
  Case LoggerFactory.WARN_LEVEL
    sLogLevel = WARN_LEVEL_TEXT
  Case LoggerFactory.FATAL_LEVEL
    sLogLevel = FATAL_LEVEL_TEXT
  Case LoggerFactory.EVER_LEVEL
    sLogLevel = EVER_LEVEL_TEXT
  Case Else
    sLogLevel = "!INVALID LOG LEVEL!"
  End Select
  sDateTime = CStr(Now())
  LogMessage = sLogLevel & " " & Environ("Userdomain") & "\" & Environ("Username") & " " & _
    sDateTime & " [" & Me.SubName & "] - " & sLogText
  #If Not Logging_cashed Then
    FileNum = FreeFile
    Open Me.LogFilePath For Append As #FileNum
    Print #FileNum, LogMessage
    Close #FileNum
  #End If
  #If Logging_on_Screen Then
    wsW.Cells(iThisLogRow, 5) = LogMessage
    iThisLogRow = iThisLogRow + 1
  #End If
End Sub

Private Sub Class_Initialize()
  #If Logging_cashed And Not Logging_on_Screen Then
    Err.Raise Number:=vbObjectError + 513, Description:="Logging_cashed requires Logging_on_Screen"
  #End If
End Sub

Private Sub Class_Terminate()
  #If Logging_cashed Then
    Dim i As Long, FileNum As Integer, LogMessage As String
    FileNum = FreeFile
    Open Me.LogFilePath For Append As #FileNum
    For i = 3 To iThisLogRow - 1
      LogMessage = wsW.Cells(i, 5).Text
      Print #FileNum, LogMessage
    Next i
    Close #FileNum
  #End If
End Sub
```

Excursus: Logger for PowerShell – Write-Log

PowerShell is a Microsoft Windows-based object-oriented scripting or programming language. It has its own command-line interface. A useful log function:

```
<#
.SYNOPSIS
    Write-Log -Message [string] -Severity EVER|INFO|WARN|FATAL

.DESCRIPTION
    This script/function enables logging within PowerShell scripts.

    Global variables which control features of this script are:
    [string]$global:LogFilePath    - The text file into which all messages are getting written
    [string]$global:SubName        - The name of the script calling this Log script
    [int]$global:LogLevel          - The log levels:
                                                                                                  1 logs
everything,
                                                                                                  2 omits
INFOs,
                                     3 omits INFOs and WARNings,
                                                                                                  4 only
logs severity EVER messages
    [bool]$global:WriteToHost      - $FALSE = Only write into logfile; $TRUE = Also write to Host

.OUTPUTS
           Write-Log writes into a logfile with name [string]$global:LogFilePath.
           if the user has not given this, the standard filename "$PSScriptRoot\Logs\LogFile.txt" is used.
    The contents of a log file will look like:
    INFO: BERND-PC\Bernd 2020-02-16 12:19:36 [zip_and_copy.ps1] - Log started
    INFO: BERND-PC\Bernd 2020-02-16 12:19:36 [zip_and_copy.ps1] - Log finished

           In case the user has set [bool]$global:WriteToHost to $TRUE, Write-Log also repeats all log
           messages on screen (Host).
```

```
.PARAMETER Message
    The message you want to log.

.PARAMETER Severity
    The severity of the message. This can be 'INFO', 'WARN', 'FATAL', or 'EVER'.
    If global variable LogLevel is 1 then messages with all severities will be logged.
    If LogLevel is 2 then INFO messages will not be logged.
    If LogLevel is 3 then INFO and WARN messages will not be logged.
    If LogLevel is 4 then INFO, WARN and FATAL messages will not be logged.

.EXAMPLE
    # Initialize global logging variables and make log script known
    $global:SubName = $MyInvocation.MyCommand.Name
    $global:LogLevel = 1
    $global:LogFilePath = "$PSScriptRoot\Logs\LogFile_" + $SubName + "_" +
        (Get-Date -UFormat "%Y%m%d") + ".txt"
    $global:WriteToHost = $TRUE
    . "$PSScriptRoot\Write-Log.ps1"

.EXAMPLE
    # How to call
    Write-Log -Message 'Log started' -Severity INFO
    If(-Not (Test-path $ArchiveFile)) {Write-Log -Message "$ArchiveFile was not created" -Severity FATAL}

.NOTES
    Version Date        Who                   What
    1.0     16-Feb-2020 Bernd Plumhoff        Initial Version
    1.1     13-Jul-2020 Bernd Plumhoff        EVER instead of ALWAYS, and optional Write-Host
#>

#Set-ExecutionPolicy -ExecutionPolicy Bypass -Scope CurrentUser -Force
#Clear
#Requires -Version 4
Set-StrictMode -Version Latest

function Write-Log {
    [CmdletBinding()]
    param(
        [Parameter()]
        [ValidateNotNullOrEmpty()]
        [string]$Message,
        [Parameter()]
        [ValidateNotNullOrEmpty()]
        [ValidateSet('INFO','WARN','FATAL','EVER')]
        [string]$Severity = 'INFO'
    )
    switch ($Severity)
    {
        'INFO'        { If ([int]$global:LogLevel -gt 1) {Exit} }
        'WARN'        { If ([int]$global:LogLevel -gt 2) {Exit} }
        'FATAL'       { If ([int]$global:LogLevel -gt 3) {Exit} }
        'EVER'        { <# Will always be logged! #> }
    }
    if (-Not (Test-Path -IsValid $global:LogFilePath)) { $global:LogFilePath =
"$PSScriptRoot\Logs\LogFile.txt" }
    [string]$Output = $Severity + ": " + [System.Security.Principal.WindowsIdentity]::GetCurrent().Name +
        " " + (Get-Date -UFormat "%Y-%m-%d %T") + " [" + $global:SubName + "] - " + $Message
    Write-Output $Output | Out-File -append $global:LogFilePath
           If (Test-Path variable:global:WriteToHost) {
                      If ($global:WriteToHost) { Write-Host $Output }
           }
}
```

Export a Range as Picture – *sbExportRange2Picture*

You need to export a range as a graphic? With this subroutine you can save ranges as bmp, gif, jpg, or png graphic files:

sbExportRange2Picture Program Code

```
Sub sbExportRange2Picture(sPath As String, _
  rngInput As Range, _
  Optional bHighResolution As Boolean = True)
'Export range as image
'(C) (P) by Bernd Plumhoff 08-Feb-2025 PB V4

Dim ch As ChartObject

rngInput.CopyPicture Appearance:=xlPrinter, Format:=xlPicture
Set ch = ActiveSheet.ChartObjects.Add(1, 1, rngInput.Width, rngInput.Height)
ch.Name = "TemporaryPictureChart"
ch.Width = rngInput.Width
ch.Height = rngInput.Height

ch.Activate
ActiveChart.Paste

If bHighResolution Then
  Dim pptApp As Object, pptPres As Object, pptSlide As Object, pptShape As Shape
  Dim j As Long
  Set pptApp = CreateObject("PowerPoint.Application")
  pptApp.Visible = True
  Set pptPres = pptApp.Presentations.Add
  Set pptSlide = pptPres.Slides.Add(1, 12)
  ActiveChart.CopyPicture Appearance:=xlScreen, Format:=xlPicture
  pptSlide.Shapes.Paste
  pptPres.PageSetup.SlideWidth = rngInput.Width
  pptSlide.Shapes(1).Width = rngInput.Width
  pptPres.PageSetup.SlideHeight = rngInput.Height
  pptSlide.Shapes(1).Height = rngInput.Height
  pptSlide.Shapes(1).Left = 0
  pptSlide.Shapes(1).Top = 0
  'File extension: gif, png, or jpg
  j = 6000000# / rngInput.Width / rngInput.Height
  pptPres.Export Path:=sPath, FilterName:=Right(sPath, 3), _
    ScaleWidth:=j * rngInput.Width, _
    ScaleHeight:=j * rngInput.Height
  pptPres.Close
  pptApp.Quit
  Set pptShape = Nothing
  Set pptSlide = Nothing
  Set pptPres = Nothing
  Set pptApp = Nothing
Else
  ActiveChart.Export fileName:=sPath, _
    FilterName:=Right(sPath, 3) 'File extension: bmp, gif, png, or jpg
End If
ch.Delete

End Sub
```

This way you can export ranges with or without column character headers or row numbers:

ExportAllPics Program Code

```
Sub ExportAllPics()
'Exports all named ranges as pictures into subfolder
'Pictures if their names start with "Pic_" (then we
'add column letters and row numbers) or with "Fml_".
'Requires LibFileTools:
'https://github.com/cristianbuse/VBA-FileTools
'(C) (P) by Bernd Plumhoff 08-Feb-2025 PB V0.1
Dim bWSadded    As Boolean
Dim i           As Long
Dim lMinC       As Long
Dim lMinR       As Long
Dim lMaxC       As Long
Dim lMaxR       As Long
Dim sFolder     As String
Dim v           As Variant
Dim vT          As Variant
Dim ws          As Worksheet
Dim state       As SystemState

Set state = New SystemState

'LibFileTools are necessary just for the next 5 lines:
sFolder = GetLocalPath(ThisWorkbook.Path)
If Right(sFolder, 1) <> "\" Then sFolder = sFolder & "\"
sFolder = sFolder & "Pictures\"
If Not IsFolder(sFolder) Then
  CreateFolder (sFolder)
End If

For Each v In ActiveWorkbook.Names
  Select Case Left(v.Name, 4)
  Case "Pic_"

    If Not bWSadded Then
      Set ws = ThisWorkbook.Worksheets.Add
      bWSadded = True
    End If
    v.RefersToRange.Copy
    ws.Cells(2, 2).PasteSpecial Paste:=xlPasteValuesAndNumberFormats
    ws.Cells(2, 2).PasteSpecial Paste:=xlPasteFormats

    'Format column A

    lMinR = v.RefersToRange.Rows(1).Row
    lMaxR = lMinR + v.RefersToRange.Rows.Count - 1
    ws.Columns("A:A").ColumnWidth = Application.Max(3.99, 1.33 * Int(Application.Log10(lMaxR)))
    For i = lMinR To lMaxR
      ws.Cells(i - lMinR + 2, 1) = i
      'Set row heights
      ws.Cells(i - lMinR + 2, 1).EntireRow.RowHeight = v.RefersToRange.Rows(i - lMinR + 1).Height
    Next i

    With ws.Range("A:A")

      'Centered
      .HorizontalAlignment = xlCenter

      'Grey interior
      .Interior.ThemeColor = xlThemeColorDark1
      .Interior.TintAndShade = -0.05

      'Light grey borders
      .Borders(xlDiagonalDown).LineStyle = xlNone
      .Borders(xlDiagonalUp).LineStyle = xlNone
      With .Borders(xlEdgeLeft)
        .LineStyle = xlContinuous: .ThemeColor = 1: .TintAndShade = -0.15: .Weight = xlThin
      End With
      With .Borders(xlEdgeTop)
        .LineStyle = xlContinuous: .ThemeColor = 1: .TintAndShade = -0.15: .Weight = xlThin
      End With
      With .Borders(xlEdgeBottom)
        .LineStyle = xlContinuous: .ThemeColor = 1: .TintAndShade = -0.15: .Weight = xlThin
      End With
      With .Borders(xlEdgeRight)
        .LineStyle = xlContinuous: .ThemeColor = 1: .TintAndShade = -0.15: .Weight = xlThin
      End With
      .Borders(xlInsideVertical).LineStyle = xlNone
      With .Borders(xlInsideHorizontal)
        .LineStyle = xlContinuous: .ThemeColor = 1: .TintAndShade = -0.15: .Weight = xlThin
      End With
    End With
```

```
    'Format row 1

    lMinC = v.RefersToRange.Columns(1).Column
    lMaxC = lMinC + v.RefersToRange.Columns.Count - 1
    For i = lMinC To lMaxC
      Debug.Assert 64 + i < Asc("Z")
      ws.Cells(1, i - lMinC + 2) = Chr(64 + i)
      'Set column widths
      ws.Cells(1, i - lMinC + 2).EntireColumn.ColumnWidth = _
        v.RefersToRange.Columns(i - lMinC + 1).Width * 0.18548
    Next i

    With ws.Range("1:1")

      'Centered
      .HorizontalAlignment = xlCenter

      'Grey interior
      .Interior.ThemeColor = xlThemeColorDark1
      .Interior.TintAndShade = -0.05

      'Light grey borders
      .Borders(xlDiagonalDown).LineStyle = xlNone
      .Borders(xlDiagonalUp).LineStyle = xlNone
      With .Borders(xlEdgeLeft)
        .LineStyle = xlContinuous: .ThemeColor = 1: .TintAndShade = -0.15: .Weight = xlThin
      End With
      With .Borders(xlEdgeTop)
        .LineStyle = xlContinuous: .ThemeColor = 1: .TintAndShade = -0.15: .Weight = xlThin
      End With
      With .Borders(xlEdgeBottom)
        .LineStyle = xlContinuous: .ThemeColor = 1: .TintAndShade = -0.15: .Weight = xlThin
      End With
      With .Borders(xlEdgeRight)
        .LineStyle = xlContinuous: .ThemeColor = 1: .TintAndShade = -0.15: .Weight = xlThin
      End With
      With .Borders(xlInsideVertical)
        .LineStyle = xlContinuous: .ThemeColor = 1: .TintAndShade = -0.15: .Weight = xlThin
      End With
      .Borders(xlInsideHorizontal).LineStyle = xlNone
    End With

    'Now export v with column letters and with row numbers, file extension: bmp, gif, png, or jpg
    Call sbExportRange2Picture(sFolder & v.Name & ".gif", _
      ws.Range(ws.Cells(1, 1), ws.Cells(lMaxR - lMinR + 2, lMaxC - lMinC + 2)))

  Case "Fml_"
    'File extension: bmp, gif, png, or jpg
    Call sbExportRange2Picture(sFolder & v.Name & ".gif", v.RefersToRange)

  End Select
Next v

ws.Delete

End Sub
```

List Named Ranges – *sbNamedRanges*

It is good programming practice to assign names to important cells or ranges. These named ranges can also contain formulas. If you want to list all used named ranges, for example:

Named Ranges		
Name	Scope	Refers to
A_global_Name	Workbook	=TODAY()
Deutsch_Bereich	Deutsch	=Deutsch!$A$1
English_Named_Range	English	=English!$A$1

sbNamedRange Program Code

```
Function sbNamedRanges(Optional bRefersToLocal As Boolean = True) As String()
'List all named ranges.
'(C) (P) by Bernd Plumhoff 01-Feb-2025 PB V3.1
Dim i           As Long
Dim s           As String
Dim wb          As Workbook

Set wb = ThisWorkbook
ReDim sR(1 To wb.Names.Count, 1 To 3) As String
For i = 1 To wb.Names.Count
  If TypeOf wb.Names(i).Parent Is Worksheet Then
    s = wb.Names(i).Parent.Name
    sR(i, 1) = Mid(wb.Names(i).Name, Len(s) + 2, 2048)
    sR(i, 2) = s
  Else
    sR(i, 1) = wb.Names(i).Name
    If bRefersToLocal Then
      sR(i, 2) = "Arbeitsmappe"
    Else
      sR(i, 2) = "Workbook"
    End If
  End If
  If bRefersToLocal Then
    sR(i, 3) = wb.Names(i).RefersToLocal
  Else
    sR(i, 3) = wb.Names(i).RefersTo
  End If
Next i
sbNamedRanges = sR
End Function
```

Show Excel Version – *ApplicationVersion*

Microsoft decided not to increase the value 16 of the function *Application.Version* since Excel 2016. This user-defined function *ApplicationVersion* is correcting for that.

ApplicationVersion Program Code

```
Function ApplicationVersion(Optional bShowBuild365 As Boolean = True) As String
'Returns MS Excel's version - with a little kludge
'(C) (P) by Bernd Plumhoff 20-Oct-2024 PB V0.61
Dim n As Integer
With Application
n = Val(.Version)
Select Case n
Case 16
  ApplicationVersion = "Excel 2016"
  On Error Resume Next 'We know what we are doing
  'Excel 365 and 2024 (LTSC) introduced ValueToText
  n = Val(.ValueToText(19))
  If n = 19 Then
    If .Build = "17932" Then
      ApplicationVersion = "Excel 2024"
    Else
      If bShowBuild365 Then
        'When all of them are 365 you might want to know the build.
        ApplicationVersion = "Excel 365 (Build " & .Build & ")"
      Else
        ApplicationVersion = "Excel 365"
      End If
    End If
  Else
    'Excel 2021 (LTSC) introduced RandArray
    n = .RandArray(1, 1, 18, 18, True)(1)
    If n = 18 Then
      ApplicationVersion = "Excel 2021"
    Else
      'Excel 2019 introduced TextJoin
      n = Val(.TextJoin(" ", True, "17"))
      If n = 17 Then ApplicationVersion = "Excel 2019"
    End If
  End If
  On Error GoTo 0
Case 15
  ApplicationVersion = "Excel 2013"
Case 14
  ApplicationVersion = "Excel 2010"
Case 12
  ApplicationVersion = "Excel 2007"
Case 11
  ApplicationVersion = "Excel 2003"
Case 10
  ApplicationVersion = "Excel 2002"
Case 9
  ApplicationVersion = "Excel 2000"
Case 8
  ApplicationVersion = "Excel 97"
Case 7
  ApplicationVersion = "Excel 7/95"
Case 5
  ApplicationVersion = "Excel 5"
Case Else
  ApplicationVersion = "[Error]"
End Select
End With
End Function
```

Number of Dimensions of an Array – *ArrayDim*

How to you determine the number of dimensions of an array?

ArrayDim Program Code

```
Function ArrayDim (v As Variant) As Long
'Returns number of dimensions of an array or 0 for
'an undimensioned array or -1 if no array at all.
'(C) (P) by Bernd Plumhoff 10-May-2010 PB V0.1
Dim i As Long
ArrayDim = -1
If Not IsArray(v) Then Exit Function
On Error Resume Next
'Err.Clear 'Not necessary
Do While IsNumeric(UBound(v, i + 1))
  If Err.Number <> 0 Then Exit Do
  i = i + 1
Loop
ArrayDim = i
End Function
```

Retrieve Cell Information – *sbGetCell*

In older versions of Excel you could retrieve interesting information about worksheet cells by using the Excel4 macro function GET.CELL. Define the name HasFormula with the value

```
=GET.CELL(48,INDIRECT("RC[-1]",FALSE))
```

for example. If you then had inserted =HasFormula next right to a cell, you would have been shown whether the cell had a formula ("True") or not ("False").

If you want to achieve similar results with VBA use the UDF sbGetCell below.

Parameters

Get.Cell Parameter	sbGetCell Parameter	Comment
1	AbsReference	Absolute reference of the upper-left cell in reference, as text in the current workspace reference style
2	RowNumber	Row number of the top cell in reference
3	ColumnNumber	Column number of the leftmost cell in reference
4	Type	Same as TYPE(reference): 1=Number, 2=Text, 4=Boolean, 16=Error, 64=Array
5	Contents	Contents of reference
6	FormulaLocal, ShowFormula	Formula in reference, as text, in either A1 or R1C1 style depending on the workspace setting
7	NumberFormat	Number format of the cell, as text (for example, "m/d/yy" or "General")
8	Horizontal alignment	Number indicating the cell's horizontal alignment: 1=General, 2=Left, 3=Center, 4=Right, 5=Fill, 6=Justify, 7=Center across cells
9	LeftBorderStyle	Number indicating the left-border style assigned to the cell: 0=None, 1=Thin, 2=Medium, 3=Dashed, 4=Dotted, 5=Thick, 6=Double, 7=Hair, 8 = MediumDashed, 9 = DashDot, 10 = MediumDashDot, 11 = DashDotDot, 12 = MediumDashDotDot, 13 = SlantDashDot

Get.Cell Parameter	sbGetCell Parameter	Comment
10	RightBorderStyle	Number indicating the right-border style assigned to the cell: 0=None, 1=Thin, 2=Medium, 3=Dashed, 4=Dotted, 5=Thick, 6=Double, 7=Hair, 8 = MediumDashed, 9 = DashDot, 10 = MediumDashDot, 11 = DashDotDot, 12 = MediumDashDotDot, 13 = SlantDashDot
11	TopBorderStyle	Number indicating the top-border style assigned to the cell: 0=None, 1=Thin, 2=Medium, 3=Dashed, 4=Dotted, 5=Thick, 6=Double, 7=Hair, 8 = MediumDashed, 9 = DashDot, 10 = MediumDashDot, 11 = DashDotDot, 12 = MediumDashDotDot, 13 = SlantDashDot
12	BottomBorderStyle	Number indicating the bottom-border style assigned to the cell: 0=None, 1=Thin, 2=Medium, 3=Dashed, 4=Dotted, 5=Thick, 6=Double, 7=Hair, 8 = MediumDashed, 9 = DashDot, 10 = MediumDashDot, 11 = DashDotDot, 12 = MediumDashDotDot, 13 = SlantDashDot
13	Pattern	Cell pattern, 0-18
14	IsLocked	
15	HiddenFormula	
16	CellWidth	Width and whether it changes with standard width, two item horizontal array
17	RowHeight	
18	FontName	
19	FontSize	
20	IsBold	
21	IsItalic	
22	IsUnderlined	
23	IsStruckThrough	
24	FontColorIndex	Font color index of first character, 1-56, 0 if automatic
25	IsOutlined	Not supported by MS Excel
26	IsShaddowed	Not supported by MS Excel
27	PageBreak	0 = no break, 1 = row, 2 = column, 3 = row and column
28	RowLevelOutline	
29	ColumnLevelOutline	
30	IsSummaryRow	
31	IsSummaryColumn	
32	WorkbookSheetName	Name of the sheet in the form "[Book1]Sheet1", or BOOK1.XLS if single sheet has identical name as workbook
33	IsWrapped	
34	LeftBorderColorIndex	Left border color index
35	RightBorderColorIndex	Right border color index
36	TopBorderColorIndex	Top border color index
37	BottomBorderColorIndex	Bottom border color index
38	ShadeForeGroundColor	
39	ShadeBackGroundColor	
40	TextStyle	
41	FormulaWOT	Formula without translation
46	HasNote	Contains text note?
47	HasSound	Not supported. Sound notes have been removed from MS Excel
48	HasFormula	If the cells contains a formula, returns TRUE; if a constant, returns FALSE
49	IsArray	If the cell is part of an array, returns TRUE; otherwise, returns FALSE
50	VerticalAlignment	1 = Top, 2 = Center, 3 = Bottom, 4 = Justified, 5 = Distributed
51	VerticalOrientation	0 = Horizontal, 1 = Vertical, 2 = Upward, 3 = Downward
52	IsStringConstant	Returns apostrophe if input cell is a string constant
53	AsText	Contents of the cell as it is currently displayed, as text, including any additional numbers or symbols resulting from the cell's formatting
54	PivotTableViewName	Returns the name of the PivotTable view containing the active cell
56	PivotTableViewFieldName	Returns the name of the field containing the active cell reference if inside a PivotTable view
57	IsSuperscript	
58	FontStyleText	Font style text of all styles in cell
59	UnderlineStyle	1 = none, 2 = single, 3 = double, 4 = single accounting, 5 = double accounting
60	IsSubscript	

Get.Cell Parameter	sbGetCell Parameter	Comment
61	PivotTableItemName	For demonstration purposes referring to cell B86
62	WorksheetName	Worksheet name like [Book1.xls]Sheet1
65	IsAddIndentAlignment	Not supported
66	WorkbookName	Book name of the workbook containing the cell in the form BOOK1.XLS
	IsHidden	

sbGetCell Program Code

```
Function sbGetCell(r As Range, s As String) As Variant
'Bernd Plumhoff V0.33 30-Oct-2022
With Application.WorksheetFunction
Application.Volatile
Select Case s
Case "AbsReference", "1"
    'Absolute style reference like $A$1
    If Application.Caller.Parent.Parent.Name = r.Worksheet.Parent.Name And _
        Application.Caller.Parent.Name = r.Worksheet.Name Then
        sbGetCell = r.Address
    Else
        If InStr(r.Worksheet.Parent.Name & r.Worksheet.Name, " ") > 0 Then
            sbGetCell = "'[" & r.Worksheet.Parent.Name & "]" & r.Worksheet.Name & "'!" & r.Address
        Else
            sbGetCell = "[" & r.Worksheet.Parent.Name & "]" & r.Worksheet.Name & "!" & r.Address
        End If
    End If
Case "RowNumber", "2"
    'Row number in the top cell reference
    sbGetCell = r.Row
Case "ColumnNumber", "3"
    'Column number of the leftmost cell in reference
    sbGetCell = r.Column
Case "Type", "4"
    'Same as TYPE(reference)
    sbGetCell = -IsEmpty(r) - .IsNumber(r) - .IsText(r) * 2 - .IsLogical(r) _
                * 4 - .IsError(r) * 16 - IsArray(r) * 64
Case "Contents", "5"
    'Contents of reference
    sbGetCell = r.Value
Case "FormulaLocal", "ShowFormula", "6"
    'Cell formula
    sbGetCell = r.FormulaLocal
Case "NumberFormat", "7"
    'Number format of cell
    sbGetCell = r.NumberFormatLocal
Case "HorizontalAlignment", "8"
    'Number indicating the cell's horizontal alignment
    Select Case r.HorizontalAlignment
    Case xlGeneral
        sbGetCell = 1
    Case xlLeft
        sbGetCell = 2
    Case xlCenter
        sbGetCell = 3
    Case xlRight
        sbGetCell = 4
    Case xlFill
        sbGetCell = 5
    Case xlJustify
        sbGetCell = 6
    Case xlCenterAcrossSelection
        sbGetCell = 7
    Case xlDistributed
        sbGetCell = 8
    Case Else
        Debug.Assert False 'Should not get here
    End Select
Case "LeftBorderStyle", "9"
    'Number indicating the left-border style assigned to the cell
    Select Case r.Borders(1).LineStyle
    Case xlLineStyleNone
        sbGetCell = 0
    Case xlHairline
        sbGetCell = IIf(r.Borders(1).Weight = xlMedium, 2, 7)
    Case xlDot
        sbGetCell = 4
    Case xlDashDotDot
        sbGetCell = IIf(r.Borders(1).Weight = xlMedium, 12, 11)
    Case xlDashDot
        sbGetCell = IIf(r.Borders(1).Weight = xlMedium, 10, 9)
    Case xlDash
```

```
        sbGetCell = IIf(r.Borders(1).Weight = xlMedium, 8, 3)
    Case xlSlantDashDot
        sbGetCell = 13
    Case xlDouble
        sbGetCell = 6
    Case Else
        sbGetCell = CVErr(xlErrValue)
    End Select
Case "RightBorderStyle", "10"
    'Number indicating the right-border style assigned to the cell
    Select Case r.Borders(2).LineStyle
    Case xlLineStyleNone
        sbGetCell = 0
    Case xlHairline
        sbGetCell = IIf(r.Borders(2).Weight = xlMedium, 2, 7)
    Case xlDot
        sbGetCell = 4
    Case xlDashDotDot
        sbGetCell = IIf(r.Borders(2).Weight = xlMedium, 12, 11)
    Case xlDashDot
        sbGetCell = IIf(r.Borders(2).Weight = xlMedium, 10, 9)
    Case xlDash
        sbGetCell = IIf(r.Borders(2).Weight = xlMedium, 8, 3)
    Case xlSlantDashDot
        sbGetCell = 13
    Case xlDouble
        sbGetCell = 6
    Case Else
        sbGetCell = CVErr(xlErrValue)
    End Select
Case "TopBorderStyle", "11"
    'Number indicating the top-border style assigned to the cell
    Select Case r.Borders(3).LineStyle
    Case xlLineStyleNone
        sbGetCell = 0
    Case xlHairline
        sbGetCell = IIf(r.Borders(3).Weight = xlMedium, 2, 7)
    Case xlDot
        sbGetCell = 4
    Case xlDashDotDot
        sbGetCell = IIf(r.Borders(3).Weight = xlMedium, 12, 11)
    Case xlDashDot
        sbGetCell = IIf(r.Borders(3).Weight = xlMedium, 10, 9)
    Case xlDash
        sbGetCell = IIf(r.Borders(3).Weight = xlMedium, 8, 3)
    Case xlSlantDashDot
        sbGetCell = 13
    Case xlDouble
        sbGetCell = 6
    Case Else
        sbGetCell = CVErr(xlErrValue)
    End Select
Case "BottomBorderStyle", "12"
    'Number indicating the bottom-border style assigned to the cell
    Select Case r.Borders(4).LineStyle
    Case xlLineStyleNone
        sbGetCell = 0
    Case xlHairline
        sbGetCell = IIf(r.Borders(4).Weight = xlMedium, 2, 7)
    Case xlDot
        sbGetCell = 4
    Case xlDashDotDot
        sbGetCell = IIf(r.Borders(4).Weight = xlMedium, 12, 11)
    Case xlDashDot
        sbGetCell = IIf(r.Borders(4).Weight = xlMedium, 10, 9)
    Case xlDash
        sbGetCell = IIf(r.Borders(4).Weight = xlMedium, 8, 3)
    Case xlSlantDashDot
        sbGetCell = 13
    Case xlDouble
        sbGetCell = 6
    Case Else
        sbGetCell = CVErr(xlErrValue)
    End Select
Case "Pattern", "13"
    'Number indicating cell pattern
    sbGetCell = r.Interior.Pattern
Case "IsLocked", "14"
    'True if cell is locked
    sbGetCell = r.Locked
Case "FormulaHidden", "HiddenFormula", "15"
    'True if cell formula is hidden
    sbGetCell = r.FormulaHidden
Case "Width", "CellWidth", "16"
    'Cell width. If array-entered into two cells of a row,
    'second value is true if width is standard
    sbGetCell = Array(r.ColumnWidth, r.UseStandardWidth) 'Not width!
Case "Height", "RowHeight", "17"
    'Cell height
    sbGetCell = r.RowHeight
```

```
Case "FontName", "18"
    'Cell font name
    sbGetCell = r.Font.Name
Case "FontSize", "19"
    'Cell font size
    sbGetCell = r.Font.Size
Case "IsBold", "20"
    'Cell is formatted bold?
    sbGetCell = r.Font.Bold
Case "IsItalic", "21"
    'Cell is formatted in Italics?
    sbGetCell = r.Font.Italic
Case "IsUnderlined", "22"
    'Cell is formatted as underlined?
    sbGetCell = (r.Font.Underline = xlUnderlineStyleSingle Or _
                 r.Font.Underline = xlUnderlineStyleSingleAccounting Or _
                 r.Font.Underline = xlUnderlineStyleDouble Or _
                 r.Font.Underline = xlUnderlineStyleDoubleAccounting)
Case "IsStruckThrough", "23"
    'Cell characters are struck through?
    sbGetCell = r.Font.Strikethrough
Case "FontColorIndex", "24"
    'Cell font color of first character, 1-56, 0 = automatic
    sbGetCell = r.Font.ColorIndex
Case "IsOutlined", "25", "IsShaddowed", "26"
    'Cell font is outlined or shaddowed? (Not supported by Excel)
    sbGetCell = False
Case "PageBreak", "27"
    '0 = no break, 1 = row, 2 = column, 3 = row and column
    sbGetCell = -(r.EntireRow.PageBreak <> xlPageBreakNone) - 2 * (r.EntireColumn.PageBreak <>
xlPageBreakNone)
Case "RowLevelOutline", "28"
    'Row level outline
    sbGetCell = r.EntireRow.OutlineLevel
Case "ColumnLevelOutline", "29"
    'Row level outline
    sbGetCell = r.EntireColumn.OutlineLevel
Case "IsSummaryRow", "30"
    'Row is a summary row?
    sbGetCell = r.EntireRow.Summary
Case "IsSummaryColumn", "31"
    'Column is a summary column?
    sbGetCell = r.EntireColumn.Summary
Case "WorkbookSheetName", "32"
    'Workbook name like [Book1.xls]Sheet1 or Book1.xls if
    'workbook and single sheet have
    'identical names
    If r.Worksheet.Parent.Name = r.Worksheet.Name & ".xls" And _
        Application.Worksheets.Count = 1 Then
        sbGetCell = r.Worksheet.Parent.Name
    Else
        sbGetCell = "[" & r.Worksheet.Parent.Name & "]" & _
        r.Worksheet.Name
    End If
Case "IsWrapped", "33"
    'Cell text is formatted as wrapped?
    sbGetCell = r.WrapText
Case "LeftBorderColorIndex", "34"
    'Left border color index
    sbGetCell = r.Borders.Item(1).ColorIndex
Case "RightBorderColorIndex", "35"
    'Right border color index
    sbGetCell = r.Borders.Item(2).ColorIndex
Case "TopBorderColorIndex", "36"
    'Top border color index
    sbGetCell = r.Borders.Item(3).ColorIndex
Case "BottomBorderColorIndex", "37"
    'Bottom border color index
    sbGetCell = r.Borders.Item(4).ColorIndex
Case "ShadeForeGroundColor", "38", "PatternBackGroundColor", "64"
    'ShadeForeGroundColor
    sbGetCell = r.Interior.PatternColorIndex
Case "ShadeBackGroundColor", "39", "PatternForeGroundColor", "63"
    'ShadeBackGroundColor
    sbGetCell = r.Interior.ColorIndex
Case "TextStyle", "40"
    'Style of the cell, as text
    sbGetCell = r.Style.NameLocal
Case "FormulaWOT", "41"
    'Returns the formula in the active cell without translating it (useful for international macro sheets)
    sbGetCell = r.Formula
'Case "HDistWinToLCell", "42"
'    'Horizontal distance, measured in points, from the left edge of the active window to the left edge of
the cell
'    sbGetCell = r. 'Does not work yet
Case "HasNote", "46"
    'True if cell contains a text note
    sbGetCell = Len(r.NoteText) > 0
Case "HasSound", "47"
    'True if cell has a sound note. Not supported.
```

```
    sbGetCell = False
Case "HasFormula", "48"
    'True if cell contains a formula
    sbGetCell = r.HasFormula
Case "IsArray", "49"
    'True if cell is part of an array formula
    sbGetCell = r.HasArray
Case "VerticalAlignment", "50"
    '1 = Top, 2 = Center, 3 = Bottom, 4 = Justified, 5 = Distributed
    sbGetCell = -(r.VerticalAlignment = xlVAlignTop) - 2 * (r.VerticalAlignment = xlVAlignCenter) _
                - 3 * (r.VerticalAlignment = xlVAlignBottom) - 4 * (r.VerticalAlignment = _
                xlVAlignJustify) - 5 * (r.VerticalAlignment = xlVAlignDistributed)
Case "VerticalOrientation", "51"
    '0 = Horizontal, 1 = Vertical, 2 = Upward, 3 = Downward
    sbGetCell = -(r.Orientation = xlVertical) - 2 * (r.Orientation = xlUpward) - _
                3 * (r.Orientation = xlDownward)
Case "IsStringConst", "IsStringConstant", "52"
    'Text alignment char "'" if cell is a string constant,
    'empty string "" if not
    sbGetCell = r.PrefixCharacter
Case "AsText", "53"
    'Cell displayed as text with numbers formatted and symbols included
    sbGetCell = r.Text
Case "PivotTableViewName", "54"
    'PivotTableViewName
    sbGetCell = r.PivotTable.Name
'Case "PivotTableViewPosition", "55"
'     'PivotTableViewPosition
'     sbGetCell = r.PivotField.Position 'Not correct yet
Case "PivotTableViewFieldName", "56"
    'PivotTableViewFieldName
    sbGetCell = r.PivotField.Name
Case "IsSuperscript", "57"
    'Cell text is formatted as superscript?
    sbGetCell = r.Font.Superscript
Case "FontStyleText", "58"
    'FontStyleText
    sbGetCell = r.Font.FontStyle
Case "UnderlineStyle", "59"
    'Underline style, 1 = none, 2 = single, 3 = double, 4 = single accounting, 5 = double accounting
    Select Case r.Font.Underline
    Case xlUnderlineStyleNone
        sbGetCell = 1
    Case xlUnderlineStyleSingle
        sbGetCell = 2
    Case xlUnderlineStyleDouble
        sbGetCell = 3
    Case xlUnderlineStyleSingleAccounting
        sbGetCell = 4
    Case xlUnderlineStyleDoubleAccounting
        sbGetCell = 5
    Case Else
        sbGetCell = CVErr(xlErrValue)
    End Select
Case "IsSubscript", "60"
    'Cell text is formatted as subscript?
    sbGetCell = r.Font.Subscript
Case "PivotTableItemName", "61"
    'PivotTableItemName
    sbGetCell = r.PivotItem.Name
Case "WorksheetName", "62"
    'Worksheet name like [Book1.xls]Sheet1
        sbGetCell = "[" & r.Worksheet.Parent.Name & "]" & _
                    r.Worksheet.Name
Case "IsAddIndentAlignment", "65"
    'Only Far East Excel Versions
    sbGetCell = False 'Not supported here
Case "WorkbookName", "66"
    'Workbook name like Book1.xls
    sbGetCell = r.Worksheet.Parent.Name
Case "IsHidden"
    'Cell hidden?
    sbGetCell = r.EntireRow.Hidden Or r.EntireColumn.Hidden
Case Else
    sbGetCell = CVErr(xlErrValue)
End Select
End With
End Function
```

Next Floating Point Number – *sbNextFloat*

If you need to identify the next closest floating point number:

	A	B
1	**Input**	**Output**
2	0,9999999999999990	1,0000000000000000
3	1,0000000000000000	1,0000000000000100
4	10,0000000000000000	10,0000000000001000
5	100.000.000.000.000,0000000000000000	100.000.000.000.001,0000000000000000
6	3,1415926535897900	3,1415926535898000
7	-1,0000000000000000	-0,9999999999999900
8	-1,0000000000000100	-1,0000000000000000
9	-10,0000000000001000	-10,0000000000000000
10	-100.000.000.000.001,0000000000000000	-100.000.000.000.000,0000000000000000
11	-3,1415926535898000	-3,1415926535897900

Worksheet Formulas		
Range	Formula	
B2:B11	**B2**	=sbNextFloat(A2)

See also Increment Lowest Significant Digit on page 92.

sbNextFloat Program Code

```
Function sbNextFloat(d As Double, Optional bUp As Boolean = True) As Double
'Returns the smallest double which is greater than the input (bUp = True)
'or the greatest double which is smaller than the input value (bUp = False).
'(C) (P) by Bernd Plumhoff 03-Oct-2010 PB V0.11
sbNextFloat = d - (2# * bUp + 1#) * CDbl("1e" & Right(Format(d, "." & _
              String(15, "0") & "E+000"), 4) - 15)
End Function
```

Calling Other Windows Programs Using the Example *sbZip*

With VBA, you can call other Windows programs. Example: If you want to compress a file or a directory (including all subdirectories), you can either do this directly with VBA or use the freely available program 7zip.

The VBA program *sbZip* uses the logger presented here (see Program Flow Documentation – Logging Class) to log the standard output and standard error output of 7zip and LibFileTools.

sbZip – Program Code

```
Public Const AppVersion As String = "sbZip_Version_13"

Sub sbZip(ByVal vSourceFullPathName As Variant, _
          ByVal vDestinationZipFullPathName As Variant, _
          Optional bCreate As Boolean = True, _
          Optional bUse7zip As Boolean = False)
'Create zip file vDestinationZipFullPathName and insert zipped file or folder vSourceFullPathName.
'This sub is using LibFileTools, https://github.com/cristianbuse/VBA-FileTools.
'If bUse7zip:=True then 7zip needs to be installed at C:\Program Files\7-Zip\7z.exe.
'Version When        Who   What
'     13 03-Mar-2025 Bernd Execution order of Set oShell and oShell.NameSpace corrected
```

```
Dim iFile     As Integer
Dim lItems    As Long
Dim lRepeat   As Long
Dim sLine     As String
Dim sShellCmd As String
Dim sPath     As String
Dim oExec     As Object
Dim oOutput   As Object
Dim oShell    As Object
If GLogger Is Nothing Then Call Start_Log
If bCreate Then
  If IsFile(CStr(vDestinationZipFullPathName)) Then
    If Not DeleteFile(CStr(vDestinationZipFullPathName)) Then
      GLogger.warn "Could not delete file '" & vDestinationZipFullPathName & "'"
    End If
  End If
End If
If bUse7zip Then
  If IsFile("C:\Program Files\7-Zip\7z.exe") Then
    Set oShell = CreateObject("WScript.Shell")
    sShellCmd = "C:\Program Files\7-Zip\7z.exe a """ & vDestinationZipFullPathName & _
      """ """ & vSourceFullPathName & """"
    Set oExec = oShell.exec(sShellCmd)
    Set oOutput = oExec.StdOut
    Do While Not oOutput.AtEndOfStream
      sLine = oOutput.ReadLine
      If sLine <> "" Then GLogger.info "STDOUT " & sLine
    Loop
    Set oOutput = oExec.StdErr
    Do While Not oOutput.AtEndOfStream
      sLine = oOutput.ReadLine
      If sLine <> "" Then GLogger.warn "STDERR " & sLine
    Loop
    Do While oExec.Status = 0
      Application.Wait (Now + TimeValue("0:00:01"))
    Loop
    GLogger.info vSourceFullPathName & "' zipped into '" & vDestinationZipFullPathName & "'"
  Else
    GLogger.fatal "C:\Program Files\7-Zip\7z.exe doesn't exist. Cannot zip '" & _
    vSourceFullPathName & "'"
  End If
Else
  If bCreate Then
    sPath = GetLocalPath(ThisWorkbook.Path)
    If Right(sPath, 1) <> "\" Then sPath = sPath & "\"
    If IsFile(sPath & "Zip_Template.zip") Then
      'Workaround in case print sequence in Else clause does not work
      CopyFile sPath & "Zip_Template.zip", CStr(vDestinationZipFullPathName)
      If Not IsFile(CStr(vDestinationZipFullPathName)) Then
        GLogger.warn "Could not copy template file '" & vDestinationZipFullPathName & "'"
      End If
    Else
      iFile = FreeFile
      Open vDestinationZipFullPathName For Output As #iFile
      Print #iFile, Chr$(80) & Chr$(75) & Chr$(5) & Chr$(6) & String(18, 0)
      Close #iFile
    End If
  End If
  Set oShell = CreateObject("Shell.Application")
  On Error Resume Next
  lItems = oShell.Namespace(vDestinationZipFullPathName).Items.Count
  On Error GoTo 0
  If GetAttr(vSourceFullPathName) = vbDirectory Then
    oShell.Namespace(vDestinationZipFullPathName).CopyHere _
    oShell.Namespace(vSourceFullPathName).Items, 16
    lRepeat = 0
    On Error Resume Next
    Do Until oShell.Namespace(vDestinationZipFullPathName).Items.Count = _
      lItems + oShell.Namespace(vSourceFullPathName).Items.Count Or lRepeat > 5
      Application.Wait (Now + TimeValue("0:00:01"))
      lRepeat = lRepeat + 1
    Loop
    On Error GoTo 0
  Else
    oShell.Namespace(vDestinationZipFullPathName).CopyHere vSourceFullPathName, 16
    lRepeat = 0
    On Error Resume Next
    Do Until oShell.Namespace(vDestinationZipFullPathName).Items.Count = _
      lItems + 1 Or lRepeat > 3
      Application.Wait (Now + TimeValue("0:00:01"))
      lRepeat = lRepeat + 1
    Loop
    On Error GoTo 0
  End If
End If
End Sub
```

Excel Don't's – What You Should not be Doing with Excel

"Always listen to experts. They'll tell you what can't be done and why. Then do it." [Robert Heinlein]

Criticism should be constructive - in general I try to show how things could or should positively be done. But during more than 30 years of Excel practice I encountered some examples which could and maybe should be avoided.

A table of what you should avoid

Please look at this list as an intellectual challenge and take it with a grain of salt.

#	Do not use this	Why?
1	Excel Add-in freeware like Morefunc	You should have a license or at least the free source code for add-ins so that you can help yourself in case the provider vanishes. If not, it is a security threat, too. For Morefunc the provider has obviously disappeared, no source code seems to be available, some of its functions no longer work with Excel 2007 or later. I think it's irresponsible to suggest to use it anymore.
2	Worksheet formulas you do not understand	How would you know that it is correct? After row or column insertions/deletions? How would you adapt it to future needs or even to simple sheet rearrangements? With a UDF you only need to understand the functionality and the interface. This is called encapsulation.
3	Enable option "Precision as Displayed" from Calculate tab on the Options dialog (Tools menu, up to Excel 2003) or from Excel Options (Advanced menu, Excel 2007 onwards)	A display format change in any input cell or intermediary cell can ruin the results. Permanently.
4	Enable option "Iteration" from Calculate tab on the Options dialog (Tools menu, up to Excel 2003) or from Excel Options (Formulas menu, Excel 2007 onwards)	You will not see accidental circular references.
5	1904 date system (to be able to present negative time = hours/minutes, for example)	Almost everybody is using the 1900 date system. If you switch between these systems you will need to remember the number 1462 by heart (to add or subtract it after CTRL + c with ALT + e, s, v and then d [add] or s [subtract]).
6	Do not test double precision numbers with = for equality. Use a test like ABS(a - b) < 1E-13 instead	=(43.1-43.2)+1=0.9 does not result in True as you might think, for example. See Microsoft's article on this topic at https://learn.microsoft.com/en-us/office/troubleshoot/excel/floating-point-arithmetic-inaccurate-result. Best read on this topic is Goldberg's article https://www.itu.dk/~sestoft/bachelor/IEEE754_article.pdf.
7	Count or sum specified cells conditionally for all different elements of a list with SUMPRODUCT	SUMPRODUCT is powerful but for a statistic like sum or count on all elements of a list there are better ways like a pivot table or PowerQuery.
8	Worksheet function INDIRECT	This function is volatile which means it gets recalculated each time you press F9. If you use INDIRECT something is terribly wrong with your worksheet design.
9	Worksheet function AGGREGATE	If you exclude hidden rows or exclude error values, no auditor will accept this (I bet).

Number Systems, Formats, and Transformations

Abstract

As a programmer, one often deals with number systems and their representation or conversion. Here, I present some programs that I have come to know and use over time.

Transformations and Calculations of Numbers

Spell Numbers in English Words – *sbSpellNumber*

Sometimes you need to spell numbers in English words with Dollars/Cents or British Pound Sterling/Pence or European Euro/Cent. 12.31 would result in Twelve Dollars and Thirtyone Cents, for example.

Note: There are many faulty spellnumber versions circulating in the web. I suggest to test your preferred version with the inputs listed below:

	A	B	C
1	**Spell numbers:**		
2			
3	**Number**	**Spell Number**	**In Worten**
4	1.000.000.000.000.000,00	>>>>> Error (Absolute amount > 999999999999999)! <<<<<	>>>>> Fehler (Absolutbetrag > 999999999999999)! <<<<<
5	0,123	Zero Dollars and Twelve Cents (rounded)	Null Euro und Zwölf Cent (gerundet)
6	-1,00	Minus One Dollar and Zero Cents	Minus Ein Euro und Null Cent
7	20,123	Twenty Dollars and Twelve Cents (rounded)	Zwanzig Euro und Zwölf Cent (gerundet)
8	-20,123	Minus Twenty Dollars and Twelve Cents (rounded)	Minus Zwanzig Euro und Zwölf Cent (gerundet)
9	1,01	One Dollar and One Cent	Ein Euro und Ein Cent
10	1.000.001,01	One Million One Dollars and One Cent	Eine Million und Ein Euro und Ein Cent

sbInWorten / sbSpellNumber Code

```
Private sNWord(0 To 28) As String
Private sHWord(1 To 4) As String

Function sbInWorten(ByVal sNumber As String) As String
   sbInWorten = sbSpellNumber(sNumber, "German", "EUR")
End Function

Function sbSpellNumber(ByVal sNumber As String, _
           Optional sLang As String = "English", _
           Optional sCcy As String = "USD") As String
'Template was Microsoft's limited version:
'https://support.microsoft.com/de-de/help/213360/
'how-to-convert-a-numeric-value-into-english-words-in-excel
'This version informs the user about its limits.
'(C) (P) by Bernd Plumhoff  02-Mar-2018 PB V1.0

Dim Euros As String, cents As String
Dim Result As String, Temp As String
Dim DecimalPlace As Integer, Count As Integer
Dim Place(1 To 6) As String
Dim dNumber As Double
Dim prefix As String, suffix As String

Select Case sLang
Case "English"
   Place(1) = ""
   Place(2) = " Thousand "
   Place(3) = " Million "
   Place(4) = " Billion "
   Place(5) = " Trillion "
```

```
   Place(6) = " Mantissa not wide enough for this number "
   sHWord(1) = ">>>>> Error (Absolute amount > 999999999999999)! <<<<<"
   sHWord(2) = " (rounded)"
   sHWord(3) = "Minus "
   sHWord(4) = "and"
   sNWord(0) = "zero"
   sNWord(1) = "one"
   sNWord(2) = "two"
   sNWord(3) = "three"
   sNWord(4) = "four"
   sNWord(5) = "five"
   sNWord(6) = "six"
   sNWord(7) = "seven"
   sNWord(8) = "eight"
   sNWord(9) = "nine"
   sNWord(10) = "ten"
   sNWord(11) = "eleven"
   sNWord(12) = "twelve"
   sNWord(13) = "thirteen"
   sNWord(14) = "fourteen"
   sNWord(15) = "fifteen"
   sNWord(16) = "sixteen"
   sNWord(17) = "seventeen"
   sNWord(18) = "eighteen"
   sNWord(19) = "nineteen"
   sNWord(20) = "twenty"
   sNWord(21) = "thirty"
   sNWord(22) = "fourty"
   sNWord(23) = "fifty"
   sNWord(24) = "sixty"
   sNWord(25) = "seventy"
   sNWord(26) = "eighty"
   sNWord(27) = "ninety"
   sNWord(28) = "hundred"
Case "German"
   Place(1) = ""
   Place(2) = " Tausend "
   Place(3) = " Millionen "
   Place(4) = " Milliarden "
   Place(5) = " Billionen "
   Place(6) = " Die Mantisse ist nicht groß genug für diese Zahl "
   sHWord(1) = ">>>>> Fehler (Absolutbetrag > 999999999999999)! <<<<<"
   sHWord(2) = " (gerundet)"
   sHWord(3) = "Minus "
   sHWord(4) = "und"
   sNWord(0) = "null"
   sNWord(1) = "ein"
   sNWord(2) = "zwei"
   sNWord(3) = "drei"
   sNWord(4) = "vier"
   sNWord(5) = "fünf"
   sNWord(6) = "sechs"
   sNWord(7) = "sieben"
   sNWord(8) = "acht"
   sNWord(9) = "neun"
   sNWord(10) = "zehn"
   sNWord(11) = "elf"
   sNWord(12) = "zwölf"
   sNWord(13) = "dreizehn"
   sNWord(14) = "vierzehn"
   sNWord(15) = "fünfzehn"
   sNWord(16) = "sechzehn"
   sNWord(17) = "siebzehn"
   sNWord(18) = "achtzehn"
   sNWord(19) = "neunzehn"
   sNWord(20) = "zwanzig"
   sNWord(21) = "dreißig"
   sNWord(22) = "vierzig"
   sNWord(23) = "fünfzig"
   sNWord(24) = "sechzig"
   sNWord(25) = "siebzig"
   sNWord(26) = "achtzig"
   sNWord(27) = "neunzig"
   sNWord(28) = "hundert"
End Select

'Empty string = 0
If "" = sNumber Then
   sNumber = "0"
End If
dNumber = sNumber + 0#
'If we cannot cope with it, tell the user!
If Abs(dNumber) > 999999999999999# Then
   sbSpellNumber = sHWord(1)
   Exit Function
End If

'If we have to round we present a suffix "(rounded)"
If Abs(dNumber - Round(dNumber, 2)) > 1E-16 Then
   dNumber = Round(dNumber, 2)
```

```
   suffix = sHWord(2)
End If

'Negative numbers get a prefix "Minus"
If dNumber < 0# Then
   prefix = sHWord(3)
   dNumber = -dNumber
   sNumber = Right(sNumber, Len(sNumber) - 1)
End If

sNumber = Trim(Str(sNumber))
If Left(sNumber, 1) = "." Then
   sNumber = "0" & sNumber
End If
DecimalPlace = InStr(sNumber, ".")
If DecimalPlace > 0 Then
   cents = GetTens(Left(Mid(sNumber, DecimalPlace + 1) & "00", 2), _
                 sLang, sCcy)
   sNumber = Trim(Left(sNumber, DecimalPlace - 1))
End If

Count = 1
Do While sNumber <> ""
   Temp = GetHundreds(Right(sNumber, 3), sLang, sCcy)
   If Temp <> "" Then
       If Euros <> "" And sLang = "German" Then
           Euros = Temp & Place(Count) & " " & _
                   sHWord(4) & " " & Euros
       Else
           Euros = Temp & Place(Count) & Euros
       End If
   End If
   If Len(sNumber) > 3 Then
       sNumber = Left(sNumber, Len(sNumber) - 3)
   Else
       sNumber = ""
   End If
   Count = Count + 1
Loop

Select Case sCcy
Case "EUR"
   Select Case Euros
       Case ""
           Euros = sNWord(0) & " Euros"
       Case sNWord(1)
           Euros = sNWord(1) & " Euro"
       Case Else
           Euros = Euros & " Euros"
   End Select

   Select Case cents
       Case ""
           cents = " " & sHWord(4) & " " & sNWord(0) & " Cents"
       Case sNWord(1)
           cents = " " & sHWord(4) & " " & sNWord(1) & " Cent"
       Case Else
           cents = " " & sHWord(4) & " " & cents & " Cents"
   End Select
Case "GBP"
   Select Case Euros
       Case ""
           Euros = sNWord(0) & " Pounds"
       Case sNWord(1)
           Euros = sNWord(1) & " Pound"
       Case Else
           Euros = Euros & " Pounds"
   End Select
   Select Case cents
       Case ""
           cents = " " & sHWord(4) & " " & sNWord(0) & " Pence"
       Case sNWord(1)
           cents = " " & sHWord(4) & " " & sNWord(1) & " Penny"
       Case Else
           cents = " " & sHWord(4) & " " & cents & " Pence"
   End Select
Case "USD"
   Select Case Euros
       Case ""
           Euros = sNWord(0) & " Dollars"
       Case sNWord(1)
           Euros = sNWord(1) & " Dollar"
       Case Else
           Euros = Euros & " Dollars"
   End Select

   Select Case cents
       Case ""
           cents = " " & sHWord(4) & " " & sNWord(0) & " Cents"
       Case sNWord(1)
```

```
            cents = " " & sHWord(4) & " " & sNWord(1) & " Cent"
        Case Else
            cents = " " & sHWord(4) & " " & cents & " Cents"
    End Select
End Select

Temp = UCase(Replace(Euros & cents, "  ", " "))
Select Case sLang
Case "English"
    Temp = Application.WorksheetFunction.Proper(Temp)
    Temp = Replace(Temp, " And ", " and ")
Case "German"
    Temp = Application.WorksheetFunction.Proper(Temp)
    Temp = Replace(Temp, "Ein Millionen", "Eine Million")
    Temp = Replace(Temp, "Ein Milliarden", "Eine Milliarde")
    Temp = Replace(Temp, "Ein Billionen", "Eine Billion")
    Temp = Replace(Temp, "Dollars", "Dollar")
    Temp = Replace(Temp, "Cents", "Cent")
    Temp = Replace(Temp, "Pounds", "Pfund")
    Temp = Replace(Temp, "Pound", "Pfund")
    Temp = Replace(Temp, "Euros", "Euro")
    Temp = Replace(Temp, "Pence", "Pennies")
    Temp = Replace(Temp, " Und ", " und ")
End Select
sbSpellNumber = prefix & Temp & suffix
End Function

Private Function GetHundreds(ByVal sNumber, _
            Optional sLang As String = "English", _
            Optional sCcy As String = "USD") As String
Dim Result As String

If Val(sNumber) = 0 Then Exit Function
    sNumber = Right("000" & sNumber, 3)
    If Mid(sNumber, 1, 1) <> "0" Then
        Result = GetDigit(Mid(sNumber, 1, 1)) _
                & sNWord(28)
        If Mid(sNumber, 2, 2) <> "00" Then
            Result = Result & sHWord(4)
        End If
    End If
    If Mid(sNumber, 2, 1) <> "0" Then
        Result = Result & GetTens(Mid(sNumber, 2), sLang, sCcy)
    ElseIf Mid(sNumber, 3, 1) <> "0" Then
        Result = Result & GetDigit(Mid(sNumber, 3))
    End If
    GetHundreds = Result
End Function

Private Function GetTens(TensText As String, _
            Optional sLang As String = "English", _
            Optional sCcy As String = "USD")
Dim Result As String

Result = ""
If Val(Left(TensText, 1)) = 1 Then   '10-19...
    If Val(TensText) > 9 And Val(TensText) < 20 Then
        GetTens = sNWord(Val(TensText))
    End If
    Exit Function
Else                                '20-99...
    If Val(Left(TensText, 1)) > 1 And _
        Val(Left(TensText, 1)) < 10 Then
        Result = sNWord(18 + Val(Left(TensText, 1)))
    Else
        Result = GetDigit(Right(TensText, 1))
    End If
    If Right(TensText, 1) <> "0" And Left(TensText, 1) <> "0" Then
        Select Case sLang
        Case "German"
            Result = GetDigit(Right(TensText, 1)) & _
                        sHWord(4) & Result
        Case "English"
            Result = Result & GetDigit(Right(TensText, 1))
        End Select
    End If
End If
GetTens = Result
End Function

Private Function GetDigit(Digit As String) As String
If Val(Digit) < 10 Then
    GetDigit = sNWord(Val(Digit))
Else
    GetDigit = ""
End If
End Function
```

Convert Decimal into Binary Numbers – sbDec2Bin / sbBin2Dec

What is the binary representation (bitlength = 256) of the decimal number -87236234623462783462873462783462783462 8? Don't ask Excel's built-in function DEC2BIN. It can only deal with numbers between -512 and 511. If you want to get the correct answer
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111010110111110110101000111111001110111100101111001000010000111010110010010100110011010000100
11001111010100001010101001011110011111100
then have a look at the function *sbDec2Bin* listed below.

Please note that fractional parts are supported for positive decimals only. The decimal 0.5 is in binary format equal to 0.1, for example.

Funktion	Parameter 1	Parameter 2	Result	Comment
sbDec2Bin	2005		1111101010	
sbDec2Bin	2005	11	#Value!	2005 cannot be represented with 11 bits
sbDec2Bin	11.5		1011.1	
sbDec2Bin	2.25		10.01	
sbBin2Dec	1111		15	
sbBin2Dec	1111	4	-1	4 bits, the first being the sign
sbBin2Dec	11.11		3.75	

sbDec2Bin, sbBin2Dec, sbDivBy2, sbBinNeg, and *sbDecAdd* Code

```
Function sbDec2Bin(ByVal sDecimal As String, _
               Optional lBits As Long = 32, _
               Optional blZeroize As Boolean = False) As String
'Convert a decimal number into its binary equivalent.
'(C) (P) by Bernd Plumhoff 18-Dec-2021 PB V0.4
Dim sDec As String, sFrac As String
Dim sD As String, sB As String
Dim blNeg As Boolean
Dim i As Long, lPosDec As Long, lLenBinInt As Long
lPosDec = InStr(sDecimal, Application.DecimalSeparator)
If lPosDec > 0 Then
   If Left(sDecimal, 1) = "-" Then 'So far we cannot handle
                           'negative fractions, will come later
       sbDec2Bin = CVErr(xlErrValue)
       Exit Function
   End If
   sDec = Left(sDecimal, lPosDec - 1)
   sFrac = Right(sDecimal, Len(sDecimal) - lPosDec)
   lPosDec = Len(sFrac)
Else
   sDec = sDecimal
   sFrac = ""
End If
sB = ""
If Left(sDec, 1) = "-" Then
   blNeg = True
   sD = Right(sDec, Len(sDec) - 1)
Else
   blNeg = False
   sD = sDec
End If
Do While Len(sD) > 0
   Select Case Right(sD, 1)
       Case "0", "2", "4", "6", "8"
           sB = "0" & sB
       Case "1", "3", "5", "7", "9"
           sB = "1" & sB
       Case Else
           sbDec2Bin = CVErr(xlErrValue)
           Exit Function
   End Select
   sD = sbDivBy2(sD, True)
   If sD = "0" Then
       Exit Do
   End If
Loop
If blNeg And sB <> "1" & String(lBits - 1, "0") Then
   sB = sbBinNeg(sB, lBits)
End If
'Test whether string representation is in range and correct
'If not, the user has to increase lbits
lLenBinInt = Len(sB)
If lLenBinInt > lBits Then
   sbDec2Bin = CVErr(xlErrNum)
   Exit Function
Else
   If (Len(sB) = lBits) And (Left(sB, 1) <> -blNeg & "") Then
       sbDec2Bin = CVErr(xlErrNum)
       Exit Function
   End If
End If

If blZeroize Then sB = Right(String(lBits, "0") & sB, lBits)

If lPosDec > 0 And lLenBinInt + 1 < lBits Then
   sB = sB & Application.DecimalSeparator
   i = 1
   Do While i + lLenBinInt < lBits
       sFrac = sbDecAdd(sFrac, sFrac) 'Double fractional part
       If Len(sFrac) > lPosDec Then
           sB = sB & "1"
           sFrac = Right(sFrac, lPosDec)
           If sFrac = String(lPosDec, "0") Then
               Exit Do
           End If
       Else
           sB = sB & "0"
       End If
       i = i + 1
   Loop
   sbDec2Bin = sB
Else
   sbDec2Bin = sB
End If
End Function
```

```
Function sbBin2Dec(sBinary As String, _
    Optional lBits As Long = 32) As String
'Converts a binary number into its decimal equivalent.
'(C) (P) by Bernd Plumhoff 18-Dec-2021 PB V0.4
Dim sBin As String
Dim sB As String
Dim sFrac As String
Dim sD As String
Dim sR As String
Dim blNeg As Boolean
Dim i As Long
Dim lPosDec As Long

lPosDec = InStr(sBinary, Application.DecimalSeparator)
If lPosDec > 0 Then
   If (Left(Right(String(lBits, "0") & sBinary, lBits), 1) = "1") And _
       Len(sBin) >= lBits Then 'So far we cannot handle Right(String(lBits, "0") & sB, lBits)
                    'negative fractions, will come later
       sbBin2Dec = CVErr(xlErrValue)
       Exit Function
   End If
   sBin = Left(sBinary, lPosDec - 1)
   sFrac = Right(sBinary, Len(sBinary) - lPosDec)
   lPosDec = Len(sFrac)
Else
   sBin = sBinary
   sFrac = ""
End If

Select Case Sgn(Len(sBin) - lBits)
   Case 1
       sbBin2Dec = CVErr(xlErrNum)
       Exit Function
   Case 0
       If Left(sBin, 1) = "1" Then
           sB = sbBinNeg(sBin, lBits)
           blNeg = True
       Else
           sB = sBin
           blNeg = False
       End If
   Case -1
       sB = sBin
       blNeg = False
End Select
sD = "1"
sR = "0"
For i = Len(sB) To 1 Step -1
   Select Case Mid(sB, i, 1)
       Case "1"
           sR = sbDecAdd(sR, sD)
       Case "0"
           'Do nothing
       Case Else
           sbBin2Dec = CVErr(xlErrNum)
           Exit Function
   End Select
   sD = sbDecAdd(sD, sD) 'Double sd
Next i

If lPosDec > 0 Then 'now the fraction
   sD = "0" & Application.DecimalSeparator & "5"
   For i = 1 To lPosDec
       If Mid(sFrac, i, 1) = "1" Then
           sR = sbDecAdd(sR, sD)
       End If
       sD = sbDivBy2(sD, False)
   Next i
End If

If blNeg Then
   sbBin2Dec = "-" & sR
Else
   sbBin2Dec = sR
End If
End Function

Function sbDivBy2(sDecimal As String, blInt As Boolean) As String
'Divide positive sDecimal by two, blInt = TRUE returns integer only
'(C) (P) by Bernd Plumhoff 18-Dec-2021 PB V0.4
Dim i As Long, lPosDec As Long
Dim sDec As String, sD As String
Dim lCarry As Long

If Not blInt Then
   lPosDec = InStr(sDecimal, Application.DecimalSeparator)
   If lPosDec > 0 Then
       sDec = Left(sDecimal, lPosDec - 1) & _
              Right(sDecimal, Len(sDecimal) - lPosDec) 'Without decimal point
       'lposdec already defines location of decimal point
```

```
   Else
       sDec = sDecimal
       lPosDec = Len(sDec) + 1 'Location of decimal point
   End If
   If ((1 * Right(sDec, 1)) Mod 2) = 1 Then
       sDec = sDec & "0"  'Append zero so that integer algorithm
                          'below calculates division exactly
   End If
Else
   sDec = sDecimal
End If

lCarry = 0
For i = 1 To Len(sDec)
   sD = sD & Int((lCarry * 10 + Mid(sDec, i, 1)) / 2)
   lCarry = (lCarry * 10 + Mid(sDec, i, 1)) Mod 2
Next i

If Not blInt Then
   If Right(sD, Len(sD) - lPosDec + 1) <> _
       String(Len(sD) - lPosDec + 1, "0") Then   'frac part is non-zero
       i = Len(sD)
       Do While Mid(sD, i, 1) = "0"
           i = i - 1  'Skip trailing zeros
       Loop
       sD = Left(sD, lPosDec - 1) & Application.DecimalSeparator & _
            Mid(sD, lPosDec, i - lPosDec + 1) 'Insert decimal point again
   End If
End If

i = 1
Do While i < Len(sD)
   If Mid(sD, i, 1) = "0" Then
       i = i + 1
   Else
       Exit Do
   End If
Loop
If Mid(sD, i, 1) = Application.DecimalSeparator Then
   i = i - 1
End If
sbDivBy2 = Right(sD, Len(sD) - i + 1)
End Function

Function sbBinNeg(sBin As String, _
               Optional lBits As Long = 32) As String
'Negate sBin: take the 2's-complement, then add one
'(C) (P) by Bernd Plumhoff 18-Dec-2021 PB V0.4
Dim i As Long, sB As String

If Len(sBin) > lBits Or sBin = "1" & String(lBits - 1, "0") Then
   sbBinNeg = CVErr(xlErrValue)
   Exit Function
End If

'Calculate 2's-complement
For i = Len(sBin) To 1 Step -1
   Select Case Mid(sBin, i, 1)
       Case "1"
           sB = "0" & sB
       Case "0"
           sB = "1" & sB
       Case Else
           sbBinNeg = CVErr(xlErrValue)
           Exit Function
   End Select
Next i

sB = String(lBits - Len(sBin), "1") & sB

'Now add 1
i = lBits
Do While i > 0
   If Mid(sB, i, 1) = "1" Then
       Mid(sB, i, 1) = "0"
       i = i - 1
   Else
       Mid(sB, i, 1) = "1"
       i = 0
   End If
Loop

'Finally strip leading zeros
i = InStr(sB, "1")
If i = 0 Then
   sbBinNeg = "0"
Else
   sbBinNeg = Right(sB, Len(sB) - i + 1)
End If
End Function
```

```
Function sbDecAdd(sOne As String, sTwo As String) As String
'Sum up two positive string decimals.
'(C) (P) by Bernd Plumhoff 18-Dec-2021 PB V0.4
Dim lStrLen As Long
Dim s1 As String, s2 As String
Dim sA As String, sB As String, sR As String
Dim d As Long, lCarry As Long, lPosDec1 As Long, lPosDec2 As Long
Dim sF1 As String, sF2 As String

lPosDec1 = InStr(sOne, Application.DecimalSeparator)
If lPosDec1 > 0 Then
   s1 = Left(sOne, lPosDec1 - 1)
   sF1 = Right(sOne, Len(sOne) - lPosDec1)
   lPosDec1 = Len(sF1)
Else
   s1 = sOne
   sF1 = ""
End If
lPosDec2 = InStr(sTwo, Application.DecimalSeparator)
If lPosDec2 > 0 Then
   s2 = Left(sTwo, lPosDec2 - 1)
   sF2 = Right(sTwo, Len(sTwo) - lPosDec2)
   lPosDec2 = Len(sF2)
Else
   s2 = sTwo
   sF2 = ""
End If

If lPosDec1 + lPosDec2 > 0 Then
   If lPosDec1 > lPosDec2 Then
       sF2 = sF2 & String(lPosDec1 - lPosDec2, "0")
   Else
       sF1 = sF1 & String(lPosDec2 - lPosDec1, "0")
       lPosDec1 = lPosDec2
   End If
   sF1 = sbDecAdd(sF1, sF2) 'Add fractions as integer numbers
   If Len(sF1) > lPosDec1 Then
       lCarry = 1
       sF1 = Right(sF1, lPosDec1)
   Else
       lCarry = 0
   End If
   Do While lPosDec1 > 0
       If Mid(sF1, lPosDec1, 1) <> "0" Then
           Exit Do
       End If
       lPosDec1 = lPosDec1 - 1
   Loop
   sF1 = Left(sF1, lPosDec1)
Else
   lCarry = 0
End If

lStrLen = Len(s1)
If lStrLen < Len(s2) Then
   lStrLen = Len(s2)
   sA = String(lStrLen - Len(s1), "0") & s1
   sB = s2
Else
   sA = s1
   sB = String(lStrLen - Len(s2), "0") & s2
End If

Do While lStrLen > 0
   d = 0 + Mid(sA, lStrLen, 1) + Mid(sB, lStrLen, 1) + lCarry
   If d > 9 Then
       sR = (d - 10) & sR
       lCarry = 1
   Else
       sR = d & sR
       lCarry = 0
   End If
   lStrLen = lStrLen - 1
Loop
If lCarry > 0 Then
   sR = lCarry & sR
End If

If lPosDec1 > 0 Then
   sbDecAdd = sR & Application.DecimalSeparator & sF1
Else
   sbDecAdd = sR
End If

End Function
```

Identify German Bank Holidays – *IstFeiertag*

Do you want to find out if a date is a German public holiday? Or if it is a holiday in a specific federal state?

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	Deutschland	Baden-Württemberg	Bayern	Bayern (überw. kath.)	Berlin	Brandenburg	Bremen	Hamburg	Hessen	Mecklenburg-Vorpommern	Niedersachsen	Nordrhein-Westfalen	Rheinland-Pfalz	Saarland	Sachsen	Sachsen (einig. kath.)	Sachsen-Anhalt	Schleswig-Holstein	Thüringen	Thüringen (einig. kath.)	Mein Arbeitsort					
2	DE	BW	BY	BYK	BE	BB	HB	HH	HE	MV	NI	NW	RP	SL	SN	SNK	ST	SH	TH	THK	AO			Tag	Land	Feiertag?
3	01.01.2023	06.01.2023	06.01.2023	06.01.2023	08.03.2023	31.10.2023	31.10.2023	31.10.2023	08.06.2023	08.03.2023	31.10.2023	08.06.2023	08.06.2023	08.06.2023	31.10.2023	08.06.2023	06.01.2023	31.10.2023	20.09.2023	08.06.2023	06.01.2023		Positivtest	21.04.2025	DE	WAHR
4	07.04.2023	08.06.2023	08.06.2023	08.06.2023	08.03.2024	31.10.2024	31.10.2024	31.10.2024	30.05.2024	31.10.2023	31.10.2024	01.11.2023	01.11.2023	15.08.2023	22.11.2023	31.10.2023	06.01.2024	31.10.2024	31.10.2023	20.09.2023	08.06.2023		Positivtest	08.08.2088	AO	WAHR
5	10.04.2025	01.11.2023	01.11.2023	15.08.2023	08.03.2025	31.10.2025	31.10.2025	31.10.2025	19.06.2025	08.03.2024	31.10.2025	30.05.2024	30.05.2024	01.11.2023	31.10.2024	22.11.2023	06.01.2025	31.10.2025	20.09.2024	31.10.2023	08.08.2023		Negativtest	27.12.2025	DE	FALSCH
6	01.05.2023	06.01.2024	06.01.2024	01.11.2023	08.03.2026	31.10.2026	31.10.2026	31.10.2026	04.06.2026	31.10.2024	31.10.2026	01.11.2024	01.11.2024	30.05.2024	20.11.2024	30.05.2024	06.01.2026	31.10.2026	31.10.2024	30.05.2024	15.08.2023		Negativtest	07.01.2029	BYK	FALSCH
7	18.05.2023	30.05.2024	30.05.2024	06.01.2024	08.03.2027	31.10.2027	31.10.2027	31.10.2027	27.05.2027	08.03.2025	31.10.2027	19.06.2025	19.06.2025	15.08.2024	31.10.2025	31.10.2024	06.01.2027	31.10.2027	20.09.2025	20.09.2024	01.11.2023					
8	29.05.2023	01.11.2024	01.11.2024	30.05.2024	08.03.2028	31.10.2028	31.10.2028	31.10.2028	15.06.2028	31.10.2025	31.10.2028	01.11.2025	01.11.2025	01.11.2024	19.11.2025	20.11.2024	06.01.2028	31.10.2028	31.10.2025	31.10.2024	06.01.2024					

Here are all the public holidays from 2023 to 2100. You can manually modify the list, such as adding or removing holidays. This list can be automatically generated with the program Feiertage_generieren.xlsm provided below, but it does not account for historical changes to holidays (e.g., the abolition of the "Buß- und Bettag" since 1995, except in Saxony) or territorial reforms (e.g., the reunification in 1990).

For school holidays (better: school-free days), the list of public holidays can be easily expanded: simply add all the individual holiday days to the corresponding state columns and rename the custom function "*IstFeiertag*" to "*IstSchulfreierTag*".

IstFeiertag Code

```
Enum Spalten
    col_LBound = 0
    col_DE  'Deutschland
    col_BW  'Baden_Württemberg
    col_BY  'Bayern
    col_BYK 'Bayern (überw. kath.)
    col_BE  'Berlin
    col_BB  'Brandenburg
    col_HB  'Bremen
    col_HH  'Hamburg
    col_HE  'Hessen
    col_MV  'Mecklenburg-Vorpommern
    col_NI  'Niedersachsen
    col_NW  'Nordrhein-Westfalen
    col_RP  'Rheinland-Pfalz
    col_SL  'Saarland
    col_SN  'Sachsen
    col_SNK 'Sachsen (einig. kath.)
    col_ST  'Sachsen-Anhalt
    col_SH  'Schleswig-Holstein
    col_TH  'Thüringen
    col_THK 'Thüringen (einig. kath.)
    col_AO  'Mein Arbeitsort
    col_UBound
End Enum

Function IstFeiertag(dt As Date, _
    Optional Land As String = "DE") As Variant
'Prüft, ob ein Datum ein Feiertag ist, mit Land = "DE"
'ob es ein bundeseinheitlicher Feiertag ist, sonst
'ob bundeseinheitlich oder vom entsprechenden Bundesland
'oder der individuelle "mein Arbeitsort".
'(C) (P) by Bernd Plumhoff 18-Jan-2024 PB V0.2
Static Feiertage    As Variant
Static LetzteZeile  As Variant
Dim d               As Date
Dim i               As Long
Dim j               As Long
Dim s               As String
Dim ws              As Worksheet

With Application.WorksheetFunction

If dt = 0# Or Land = "" Then
    IstFeiertag = CVErr(xlErrNull)
    Exit Function
End If
```

```
Set ws = Sheets("Feiertage")

If IsEmpty(Feiertage) Then
    LetzteZeile = ws.Cells(2, 1).End(xlDown).Row
    Set Feiertage = Range(ws.Cells(3, 1), _
        ws.Cells(LetzteZeile, col_UBound - 1))
End If

i = 1
s = ws.Cells(2, i)
Do While s <> ""
    If Land = s Then Exit Do
    i = i + 1
    s = ws.Cells(2, i)
Loop

If s = "" Then
    IstFeiertag = CVErr(xlErrName)
    Exit Function
End If

IstFeiertag = False

'Bundesweiter Feiertag?
j = 1
d = Feiertage(j, 1)
Do While j < LetzteZeile - 2
    If dt = d Then
        IstFeiertag = True
        Exit Function
    End If
    j = j + 1
    d = Feiertage(j, 1)
Loop

'Bundesland Feiertag?
If i > 1 Then
    j = 1
    d = Feiertage(j, i)
    Do While j < LetzteZeile - 2
        If dt = d Then
            IstFeiertag = True
            Exit Function
        End If
        j = j + 1
        d = Feiertage(j, i)
    Loop
End If

End With

End Function
```

Feiertagsliste_erstellen Code

```
Const StartJahr = 2023
Const EndJahr = 2100

Enum Spalten
    col_LBound = 0
    col_DE  'Deutschland
    col_BW  'Baden_Württemberg
    col_BY  'Bayern
    col_BYK 'Bayern (überw. kath.)
    col_BE  'Berlin
    col_BB  'Brandenburg
    col_HB  'Bremen
    col_HH  'Hamburg
    col_HE  'Hessen
    col_MV  'Mecklenburg-Vorpommern
    col_NI  'Niedersachsen
    col_NW  'Nordrhein-Westfalen
    col_RP  'Rheinland-Pfalz
    col_SL  'Saarland
    col_SN  'Sachsen
    col_SNK 'Sachsen (einig. kath.)
    col_ST  'Sachsen-Anhalt
    col_SH  'Schleswig-Holstein
    col_TH  'Thüringen
    col_THK 'Thüringen (einig. kath.)
    col_AO  'Mein Arbeitsort
    col_UBound
End Enum

Sub Feiertagsliste_erstellen()
```

```
'Generiert Feiertage für die Jahre StartJahr bis EndJahr.
'(C) (P) by Bernd Plumhoff 18-Jan-2024 PB V0.2

Dim Advent4                             As Date
Dim Easter                              As Date

Dim i                                   As Long
Dim r(col_LBound + 1 To col_UBound - 1) As Long

Dim state                               As SystemState

Set state = New SystemState

wsGen.Range("3:100000").Delete
For i = col_LBound + 1 To col_UBound - 1
    r(i) = 3
Next i

For i = StartJahr To EndJahr

    Advent4 = DateSerial(i, 12, 25) - Weekday(DateSerial(i, 12, 25), 2)
    Easter = EasterUSNO(i)

    'Deutschland
    wsGen.Cells(r(col_DE), col_DE) = DateSerial(i, 1, 1)    'Neujahr
    r(col_DE) = r(col_DE) + 1
    wsGen.Cells(r(col_DE), col_DE) = Easter - 2             'Karfreitag
    r(col_DE) = r(col_DE) + 1
    wsGen.Cells(r(col_DE), col_DE) = Easter + 1             'Ostermontag
    r(col_DE) = r(col_DE) + 1
    wsGen.Cells(r(col_DE), col_DE) = DateSerial(i, 5, 1)    'Maifeiertag
    r(col_DE) = r(col_DE) + 1
    wsGen.Cells(r(col_DE), col_DE) = Easter + 39            'Himmelfahrt
    r(col_DE) = r(col_DE) + 1
    wsGen.Cells(r(col_DE), col_DE) = Easter + 50            'Pfingstmontag
    r(col_DE) = r(col_DE) + 1
    wsGen.Cells(r(col_DE), col_DE) = DateSerial(i, 10, 3)   'Tag der deutschen Einheit
    r(col_DE) = r(col_DE) + 1
    wsGen.Cells(r(col_DE), col_DE) = DateSerial(i, 12, 25)  '1. Weihnachtstag
    r(col_DE) = r(col_DE) + 1
    wsGen.Cells(r(col_DE), col_DE) = DateSerial(i, 12, 26)  '2. Weihnachtstag
    r(col_DE) = r(col_DE) + 1

    'Baden-Württemberg
    wsGen.Cells(r(col_BW), col_BW) = DateSerial(i, 1, 6)    'Hl. Drei Könige
    r(col_BW) = r(col_BW) + 1
    wsGen.Cells(r(col_BW), col_BW) = Easter + 60            'Fronleichman
    r(col_BW) = r(col_BW) + 1
    wsGen.Cells(r(col_BW), col_BW) = DateSerial(i, 11, 1)   'Allerheiligen
    r(col_BW) = r(col_BW) + 1

    'Bayern
    wsGen.Cells(r(col_BY), col_BY) = DateSerial(i, 1, 6)    'Hl. Drei Könige
    r(col_BY) = r(col_BY) + 1
    wsGen.Cells(r(col_BY), col_BY) = Easter + 60            'Fronleichman
    r(col_BY) = r(col_BY) + 1
    wsGen.Cells(r(col_BY), col_BY) = DateSerial(i, 11, 1)   'Allerheiligen
    r(col_BY) = r(col_BY) + 1

    'Bayern (überwiegend katholische Bevölkerung)
    wsGen.Cells(r(col_BYK), col_BYK) = DateSerial(i, 1, 6)  'Hl. Drei Könige
    r(col_BYK) = r(col_BYK) + 1
    wsGen.Cells(r(col_BYK), col_BYK) = Easter + 60          'Fronleichman
    r(col_BYK) = r(col_BYK) + 1
    wsGen.Cells(r(col_BYK), col_BYK) = DateSerial(i, 8, 15) 'Mariä Himmelfahrt
    r(col_BYK) = r(col_BYK) + 1
    wsGen.Cells(r(col_BYK), col_BYK) = DateSerial(i, 11, 1) 'Allerheiligen
    r(col_BYK) = r(col_BYK) + 1

    'Berlin
    wsGen.Cells(r(col_BE), col_BE) = DateSerial(i, 3, 8)    'Int. Frauentag
    r(col_BE) = r(col_BE) + 1

    'Brandenburg
    wsGen.Cells(r(col_BB), col_BB) = DateSerial(i, 10, 31)  'Reformationstag
    r(col_BB) = r(col_BB) + 1

    'Bremen
    wsGen.Cells(r(col_HB), col_HB) = DateSerial(i, 10, 31)  'Reformationstag
    r(col_HB) = r(col_HB) + 1

    'Hamburg
    wsGen.Cells(r(col_HH), col_HH) = DateSerial(i, 10, 31)  'Reformationstag
    r(col_HH) = r(col_HH) + 1

    'Hessen
    wsGen.Cells(r(col_HE), col_HE) = Easter + 60            'Fronleichman
    r(col_HE) = r(col_HE) + 1

    'Mecklenburg-Vorpommern
```

```
    wsGen.Cells(r(col_MV), col_MV) = DateSerial(i, 3, 8)    'Int. Frauentag
    r(col_MV) = r(col_MV) + 1
    wsGen.Cells(r(col_MV), col_MV) = DateSerial(i, 10, 31)  'Reformationstag
    r(col_MV) = r(col_MV) + 1

    'Niedersachsen
    wsGen.Cells(r(col_NI), col_NI) = DateSerial(i, 10, 31)  'Reformationstag
    r(col_NI) = r(col_NI) + 1

    'Nordrhein-Westfalen
    wsGen.Cells(r(col_NW), col_NW) = Easter + 60            'Fronleichman
    r(col_NW) = r(col_NW) + 1
    wsGen.Cells(r(col_NW), col_NW) = DateSerial(i, 11, 1)   'Allerheiligen
    r(col_NW) = r(col_NW) + 1

    'Rheinland-Pfalz
    wsGen.Cells(r(col_RP), col_RP) = Easter + 60            'Fronleichman
    r(col_RP) = r(col_RP) + 1
    wsGen.Cells(r(col_RP), col_RP) = DateSerial(i, 11, 1)   'Allerheiligen
    r(col_RP) = r(col_RP) + 1

    'Saarland
    wsGen.Cells(r(col_SL), col_SL) = Easter + 60            'Fronleichman
    r(col_SL) = r(col_SL) + 1
    wsGen.Cells(r(col_SL), col_SL) = DateSerial(i, 8, 15)   'Mariä Himmelfahrt
    r(col_SL) = r(col_SL) + 1
    wsGen.Cells(r(col_SL), col_SL) = DateSerial(i, 11, 1)   'Allerheiligen
    r(col_SL) = r(col_SL) + 1

    'Sachsen
    wsGen.Cells(r(col_SN), col_SN) = DateSerial(i, 10, 31)  'Reformationstag
    r(col_SN) = r(col_SN) + 1
    wsGen.Cells(r(col_SN), col_SN) = Advent4 - 32           'Buß- und Bettag
    r(col_SN) = r(col_SN) + 1

    'Sachsen (einige katholische Gemeinden)
    wsGen.Cells(r(col_SNK), col_SNK) = Easter + 60          'Fronleichman
    r(col_SNK) = r(col_SNK) + 1
    wsGen.Cells(r(col_SNK), col_SNK) = DateSerial(i, 10, 31) 'Reformationstag
    r(col_SNK) = r(col_SNK) + 1
    wsGen.Cells(r(col_SNK), col_SNK) = Advent4 - 32         'Buß- und Bettag
    r(col_SNK) = r(col_SNK) + 1

    'Sachsen-Anhalt
    wsGen.Cells(r(col_ST), col_ST) = DateSerial(i, 1, 6)    'Hl. Drei Könige
    r(col_ST) = r(col_ST) + 1

    'Schleswig-Holstein
    wsGen.Cells(r(col_SH), col_SH) = DateSerial(i, 10, 31)  'Reformationstag
    r(col_SH) = r(col_SH) + 1

    'Thüringen
    wsGen.Cells(r(col_TH), col_TH) = DateSerial(i, 9, 20)   'Weltkindertag
    r(col_TH) = r(col_TH) + 1
    wsGen.Cells(r(col_TH), col_TH) = DateSerial(i, 10, 31)  'Reformationstag
    r(col_TH) = r(col_TH) + 1

    'Thüringen (einige katholische Gemeinden)
    wsGen.Cells(r(col_THK), col_THK) = Easter + 60          'Fronleichman
    r(col_THK) = r(col_THK) + 1
    wsGen.Cells(r(col_THK), col_THK) = DateSerial(i, 9, 20) 'Weltkindertag
    r(col_THK) = r(col_THK) + 1
    wsGen.Cells(r(col_THK), col_THK) = DateSerial(i, 10, 31) 'Reformationstag
    r(col_THK) = r(col_THK) + 1

    'Mein Arbeitsort (nehmen wir einmal an, Augsburg)
    wsGen.Cells(r(col_AO), col_AO) = DateSerial(i, 1, 6)    'Hl. Drei Könige
    r(col_AO) = r(col_AO) + 1
    wsGen.Cells(r(col_AO), col_AO) = Easter + 60            'Fronleichman
    r(col_AO) = r(col_AO) + 1
    wsGen.Cells(r(col_AO), col_AO) = DateSerial(i, 8, 8)    'Hohes Friedensfest Augsburg
    r(col_AO) = r(col_AO) + 1
    wsGen.Cells(r(col_AO), col_AO) = DateSerial(i, 8, 15)   'Mariä Himmelfahrt
    r(col_AO) = r(col_AO) + 1
    wsGen.Cells(r(col_AO), col_AO) = DateSerial(i, 11, 1)   'Allerheiligen
    r(col_AO) = r(col_AO) + 1

Next i
End Sub

Public Function EasterUSNO(YYYY As Long) As Long
'Source: http://www.cpearson.com/excel/easter.aspx
End Function
```

Present the Full-Length Number – *sbNum2Str*

If you need a non-scientific number representation with all significant digits and all leading and trailing zeros you can use this function *sbNum2Str*:

B2 =sbNum2Str(A2)

	A	B
1	**Input**	**sbNum2Str**
2	3,141592654	3,14159265358979
3	-1	-1
4	-1,00E+307	- 100000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000000000 00000000000000000000000
5	-1,00E-307	- 0,0000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000000000 000000000000000000000001
6	-0,001	-0,001
7	-123	-123
8	-0,1	-0,1

sbNum2Str Code

```
Function sbNum2Str(d As Double) As String
'Returns string with number representation with all
'significant digits and leading or trailing zeros, i.e.
'1E+3 will be returned as 1000
'1E-3 will be 0.001
'Pi() will be 3.14159265358979
'(C) (P) by Bernd Plumhoff 15-Nov-2010 PB V0.20
Dim v
Dim lExp As Long, lLenMant As Long
Dim sDot As String 'decimal separator
Dim sMant As String 'new mantissa

If d < 0# Then
    sbNum2Str = "-" & sbNum2Str(-d)
    Exit Function
End If
sDot = Application.DecimalSeparator
'Split scientific representation into mantissa and exponent
v = Split(Format(d, _
        "0." & String(15, "#") & "E+0"), "E")
If Left(v(0), 1) = "0" Then
    sbNum2Str = "0"
    Exit Function
End If
lExp = CLng(v(1))    'get exponent
v = Split(v(0), sDot)
If lExp < 0 Then
    sbNum2Str = "0" & sDot & String(-lExp - 1, "0") & _
              v(0) & v(1)
Else
    lLenMant = Len(v(1))
    If Len(v(1)) > lExp Then
        sMant = v(0) & v(1)
        sbNum2Str = Left(sMant, lExp + 1) & sDot & _
                  Right(sMant, Len(sMant) - lExp - 1)
    Else
        sbNum2Str = v(0) & v(1) & String(lExp - lLenMant, "0")
    End If
End If
End Function
```

Significant Digits of a Number – *sbNSig*

Significant digits of a number are the digits which contribute to its accuracy - generally these are all digits except leading and trailing zeros. For a more precise definition please see https://mathworld.wolfram.com/SignificantDigits.html

A worksheet function solution to calculate n significant digits of d – I found this in an Excel forum but I could not identify the original author:
A1: d
A2: n
Result: `=--TEXT(A1,"."&REPT("0",A2)&"E+0")`

Example: If you want to show pi with 3 significant digits:

E2 | =sbnsig(A2,B2)

	A	B	C	D	E	F
1	Number	Significant Digits	Result WS Fct	Formula	Result VBA	Formula
2	3.141592654	3	3.14	=--TEXT(A2,"."&REPT("0",B2)&"E+0")	3.14	=sbnsig(A2,B2)
3	2.718281828	3	2.72		2.72	
4	0.01234	2	0.012		0.012	
5	555000	2	560000		560000	
6	987	1	1000		1000	

A related function is *sbNum2Str* (see previous chapter) which returns a non-scientific string representation of a given number with all significant digits and all leading and trailing zeros.

sbNSig Program Code

```
Function sbNSig(d As Double, n As Long) As Double
'Returns double with n most significant digits of d.
'09-Nov-2019 by www.sulprobil.de PB V1.00
sbNSig = Format(d, "." & String(n, "0") & "E+0")
End Function
```

Return the Number for a Month's Name – *sbMonthNumber*

If you need the number for a month:

English		Deutsch		Schweizer Deutsch	
Month	Zahl	Monat	Zahl	Monet	Zahl
January	1	Januar	1	Jänner	1
February	2	Februar	2	Hornig	2
March	3	März	3	Merze	3
April	4	April	4	Abrele	4
May	5	Mai	5	Mäie	5
June	6	Juni	6	Brachet	6
July	7	Juli	7	Heuet	7
August	8	August	8	Augschte	8
September	9	September	9	Herbschtmonet	9
October	10	Oktober	10	Wiimonet	10
November	11	November	11	Wintermonet	11
December	12	Dezember	12	Chrischtmonet	12

sbMonthNumber Code

```
Function sbMonthNumber(sMonat As String) As Integer
'Return the number for a month's name.
'(C) (P) by Bernd Plumhoff  19-Nov-2022 PB V0.2
Dim s As String, c1 As String, c2 As String, c3 As String, c4 As String

s = Left(LCase(sMonat) & String(4, " "), 4)
c1 = Left(s, 1)
Select Case c1
Case "a"
    c2 = Mid(s, 2, 1)
    Select Case c2
    Case "b", "p"
        sbMonthNumber = 4 'Abrele, April, Aprilius
    Case "u"
        sbMonthNumber = 8 'August, Augschte, Augusti
    Case Else
        sbMonthNumber = CVErr(xlErrNum)
    End Select
Case "b"
    sbMonthNumber = 6 'Brachet, Brachmond
Case "c", "d"
    sbMonthNumber = 12 'Chrischtmonet, Dezember, December, Decembris, Christmond
Case "e"
    sbMonthNumber = 8 'Erntemond
Case "f"
    sbMonthNumber = 2 'Februar, February, Feber
Case "h"
    c2 = Mid(s, 2, 1)
    Select Case c2
    Case "a"
        sbMonthNumber = 1 'Hartung
    Case "e"
        c3 = Mid(s, 3, 1)
        Select Case c3
        Case "r"
            sbMonthNumber = 9 'Herbschtmonet, Herbstmond
        Case "u"
            sbMonthNumber = 7 'Heuet, Heuert, Heumond
        Case Else
            sbMonthNumber = CVErr(xlErrNum)
        End Select
    Case "o"
        sbMonthNumber = 2 'Hornig, Hornung
    Case Else
        sbMonthNumber = CVErr(xlErrNum)
    End Select
Case "j"
    c2 = Mid(s, 2, 1)
    Select Case c2
    Case "a", "ä", "e"
        sbMonthNumber = 1 'Januar, January, Jänner, Jenner
    Case "u"
        c3 = Mid(s, 3, 1)
        Select Case c3
```

```
        Case "l"
            c4 = Mid(s, 4, 1)
            Select Case c4
            Case "e", "i", "y"
                sbMonthNumber = 7 'Juli, July, Juley
            Case "m"
                sbMonthNumber = 12 'Julmond
            Case Else
                sbMonthNumber = CVErr(xlErrNum)
            End Select
        Case "n"
            sbMonthNumber = 6 'Juni, June, Juno
        Case Else
            sbMonthNumber = CVErr(xlErrNum)
        End Select
    Case Else
        sbMonthNumber = CVErr(xlErrNum)
    End Select
Case "l"
    sbMonthNumber = 3 'Lenzmond
Case "m"
    c2 = Mid(s, 2, 1)
    Select Case c2
    Case "a"
        c3 = Mid(s, 3, 1)
        Select Case c3
        Case "i", "y"
            sbMonthNumber = 5 'Mai, May
        Case "r"
            sbMonthNumber = 3 'March, Marty, Martii
        Case Else
            sbMonthNumber = CVErr(xlErrNum)
        End Select
    Case "ä"
        c3 = Mid(s, 3, 1)
        Select Case c3
        Case "i"
            sbMonthNumber = 5 'Mäie
        Case "r"
            sbMonthNumber = 3 'März
        Case Else
            sbMonthNumber = CVErr(xlErrNum)
        End Select
    Case "e"
        sbMonthNumber = 3 'Merze
    Case Else
        sbMonthNumber = CVErr(xlErrNum)
    End Select
Case "n"
    sbMonthNumber = 11 'November, Nebelmond
Case "o"
    c2 = Mid(s, 2, 1)
    Select Case c2
    Case "c", "k"
        sbMonthNumber = 10 'Oktober, October, Oktobris
    Case "s"
        sbMonthNumber = 4 'Ostermond
    Case Else
        sbMonthNumber = CVErr(xlErrNum)
    End Select
Case "s"
    c2 = Mid(s, 2, 1)
    Select Case c2
    Case "a"
        sbMonthNumber = 4 'Saating
    Case "c"
        c3 = Mid(s, 3, 1)
        Select Case c3
        Case "h"
            c4 = Mid(s, 4, 1)
            Select Case c4
            Case "e"
                sbMonthNumber = 9 'Scheiding
            Case "n"
                sbMonthNumber = 1 'Schneemond
            Case Else
                sbMonthNumber = CVErr(xlErrNum)
            End Select
        Case Else
            sbMonthNumber = CVErr(xlErrNum)
        End Select
    Case "e"
        sbMonthNumber = 9 'September, Septembris
    Case Else
        sbMonthNumber = CVErr(xlErrNum)
    End Select
Case "w"
    sbMonthNumber = 9 'September
    c2 = Mid(s, 2, 1)
    Select Case c2
```

```
    Case "e"
        sbMonthNumber = 10 'Weinmond
    Case "i"
        c3 = Mid(s, 3, 1)
        Select Case c3
        Case "i"
            sbMonthNumber = 10 'Wiimonet
        Case "n"
            sbMonthNumber = 11 'Wintermonet
        Case Else
            sbMonthNumber = CVErr(xlErrNum)
        End Select
    Case "o"
        sbMonthNumber = 5 'Wonnemond
    Case Else
        sbMonthNumber = CVErr(xlErrNum)
    End Select
Case Else
    sbMonthNumber = CVErr(xlErrNum)
End Select
End Function
```

Calculation of the Circle Constant π

How many decimal places of the circle constant π are needed for practical purposes?

The Hamburg mathematics professor Hermann Schubert demonstrated in 1889 how many decimal places are no longer necessary: Imagine a sphere with the Earth at its center. The radius is as large as the distance from Earth to Sirius: 8.8 light years. This sphere is filled with microbes, many millions of which fit into a cubic millimeter. Now, string all the microbes in this sphere together on a taut thread, leaving a space of 8.8 light years between each pair of microbes. This thread is taken as the diameter of a circle. If this length is multiplied by π (to 100 decimal places), the result will differ from the exact circumference of the circle by less than one millionth of a millimeter.

This example shows that calculating π to 100 or more places is completely useless.

Why then is there an effort to calculate π to ever more decimal places?

Recently, there has been an interest in discovering whether the sequence of decimal digits follows certain laws or not. It is also unknown how frequently each digit occurs in the decimal representation.

Until the mid-17th century, π was calculated using the method of Archimedes. He approximated the area of the unit circle using polygonal areas. James Gregory discovered this power series in 1671

$$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} - + \ldots$$

for -1 <= x <= 1.

With this series expansion and the equation, π has been calculated recently:

$$(*)\ \arctan x + \arctan y = \arctan \frac{x+y}{1-xy}$$

Pi Program Code

```
Const n = 1050
Const z1 = 554
Const z2 = 285
Const z3 = 211
Dim m1            As Integer
Dim m2            As Integer
Dim m3            As Integer
Dim a(n)          As Integer
Dim u_b(n)        As Integer
Dim c(n)          As Integer
Dim d(n)          As Integer
Dim p(n)          As Integer
Dim i             As Integer
Dim j             As Integer
Dim k             As Integer
Dim r             As Integer
Dim u             As Integer
Dim v             As Integer
Dim x             As Integer
Dim y             As Integer

Sub addiere()
u = 0
For j = n To 0 Step -1
  p(j) = p(j) + a(j) + u
  u = p(j) \ 10
  p(j) = p(j) - 10 * u
Next j
```

```
End Sub

Sub subtrahiere()
u = 1
For j = n To 0 Step -1
  p(j) = p(j) + 9 - a(j) + u
  u = p(j) \ 10
  p(j) = p(j) - 10 * u
Next j
End Sub

Sub pi()
' pi auf 1000 Stellen ausgeben
For i = 0 To n
  a(i) = 0
  u_b(i) = 0
  c(i) = 0
  d(i) = 0
  p(i) = 0
Next i
m1 = 8
m2 = 57
m3 = 239
'Der erste Summand jeder Reihe wird ermittelt
'b(I) = 24 \ 8
u_b(0) = 24 \ 8
p(0) = u_b(0)
c(0) = 8
r = 0   'r enthält immer den Rest der Division
'c(I) = 8\57
For i = 0 To n
  a(i) = (10 * r + c(i)) \ m2
  r = 10 * r + c(i) - a(i) * m2
Next i
For i = 0 To n
  c(i) = a(i)
Next i
addiere
d(0) = 4
r = 0
'd(I) = 4\239
For i = 0 To n
  a(i) = (10 * r + d(i)) \ m3
  r = 10 * r + d(i) - a(i) * m3
Next i
For i = 0 To n
  d(i) = a(i)
Next i
addiere
' Nun wird die Reihe von 24 arctan 1\8 berechnet
v = -1  'Das Vorzeichen des Summanden
m1 = m1 * m1
k = 3
For i = 1 To z1
  r = 0
  For j = 0 To n
    a(j) = (10 * r + u_b(j)) \ m1
    r = 10 * r + u_b(j) - a(j) * m1
  Next j
  For j = 0 To n
    u_b(j) = a(j)
  Next j
  r = 0
  For j = 0 To n
    a(j) = (10 * r + u_b(j)) \ k
    r = 10 * r + u_b(j) - a(j) * k
  Next j
  If v = 1 Then addiere Else subtrahiere
  k = k + 2
  v = 0 - v
Next i
' Nun wird die Reihe von 8 arctan 1\57 berechnet
v = -1
m2 = m2 * m2
k = 3
For i = 1 To z2
  r = 0
  For j = 0 To n
    a(j) = (10 * r + c(j)) \ m2
    r = 10 * r + c(j) - a(j) * m2
  Next j
  For j = 0 To n
    c(j) = a(j)
  Next j
  r = 0
  For j = 0 To n
    a(j) = (10 * r + c(j)) \ k
    r = 10 * r + c(j) - a(j) * k
  Next j
  If v = 1 Then addiere Else subtrahiere
```

```
  k = k + 2
  v = 0 - v
Next i
' Nun wird die Reihe von 4 arctan 1\239 berechnet
v = -1
k = 3
For i = 1 To z3
  r = 0
  For j = 0 To n
    a(j) = (10 * r + d(j)) \ m3
    r = 10 * r + d(j) - a(j) * m3
  Next j
  r = 0
  For j = 0 To n
    d(j) = (10 * r + d(j)) \ m3
    r = 10 * r + a(j) - d(j) * m3
  Next j
  r = 0
  For j = 0 To n
    a(j) = (10 * r + d(j)) \ k
    r = 10 * r + d(j) - a(j) * k
  Next j
  If v = 1 Then addiere Else subtrahiere
  k = k + 2
  v = 0 - v
Next i
x = 1
y = 1
Open ThisWorkbook.Path & "/pi.txt" For Output As #1
Print #1, "Pi = " & p(0) & ".";
For i = 1 To 1000
  Print #1, Format(p(i), "&");
  x = x + 1
  If x > 3 Then
    Print #1, " ";
    x = 1
  End If
  y = y + 1
  If y > 42 Then
    Print #1, " "
    Print #1, "        ";
    y = 1
  End If
Next i
Close #1
End Sub
```

First 1,000 Digits of π

```
Pi = 3.141 592 653 589 793 238 462 643 383 279 502 884 197 169
       399 375 105 820 974 944 592 307 816 406 286 208 998 628
       034 825 342 117 067 982 148 086 513 282 306 647 093 844
       609 550 582 231 725 359 408 128 481 117 450 284 102 701
       938 521 105 559 644 622 948 954 930 381 964 428 810 975
       665 933 446 128 475 648 233 786 783 165 271 201 909 145
       648 566 923 460 348 610 454 326 648 213 393 607 260 249
       141 273 724 587 006 606 315 588 174 881 520 920 962 829
       254 091 715 364 367 892 590 360 011 330 530 548 820 466
       521 384 146 951 941 511 609 433 057 270 365 759 591 953
       092 186 117 381 932 611 793 105 118 548 074 462 379 962
       749 567 351 885 752 724 891 227 938 183 011 949 129 833
       673 362 440 656 643 086 021 394 946 395 224 737 190 702
       179 860 943 702 770 539 217 176 293 176 752 384 674 818
       467 669 405 132 000 568 127 145 263 560 827 785 771 342
       757 789 609 173 637 178 721 468 440 901 224 953 430 146
       549 585 371 050 792 279 689 258 923 542 019 956 112 129
       021 960 864 034 418 159 813 629 774 771 309 960 518 707
       211 349 999 998 372 978 049 951 059 731 732 816 096 318
       595 024 459 455 346 908 302 642 522 308 253 344 685 035
       261 931 188 171 010 003 137 838 752 886 587 533 208 381
       420 617 177 669 147 303 598 253 490 428 755 468 731 159
       562 863 882 353 787 593 751 957 781 857 780 532 171 226
       806 613 001 927 876 611 195 909 216 420 198 9
```

Calculation of Euler's Number *e*

To calculate Euler's number *e*, the well-known summation formula

$$e = \sum_{i=0}^{\infty} \frac{1}{i!}$$

is used. This means

$$e - 2 = \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \frac{1}{120} + \ldots$$

Any real number 0≤x<10≤x<1 can be represented as a (possibly infinite) "variable base" fraction .a1a2a3…, where for all i, 0≤a(i)<10≤a(i)<1. The represented number is

$$x = \sum_{i=1}^{\infty} \frac{1}{(i+1)!}$$

The real number *e−2* is thus represented by the infinite fraction .111… . The problem of calculating *e* is therefore reduced to converting the above representation into decimal form. The more digits one wants to determine for the decimal representation, the "longer" the "variable base" fraction must be that is being converted.

Example: Calculation of *e* to 1000 decimal places. From which index n can the remainder of the series be neglected? Let y(n) be the (n+1)-th partial sum of the infinite series

$$\sum_{i=0}^{\infty} \frac{1}{i!}$$

It holds (Quelle: Fichtenholz Volume I, Nr. 37):

$$0 < e - y_n < \frac{1}{n!n}$$

For this example it holds:

$$e - y_n < \frac{1}{n!n} < 10^{-1000}$$

For this to be true you need to choose n = 500.

e Program Code

```
Sub e()

'eFehler! Textmarke nicht definiert. auf 1000 Stellen ausgeben

Dim a(500) As Long
Dim c      As Long
Dim d      As Long
Dim i      As Long
Dim j      As Long
Dim x      As Integer
Dim y      As Integer

Open ThisWorkbook.Path & "/e.txt" For Output As #1
Print #1, "e = 2.";

For i = 1 To 500
  a(i) = 1
Next i

For i = 1 To 1000
  c = 0
  For j = 500 To 1 Step -1
    d = 10 * a(j) + c
    c = Fix(d / (j + 1))
    a(j) = d - c * (j + 1)
  Next j
  x = x + 1
  If x > 3 Then
    Print #1, " ";
    x = 1
  End If
  y = y + 1
  If y > 42 Then
    Print #1, " "
    Print #1, "      ";
    y = 1
  End If
  Print #1, Format(c, "&");
Next i

Close #1

End Sub
```

First 1,000 Digits of e

```
e = 2.718 281 828 459 045 235 360 287 471 352 662 497 757 247
    093 699 959 574 966 967 627 724 076 630 353 547 594 571
    382 178 525 166 427 427 466 391 932 003 059 921 817 413
    596 629 043 572 900 334 295 260 595 630 738 132 328 627
    943 490 763 233 829 880 753 195 251 019 011 573 834 187
    930 702 154 089 149 934 884 167 509 244 761 460 668 082
    264 800 168 477 411 853 742 345 442 437 107 539 077 744
    992 069 551 702 761 838 606 261 331 384 583 000 752 044
    933 826 560 297 606 737 113 200 709 328 709 127 443 747
    047 230 696 977 209 310 141 692 836 819 025 515 108 657
    463 772 111 252 389 784 425 056 953 696 770 785 449 969
    967 946 864 454 905 987 931 636 889 230 098 793 127 736
    178 215 424 999 229 576 351 482 208 269 895 193 668 033
    182 528 869 398 496 465 105 820 939 239 829 488 793 320
    362 509 443 117 301 238 197 068 416 140 397 019 837 679
    320 683 282 376 464 804 295 311 802 328 782 509 819 455
    815 301 756 717 361 332 069 811 250 996 181 881 593 041
    690 351 598 888 519 345 807 273 866 738 589 422 879 228
    499 892 086 805 825 749 279 610 484 198 444 363 463 244
    968 487 560 233 624 827 041 978 623 209 002 160 990 235
    304 369 941 849 146 314 093 431 738 143 640 546 253 152
    096 183 690 888 707 016 768 396 424 378 140 592 714 563
    549 061 303 107 208 510 383 750 510 115 747 704 171 898
    610 687 396 965 521 267 154 688 957 035 035 4
```

Literature

Nievergelt, Farrer, Reingold: Computer Approaches to Mathematical Problems; Prentice Hall, Inc. 1974, p. 191 + 192, p. 198 - 202.

Ullman: Fundamental Concepts of Programming Systems; Addison-Wesley Publishing Company 1976, p. 48 + 49.

Fichtenholz; Differential- und Integralrechnung Band I + II; VEB Deutscher Verlag der Wissenschaften 1981, Nr. 37 + 50 + 410.

Shorten a Number Sequence Representation – sbParseNumSeq

Parse a comma-separated number sequence and return a shortened representation: 1,2,3,5,6,7 will result in 1-3,5-7. If bWithSingleDouble = TRUE then 1,3,5,6,8,10 will result in 1-5(single),6-10(double).

	A	B	C
1	**Input**	**bWithSingleDouble**	**Output**
2	1,2,3,4,8,11,12,16,17,18	WAHR	1-4,8,11-12,16-18
3	3,5,6,7,9,12,13,14,15,20,101	WAHR	3,5-7,9,12-15,20,101
4	1,3,5,7,9,11	WAHR	1-11(single)
5	1	WAHR	1
6	2,4,6,8,10,12,14,16,17	WAHR	2-16(double),17
7	1,5,6,7,9,12,13,14,15,16	WAHR	1,5-7,9,12-16
8	2,3,4,5,6,7,10,17,22,23,24,25,26,77	WAHR	2-7,10,17,22-26,77
9	3,4,7,13,15,16,17	WAHR	3-4,7,13,15-17
10	1,2,3,4,5,6,7,13,15,17	WAHR	1-7,13-17(single)
11	3,5,7,13,22,24,26,28,30,32,34,36,38,40,42,44,46,48,50,52,54,56,58,60,62,64,66,68,70,72,74,76,78,80	WAHR	3-7(single),13,22-80(double)
12	2,3,4,7,12,14,17,18	WAHR	2-4,7,12-14(double),17-18
13	1,3,4,5,7,9,11,12,14,16	WAHR	1,3-4,5-11(single),12-16(double)
14	3,4,5,7,9,11	WAHR	3-4,5-11(single)
15	1,2,3,4,8,11,12,16,17,18	FALSCH	1-4,8,11-12,16-18
16	3,5,6,7,9,12,13,14,15,20,101	FALSCH	3,5-7,9,12-15,20,101
17	1,3,5,7,9,11	FALSCH	1,3,5,7,9,11
18	1	FALSCH	1
19	2,4,6,8,10,12,14,16,17	FALSCH	2,4,6,8,10,12,14,16-17
20	1,5,6,7,9,12,13,14,15,16	FALSCH	1,5-7,9,12-16
21	2,3,4,5,6,7,10,17,22,23,24,25,26,77	FALSCH	2-7,10,17,22-26,77
22	3,4,7,13,15,16,17	FALSCH	3-4,7,13,15-17
23	1,2,3,4,5,6,7,13,15,17	FALSCH	1-7,13,15,17
24	3,5,7,13,22,24,26,28,30,32,34,36,38,40,42,44,46,48,50,52,54,56,58,60,62,64,66,68,70,72,74,76,78,80	FALSCH	3,5,7,13,22,24,26,28,30,32,34,36,38,40,42,44,46,48,50,52,54,56,58,60,62,64,66,68,70,72,74,76,78,80
25	2,3,4,7,12,14,17,18	FALSCH	2-4,7,12,14,17-18
26	1,3,4,5,7,9,11,12,14,16	FALSCH	1,3-5,7,9,11-12,14,16
27	3,4,5,7,9,11	FALSCH	3-5,7,9,11

sbParseNumSeq Code

```
Function sbParseNumSeq(s As String, _
  Optional bWithSingleDouble As Boolean = True) As String
'Parse a comma-separated number sequence and return a
'shortened representation:
'1,2,3,5,6,7 will result in 1-3,5-7.
'If bWithSingleDouble = TRUE then
'1,3,5,6,8,10 will result in 1-5(single),6-10(double).
'(C) (P) by Bernd Plumhoff 08-Sep-2024 PB V0.1
Dim i           As Long
Dim j           As Long
Dim k           As Long
Dim m           As Long
Dim sDel        As String
Dim suffix      As String
Dim r           As String
Dim v           As Variant

v = Split(s, ",")
j = UBound(v)
ReDim seq(0 To j, 0 To 2) As Long
For i = 0 To j - 1
  k = v(i + 1)
  If k = v(i) + 1 Then
    m = i + 1
    Do While m < j
      If v(m) + 1 = CLng(v(m + 1)) Then
        m = m + 1
      Else
        Exit Do
      End If
    Loop
    seq(i, 0) = 1
    seq(i, 1) = m - i
  ElseIf bWithSingleDouble And k = v(i) + 2 Then
    m = i + 1
    Do While m < j
      If v(m) + 2 = CLng(v(m + 1)) Then
        m = m + 1
      Else
        Exit Do
      End If
    Loop
    seq(i, 0) = 2
    seq(i, 2) = m - i
  End If
Next i
For i = 0 To j
  If seq(i, 0) = 0 Then
    r = r & sDel & v(i)
  Else
    k = seq(i, seq(i, 0))
    m = seq(i + k, seq(i + k, 0))
    If k > 0 And k >= m Then
      suffix = ""
      If seq(i, 0) = 2 Then
        If v(i) Mod 2 = 0 Then
          suffix = "(double)"
        Else
          suffix = "(single)"
        End If
      End If
      r = r & sDel & v(i) & "-" & v(i + k) & suffix
      i = i + k
    ElseIf k >= 2 Then
      suffix = ""
      If seq(i, 0) = 2 Then
        If v(i) Mod 2 = 0 Then
          suffix = "(double)"
        Else
          suffix = "(single)"
        End If
      End If
      r = r & sDel & v(i) & "-" & v(i + k - 1) & suffix
      i = i + k - 1
    Else
      r = r & sDel & v(i)
    End If
  End If
  sDel = ","
Next i
sbParseNumSeq = r
End Function
```

Rational Numbers = Fractions

Nearest Rational Number to a Given Floating Point Number – *sbNRN*

Which rational number is a good proxy of π (3.1415926…)? Enter in cell A1 '=pi()', in cell B1 your maximal denominator (for example 10), and in cells C1:D1 '=*sbNRN*(A1,B1)' as array formula (with CTRL + SHIFT + ENTER). You will get in C1:D1 22 and 7. That means: 22/7 is the nearest rational number to π with a denominator not higher than 10. For 1000 in B1 you would get 355/113.

This algorithm does not necessarily find the nearest rational number to a given floating point number with a given maximal denominator and the pre-defined maximal absolute error 1# / (2# * CDbl(*lMaxDen*) ^ 2#). The good message is, though, that it would then return a #NUM! error. In this case please try a larger individual maximal absolute error.

The author's (Oliver Aberth) original intention was to support exact computation with rational numbers, for example solving a set of linear equations with rational coefficients.

	Input			Result		Quality Measure			
dFloat	lMaxDen	dMaxErr	pK	qK		Absolute Error	# of "?"	TEXT Representation	Comment
3,14159265358979	1		3	1		0,141592654	1	22/7	
3,14159265358979	10		22	7		0,001264489	1	22/7	
3,14159265358979	112	0,001264489	333	106		8,32196E-05	3	355/113	
3,14159265358979	1000		355	113		2,66764E-07	3	355/113	
3,14159265358979	33214	2,66764E-07	103993	33102		5,77891E-10	5	312689/99532	
3,14159265358979	66316	5,77891E-10	104348	33215		3,31628E-10	5	312689/99532	
3,14159265358979	99531	3,31628E-10	208341	66317		1,22356E-10	5	312689/99532	
3,14159265358979	100000		312689	99532		2,91434E-11	5	312689/99532	
3,14159265358979	364912	2,91434E-11	833719	265381		8,71525E-12	6	1146408/364913	
3,14159265358979	1360119	8,71525E-12	1146408	364913		1,61071E-12	6	1146408/364913	
3,14159265358979	1725032	1,61071E-12	4272943	1360120		4,04121E-13	7	5419351/1725033	
3,14159265358979	25510581	4,04121E-13	5419351	1725033		2,22045E-14	7	5419351/1725033	
3,14159265358979	78256778	2,22045E-14	80143857	25510582		4,44089E-16	8	5419351/1725033	Accuracy limit of TEXT reached
3,14159265358979	100000000		245850922	78256779		0	8	5419351/1725033	Accuracy limit of TEXT reached

Note: The last row in this graphic does not tell us that we have successfully squared the circle. We have reached (my) Excel's limit of accuracy.

The fraction representations of the TEXT function are shown for comparison.
Example: =TEXT(PI(),"?/?") = "22/7"
Microsoft did not extend this representation for its 64-Bit version. It cannot get more accurate than PI() = "5419351/1725033". With 64-Bit PI() = "245850922/78256779" would be more accurate, but in this case the absolute error is already less than 1e-15, of course.

A simple sample application you find at Quota Change as Fraction.

<u>Calculation Limits</u>

Excel can represent decimal numbers from -9.99999999999999E+307 to 9.99999999999999E+307. Excel's 64-bit version can use integers of type LongLong from -9223372036854775808 to 9223372036854775807 which is about -1E+10 to 1E+10.

It is obvious that Aberth's algorithm cannot calculate sufficiently accurate fractions for all available decimal numbers with Excel.

Name

sbNRN - Compute nearest rational number to a given floating point number with a given maximal denominator

Synopsis

sbNRN(*dFloat*, *lMaxDen*, [*dMaxErr*])

Description

sbNRN computes the nearest rational number to a given floating point number *dFloat* with a given maximal denominator *lMaxDen* and an optional maximal error *dMaxErr*.

Parameter

dFloat - Floating point number for which you want to derive the nearest rational number

lMaxDen - Maximal denominator which you want to allow

dMaxErr - Optional - Maximal absolute error (absolute difference between input float and output rational number) which you want to allow for

<u>Literature</u>

Oliver Aberth, A method for exact computation with rational numbers, JCAM, vol 4, no. 4, 1978

Oliver Aberth, Introduction to Precise Numerical Methods, ISBN 0-12-373859-8

George Chrystal, Algebra an Elementary Text-Book, Part II, Chapter 32, p. 423 ff, 1900

Peter Henrici, A Subroutine for Computations with Rational Numbers, JACM, vol 3, no. 1, 1956

<u>Excursus</u>

In case you just need the relation to its power of 10, you can use the formula
`=IFERROR(-A2*10^(LEN(-A2)-SEARCH(",",-A2))&":"&10^(LEN(-A2)-SEARCH(",",-A2)),-A2&":1")`

	A	B	C
1	Eingabe	Ausgabe	Formel in B
2	1E-14	1:100000000000000	=WENNFEHLER(--A2*10^(LÄNGE(--A2)-SUCHEN(",";--A2))&":"&10^(LÄNGE(--A2)-SUCHEN(",";--A2));--A2&":1")
3	0,00001	1:100000	=WENNFEHLER(--A3*10^(LÄNGE(--A3)-SUCHEN(",";--A3))&":"&10^(LÄNGE(--A3)-SUCHEN(",";--A3));--A3&":1")
4	0,1	1:10	=WENNFEHLER(--A4*10^(LÄNGE(--A4)-SUCHEN(",";--A4))&":"&10^(LÄNGE(--A4)-SUCHEN(",";--A4));--A4&":1")
5	0,2	2:10	=WENNFEHLER(--A5*10^(LÄNGE(--A5)-SUCHEN(",";--A5))&":"&10^(LÄNGE(--A5)-SUCHEN(",";--A5));--A5&":1")
6	0,22	22:100	=WENNFEHLER(--A6*10^(LÄNGE(--A6)-SUCHEN(",";--A6))&":"&10^(LÄNGE(--A6)-SUCHEN(",";--A6));--A6&":1")
7	0,0001234	1234:10000000	=WENNFEHLER(--A7*10^(LÄNGE(--A7)-SUCHEN(",";--A7))&":"&10^(LÄNGE(--A7)-SUCHEN(",";--A7));--A7&":1")
8	0	0:1	=WENNFEHLER(--A8*10^(LÄNGE(--A8)-SUCHEN(",";--A8))&":"&10^(LÄNGE(--A8)-SUCHEN(",";--A8));--A8&":1")
9	1	1:1	=WENNFEHLER(--A9*10^(LÄNGE(--A9)-SUCHEN(",";--A9))&":"&10^(LÄNGE(--A9)-SUCHEN(",";--A9));--A9&":1")
10	10	10:1	=WENNFEHLER(--A10*10^(LÄNGE(--A10)-SUCHEN(",";--A10))&":"&10^(LÄNGE(--A10)-SUCHEN(",";--A10));--A10&":1")
11	100	100:1	=WENNFEHLER(--A11*10^(LÄNGE(--A11)-SUCHEN(",";--A11))&":"&10^(LÄNGE(--A11)-SUCHEN(",";--A11));--A11&":1")
12	3,141592654	314159265358979:100000000000000	=WENNFEHLER(--A12*10^(LÄNGE(--A12)-SUCHEN(",";--A12))&":"&10^(LÄNGE(--A12)-SUCHEN(",";--A12));--A12&":1")
13	1,1	11:10	=WENNFEHLER(--A13*10^(LÄNGE(--A13)-SUCHEN(",";--A13))&":"&10^(LÄNGE(--A13)-SUCHEN(",";--A13));--A13&":1")
14	12,12	1212:100	=WENNFEHLER(--A14*10^(LÄNGE(--A14)-SUCHEN(",";--A14))&":"&10^(LÄNGE(--A14)-SUCHEN(",";--A14));--A14&":1")
15	123,123	123123:1000	=WENNFEHLER(--A15*10^(LÄNGE(--A15)-SUCHEN(",";--A15))&":"&10^(LÄNGE(--A15)-SUCHEN(",";--A15));--A15&":1")
16	1E+200	1E+200:1	=WENNFEHLER(--A16*10^(LÄNGE(--A16)-SUCHEN(",";--A16))&":"&10^(LÄNGE(--A16)-SUCHEN(",";--A16));--A16&":1")
17	1E-200	1E-200:1	=WENNFEHLER(--A17*10^(LÄNGE(--A17)-SUCHEN(",";--A17))&":"&10^(LÄNGE(--A17)-SUCHEN(",";--A17));--A17&":1")
18	-0,1	-1:10	=WENNFEHLER(--A18*10^(LÄNGE(--A18)-SUCHEN(",";--A18))&":"&10^(LÄNGE(--A18)-SUCHEN(",";--A18));--A18&":1")
19	-3,141592654	-314159265358979:100000000000000	=WENNFEHLER(--A19*10^(LÄNGE(--A19)-SUCHEN(",";--A19))&":"&10^(LÄNGE(--A19)-SUCHEN(",";--A19));--A19&":1")
20	-123,123	-123123:1000	=WENNFEHLER(--A20*10^(LÄNGE(--A20)-SUCHEN(",";--A20))&":"&10^(LÄNGE(--A20)-SUCHEN(",";--A20));--A20&":1")
21	-12,12	-1212:100	=WENNFEHLER(--A21*10^(LÄNGE(--A21)-SUCHEN(",";--A21))&":"&10^(LÄNGE(--A21)-SUCHEN(",";--A21));--A21&":1")
22	-0,00004	-4:100000	=WENNFEHLER(--A22*10^(LÄNGE(--A22)-SUCHEN(",";--A22))&":"&10^(LÄNGE(--A22)-SUCHEN(",";--A22));--A22&":1")

sbNRN Code

```
#If Win64 Then
Function sbNRN(dFloat As Double, lMaxDen As LongLong, _
               Optional dMaxErr As Double = -1#) As Variant
#Else
Function sbNRN(dFloat As Double, lMaxDen As Long, _
               Optional dMaxErr As Double = -1#) As Variant
#End If
'Computes nearest rational number to dFloat with a maximal denominator
'lMaxDen and a maximal absolute error dMaxErr and returns result as a
'variant Nominator / Denominator.
'See: Oliver Aberth, A method for exact computation with rational numbers,
'     JCAM, vol 4, no. 4, 1978
'Bernd Plumhoff V1.21 09-Oct-2020

Dim dB As Double
#If Win64 Then
Dim lA As LongLong, lSgn As LongLong
Dim lP1 As LongLong, lP2 As LongLong, lP3 As LongLong
Dim lQ1 As LongLong, lQ2 As LongLong, lQ3 As LongLong
#Else
Dim lA As Long, lSgn As Long
Dim lP1 As Long, lP2 As Long, lP3 As Long
Dim lQ1 As Long, lQ2 As Long, lQ3 As Long
#End If

If dMaxErr = -1# Then dMaxErr = 1# / (2# * CDbl(lMaxDen) ^ 2#)
lSgn = Sgn(dFloat): dB = Abs(dFloat)
lP1 = 0: lP2 = 1: lQ1 = 1: lQ2 = 0

Do While lMaxDen > lQ2
    lA = Int(dB)
    lP3 = lA * lP2 + lP1: lQ3 = lA * lQ2 + lQ1
#If Win64 Then
    If Abs(dB - CDbl(lA)) < 1# / CLngLng("9223372036854775807") Then
#Else
    If Abs(dB - CDbl(lA)) < 1# / 2147483647# Then
#End If
        Exit Do
    End If
    dB = 1# / (dB - CDbl(lA))
    lP1 = lP2: lP2 = lP3: lQ1 = lQ2: lQ2 = lQ3
Loop

If lQ3 > lMaxDen Then
    lQ3 = lQ2: lP3 = lP2
    If lQ2 > lMaxDen Then
        lQ3 = lQ1: lP3 = lP1
    End If
End If

'If absolute error exceeds 1/2Q^2 then Aberth's lemma p. 286 might not apply.
'But the user can override this and check the result himself.
If Abs(dFloat - lSgn * lP3 / lQ3) > dMaxErr Then
    sbNRN = CVErr(xlErrNum)
Else
    sbNRN = Array(lSgn * lP3, lQ3)
End If

End Function
```

Linear Equations with Rational Coeffizients

Linear equations of the form A * x = b with the non-singular quadratic matrix A and the result vector b have a unique solution because the determinant of A is not zero. If the coefficients of A and of b are rational numbers then the solution is also rational.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Exact rational solution of linear equations with rational coefficients											
2												
3	Dimension:		6		Create linear equations sample							
4												
5	Nonsingular sample matrix											
6												
7	5	-1	22	-4	19	19		0,542156587		12		
8	-26	4	0	17	-5	28		5,276069137		8		
9	-22	10	11	13	-1	29	x	-3,603579561	=	42		
10	21	-20	9	-14	22	14		-13,10958678		48		
11	-7	-25	7	-24	10	-12		-4,934419977		19		
12	-25	7	2	4	20	26		7,113666789		50		
13												
14	Matrix was created with 1 try. Determinant is 60786621.											
15												
16	5	-1	22	-4	19	19		523109	/	964867		12
17	-26	4	0	17	-5	28		5090705	/	964867		8
18	-22	10	11	13	-1	29	x	-3476975	/	964867	=	42
19	21	-20	9	-14	22	14		-37947023	/	2894601		48
20	-7	-25	7	-24	10	-12		-4761059	/	964867		19
21	-25	7	2	4	20	26		20591227	/	2894601		50
22												
23	Rational solution is accurate!											

... Relation to Power of 10 **Linear rational equations**

Sample Code

```
Sub Generate_linear_equations_and_solve()
'Calculates next rational numbers to the double-precision solution of
'given linear equations. Accuracy of rational solution is then reported.
'(C) (P) by Bernd Plumhoff 05-Jun-2024 PB V0.1

Dim bAccurate          As Boolean
Dim abserr             As Double
Dim d                  As Double
Dim det                As Double
Dim i                  As Long
Dim iter               As Long
Dim j                  As Long
Dim k                  As Long
Dim loc                As Long
Dim n                  As Long
Dim state As SystemState

With Application.WorksheetFunction
Set state = New SystemState
n = Range("Matrix_Dimension")
If n < 2 Or n > 52 Then '52 is max dimension of MInverse
  Call MsgBox("Dimension must be between 2 and 52!", vbOKOnly, "Error")
  Exit Sub
End If
loc = Range("Sample_Matrix").Row + 2
wsLEQ.Rows(loc & ":" & 1000000).Delete
ReDim m(1 To n, 1 To n) As Variant
det = 0#
iter = 0
Do While det = 0# And iter < 20
  iter = iter + 1
  For i = 1 To n
    For j = 1 To n
      m(i, j) = Int(Rnd * 10 * n) - 5 * n
    Next j
  Next i
  det = .MDeterm(m)
Loop
```

```
If det = 0# Then
  Call MsgBox("Determinant was still 0 after 20 tries. Check the algorithm, please.", vbOKOnly, "Error")
  Exit Sub
End If
Range(wsLEQ.Cells(loc, 1), wsLEQ.Cells(loc + n - 1, n)) = m
ReDim b(1 To n) As Variant
For i = 1 To n
  b(i) = Int(Rnd * 10 * n)
  wsLEQ.Cells(loc + i - 1, n + 2) = "X" & i
Next i
wsLEQ.Cells(loc + (n - 1) \ 2, n + 1) = "x"
Range(wsLEQ.Cells(loc, n + 4), wsLEQ.Cells(loc + n - 1, n + 4)) = .Transpose(b)
wsLEQ.Cells(loc + (n - 1) \ 2, n + 3) = "="

ReDim mInv(1 To n, 1 To n) As Variant
mInv = .MInverse(m)
ReDim X(1 To n) As Variant
X = .MMult(mInv, .Transpose(b))
Range(wsLEQ.Cells(loc, n + 2), wsLEQ.Cells(loc + n - 1, n + 2)) = X
wsLEQ.Rows(loc & ":" & loc + n - 1).HorizontalAlignment = xlCenter
wsLEQ.Cells(loc + n + 1, 1) = "Matrix was created with " & iter & IIf(iter = 1, " try", " tries") & _
  ". Determinant is " & det & "."
'Just a check whether the design was ok. Commented out after successful check:
'ReDim bCheck(1 To n) As Variant
'bCheck = .MMult(m, x) 'For n=7 we could also manually have entered into cell M7: =MMULT(A7:G13;I7:I13)
'Range(wsLEQ.Cells(loc, n + 5), wsLEQ.Cells(loc + n - 1, n + 5)) = bCheck

ReDim xR(1 To n) As Variant
ReDim xCheck(1 To n, 1 To 1) As Variant
For i = 1 To n
  d = X(i, 1)
  xR(i) = sbNRN(d, 100000000#, 0.00000001)
  xCheck(i, 1) = xR(i)(0) / xR(i)(1)
Next i
'Now show the rational solution and whether it is accurate.
ReDim bCheck(1 To n) As Variant
bCheck = .MMult(m, xCheck)

Range(wsLEQ.Cells(loc + n + 3, 1), wsLEQ.Cells(loc + 2 * n + 2, n)) = m
wsLEQ.Cells(loc + n + 3 + (n - 1) \ 2, n + 1) = "x"
abserr = 0#
For i = 1 To n
  wsLEQ.Cells(loc + i + n + 2, n + 2) = xR(i)(0)
  wsLEQ.Cells(loc + i + n + 2, n + 3) = "/"
  wsLEQ.Cells(loc + i + n + 2, n + 4) = xR(i)(1)
  abserr = abserr + Abs(X(i, 1) - xR(i)(0) / xR(i)(1))
Next i
wsLEQ.Cells(loc + n + 3 + (n - 1) \ 2, n + 5) = "="
Range(wsLEQ.Cells(loc + n + 3, n + 6), wsLEQ.Cells(loc + 2 * n + 2, n + 6)) = bCheck
wsLEQ.Rows(loc + n + 3 & ":" & loc + 2 * n + 2).HorizontalAlignment = xlCenter
wsLEQ.Cells(loc + 2 * n + 4, 1) = "Rational solution is " & _
  IIf(abserr < 0.000000000001, "fairly accurate", "off by " & abserr) & "."
Range(wsLEQ.Cells(1, 1), wsLEQ.Cells(1, n)).EntireColumn.ColumnWidth = 5
Range(wsLEQ.Cells(1, n + 1), wsLEQ.Cells(1, n + 6)).EntireColumn.AutoFit
If n < 10 Then
  bAccurate = True
  ReDim b2(1 To n) As String
  Dim rT As String
  ReDim r2(1 To n) As String
  For i = 1 To n
    b2(i) = -b(i)
    r2(i) = "0"
    For j = 1 To n
      rT = xR(j)(0)
      b2(i) = sbMult(b2(i), xR(j)(1))
      'Debug.Print i, j, "rT = " & rT, "b2(" & i & ") = " & b2(i)
      For k = 1 To n
        If j <> k Then
          rT = sbMult(rT, xR(k)(1))
          'Debug.Print i, j, k, "rT = " & rT
        End If
      Next k
      r2(i) = sbPlus(r2(i), sbMult(m(i, j), rT))
      'Debug.Print i, j, "r2(" & i & ") = " & r2(i)
    Next j
    r2(i) = sbPlus(r2(i), b2(i))
    'Debug.Print i, "r2(" & i & ") = " & r2(i)
    If r2(i) <> "0" Then bAccurate = False
  Next
  If bAccurate Then wsLEQ.Cells(loc + 2 * n + 4, 1) = "Rational solution is accurate!"
End If

End With
End Sub

Function sbMult(ByVal a As String, ByVal b As String) As String
'Multiplication of big integers.
'(C) (P) by Bernd Plumhoff 05-Jun-2024 PB V0.1
Dim c As String, s As String, r As String
Dim i As Long, j As Long, k As Long, carry As Long, m As Long
```

```
With Application.WorksheetFunction
If Left(a, 1) = "-" And Left(b, 1) = "-" Then
  a = Right(a, Len(a) - 1)
  b = Right(b, Len(b) - 1)
End If
If Left(a, 1) = "-" Then
  s = "-"
  a = Right(a, Len(a) - 1)
End If
If Left(b, 1) = "-" Then
  s = "-"
  b = Right(b, Len(b) - 1)
End If
j = Len(a) + Len(b) + 1
a = Right(String(j, "0") & a, j)
b = Right(String(j, "0") & b, j)
r = "0"
carry = 0
For i = j To 1 Step -1
  c = String(j - i, "0")
  For m = j To 1 Step -1
    k = CLng(Mid(a, i, 1)) * CLng(Mid(b, m, 1)) + carry
    c = CStr(k Mod 10) & c
    carry = k \ 10
  Next m
  r = sbPlus(r, c)
Next i
For i = 1 To j - 1
  If Mid(r, i, 1) <> "0" Then Exit For
Next i
If i <= j Then r = Right(r, j - i + 1) '
sbMult = IIf(r = "0", "0", s & r)
End With
End Function

Function sbPlus(ByVal a As String, ByVal b As String) As String
'Addition of big integers.
'(C) (P) by Bernd Plumhoff 05-Jun-2024 PB V0.1
Dim bNega As Boolean, bNegb As Boolean
Dim c As String, s As String
Dim i As Long, j As Long, k As Long, carry As Long, negcarry As Long
With Application.WorksheetFunction
bNega = False
bNegb = False
If Left(a, 1) = "-" And Left(b, 1) = "-" Then
  s = "-"
  a = Right(a, Len(a) - 1)
  b = Right(b, Len(b) - 1)
End If
If Left(a, 1) = "-" Then
  bNega = True
  a = Right(a, Len(a) - 1)
  If Len(b) < Len(a) Or (Len(b) = Len(a) And b < a) Then
    bNega = False
    bNegb = True
    s = "-"
  End If
End If
If Left(b, 1) = "-" Then
  bNegb = True
  b = Right(b, Len(b) - 1)
  If Len(b) > Len(a) Or (Len(b) = Len(a) And b > a) Then
    bNega = True
    bNegb = False
    s = "-"
  End If
End If
j = .Max(Len(a), Len(b)) + 1
a = Right(String(j, "0") & a, j)
b = Right(String(j, "0") & b, j)
c = ""
carry = 0
For i = j To 1 Step -1
  k = IIf(bNega, 10 - CLng(Mid(a, i, 1)), CLng(Mid(a, i, 1))) + _
      IIf(bNegb, 10 - CLng(Mid(b, i, 1)), CLng(Mid(b, i, 1))) + _
      carry + negcarry
  c = CStr(k Mod 10) & c
  carry = k \ 10
  negcarry = bNega + bNegb
Next i
For i = 1 To j - 1
  If Mid(c, i, 1) <> "0" Then Exit For
Next i
If i <= j Then c = Right(c, j - i + 1)
sbPlus = IIf(c = "0", "0", s & c)
End With
End Function
```

Present Quota Changes as Fractions

Sometimes you need to present quota changes in a simple way. You can achieve this with fractions:

Example: Miller, Smith, and Schulz form a joint heirship. Smith dies without an heir. His quota will be distributed. Schulz also dies. His widows will receive 2/3 of his quota, his only child will get 1/3. Please note that you need to enter this as =1/3 * 2/3 resp. =1/3 * 1/3 whereby the first 1/3 represents Schulz' original quota.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Name	Quota old	As Fraction	Quota old normed	As Fraction	Quota new	As Fraction	Quota new normed	As Fraction	Largest Denominator	Same Denominator	Change	As Fraction	Largest Denominator	Same Denominator
2	*Total*	*100,00%*	*1*	*100,00%*	*1*	*66,67%*	*2/3*	*100,00%*	*1*	*6*	*6/6*			*6*	
3	Miller	33,33%	1/3	33,33%	1/3	33,33%	1/3	50,00%	1/2	2	3/6	16,67%	1/6	6	1/6
4	Smith	33,33%	1/3	33,33%	1/3							-33,33%	- 1/3	3	-2/6
5	Schulz	33,33%	1/3	33,33%	1/3							-33,33%	- 1/3	3	-2/6
6	Schulz Widow					22,22%	2/9	33,33%	1/3	3	2/6	33,33%	1/3	3	2/6
7	Schulz Child					11,11%	1/9	16,67%	1/6	6	1/6	16,67%	1/6	6	1/6
8															
9															
10															
11															
12															
13															
14	Note:	Enter values into these fields only!													

Formulas Used

You can solve this problem easily with Excel spreadsheet formulas by using Excel's internal representation of fractions as text:

Name	Quota old	As Fraction	Quota old normed	As Fraction	Quota new	As Fraction
Total	*=SUM(B3:B12)*	*=IF(B2:B12=0,"",TEXT(B2:B12,"# ???/???"))*	*=SUM(D3:D12)*	*=IF(D2:D12=0,"",TEXT(D2:D12,"# ???/???"))*	*=SUM(F3:F12)*	*=IF(F2:F12=0,"",TEXT(F2:F12,"# ???/???"))*
Miller	=1/3		=B3:B12/B2		=1/3	

Quota new normed	As Fraction	Largest Denominator	Same Denominator
=SUM(H3:H12)	*=IF(H2:H12=0,"",TEXT(H2:H12,"# ???/???"))*	*=MAX(J3:J12)*	*=IF(H2:H12<>0,ROUND(H2:H12*J2,0) & "/" & J2,"")*
=F3:F12/F2		=VALUE("0" & RIGHT(I3:I12,LEN(I3:I12)-IFERROR(SEARCH("/",I3:I12),0)))	

Change	As Fraction	Largest Denominator	Same Denominator
=H2:H12-D2:D12	*=IF(L2:L12=0,"",TEXT(L2:L12,"# ???/???"))*	*=MAX(N3:N12)*	*=IF(L2:L12<>0,ROUND(L2:L12*N2,0) & "/" & N2,"")*
		=VALUE("0" & RIGHT(M3:M12,LEN(M3:M12)-IFERROR(SEARCH("/",M3:M12),0)))	

But you could also use the user-defined function *sbNRN* instead.

17 Camels Trick

A father left 17 camels to his three sons and, according to the will, the eldest son should be given a half of all camels, the middle son the one-third part and the youngest son the one-ninth. This is hard to do, but a wise man helped the sons: he added his own camel, the oldest son took 18/2 = 9 camels, the second son took 18/3 = 6 camels, the third son 18/9 = 2 camels and the wise man took his own camel and went away.

Now we can see easily what's wrong with this will:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Name	Quota old	As Fraction	Quota old normed	As Fraction	Quota new	As Fraction	Quota new normed	As Fraction	Largest Denominator	Same Denominator	Change	As Fraction	Largest Denominator	Same Denominator
2	*Total*	*94,44%*	*17/18*	*100,00%*	*1*	*100,00%*	*1*	*100,00%*	*1*	*18*	*18/18*			*153*	
3	Eldest son	50,00%	1/2	52,94%	9/17	50,00%	1/2	50,00%	1/2	2	9/18	-2,94%	- 1/34	34	-5/153
4	Second son	33,33%	1/3	35,29%	6/17	33,33%	1/3	33,33%	1/3	3	6/18	-1,96%	- 1/51	51	-3/153
5	Youngest son	11,11%	1/9	11,76%	2/17	11,11%	1/9	11,11%	1/9	9	2/18	-0,65%	- 1/153	153	-1/153
6	Wise man					5,56%	1/18	5,56%	1/18	18	1/18	5,56%	1/18	18	9/153

The quotas do not add up to 100% but only to 94.44%. The sons' shares are too low by 1/34, 1/51, and 1/153. The father should have given his youngest son one-sixth of all camels, then all parts would have added up to 100%. The parts would not have resulted in whole camels but the sons could have agreed to compensation payments.

Monthly Fractions

If you need the number of full months or the total count of days between two days you can use *DATEDIF*. Easy so far. But what if you need the number of months in fractions?

This way it is done correctly:

```
=DATEDIF(A2,B2,"m")+IF(DAY(B2)>=DAY(A2),(DAY(B2)-DAY(A2))/(DAY(DATE(YEAR(B2),MONTH(B2)+1,0))),(DAY(DATE(YEAR(A2),MONTH(A2)+1,0))-DAY(A2))/(DAY(DATE(YEAR(A2),MONTH(A2)+1,0)))+DAY(B2)/(DAY(DATE(YEAR(B2),MONTH(B2)+1,0))))
```

	A	B	C	D	E
1	Start Date	End Date	Difference in months	...in days	Comment
2	01-Jan-2009	01-Feb-2009	1	31	One full calendar month
3	01-Jan-2009	16-Jan-2009	0.483870968	15	15 days in January 2009
4	16-Jan-2009	01-Feb-2009	0.519585253	16	15 days in January 2009 plus 1 day in February 2009
5	28-Feb-2000	28-Mar-2000	1	29	One full calendar month

Note: If you use the *EOMONTH* function you can abbreviate the formula to

```
=DATEDIF(A2,B2,"m")+IF(DAY(B2)>=DAY(A2),(DAY(B2)-DAY(A2))/DAY(EOMONTH(B2,0)),(DAY(EOMONTH(A2,0))-DAY(A2))/DAY(EOMONTH(A2,0))+DAY(B2)/DAY(EOMONTH(B2,0)))
```

If you like to represent this result as a rational number I suggest to use sbNRN.

Linear Combination of Integers

Extended Euklidean Algorithm – *sbEuklid*

You need to represent a number as a non-negative linear combination of two positive integers?

You can achieve this with the extended Euklidean Algorithm:

B9 =sbEuklid(B3;C3;B4;B5)

	A	B	C
1	**Extended Euklidian Algorithm**		
2			
3	**Input**	177	131
4	**Desired Result**	19.191.919	
5	**All numbers non-negative**	WAHR	
6			
7	**Greatest Common Divisor**	1	
8			
9	**Result**	86	146.387
10			
11	**This means:**	19191919 = 86 * 177 + 146387 * 131	

Note: If your desired result is a multiple of the greatest common divisor of the inputs then there will always be an integer solution, but not necessarily a non-negative one.

Example: You can represent 1 with the inputs 5 and 3 because 1 is the GCD of 3 and 5: 1 = 2 * 5 + (-3) * 3. But you cannot achieve this with non-negative integers only.

Another note: There might be more than one non-negative integer solution, with bAllNonNegative = True the algorithm below will return a minimal sum of the output values. When there is an integer solution then there is a countable number of solutions.

sbEuklid Code

Possible error returns of the program are:

- #NV! - There is no solution
- #VALUE! - *bAllNonNegative* = True but not all inputs are non-negative
- #NUM! - *bAllNonNegative* = True but there is no non-negative integer solution

```vba
Function sbEuklid(lInput1 As Long, _
                  lInput2 As Long, _
                  Optional lDesiredResult As Long, _
                  Optional bAllNonNegative As Boolean = False) As Variant
'Extended Euklidean Algorithm which calculates two factors f1 and f2
'so that lDesiredResult = f1 * lInput1 + f2 * lInput2. If lDesiredResult
'is not given, the greatest common divisor (GCD) of lInput1 and lInput2
'will be calculated. If bAllNonNegative is True then we try to achieve a
'non-negative result of all inputs and outputs with minimal Sum(f1+f2).
'Error return values can be:
'xlErrNA    - There is no solution
'xlErrValue - bAllNonNegative = True but not all inputs are non-negative
'xlErrNum   - bAllNonNegative = True but there is no non-negative solution
'(C) (P) by Bernd Plumhoff 20-May-2024 PB V0.4
Dim lDiv                    As Long
Dim lGCD                    As Long
Dim lRest                   As Long
Dim lT1                     As Long
Dim lT2                     As Long
Dim vR                      As Variant
Dim vT                      As Variant

With Application '.WorksheetFunction 'Test with, release without
  lGCD = .Gcd(lInput1, lInput2)
  If IsMissing(lDesiredResult) Then lDesiredResult = lGCD
  lRest = lDesiredResult Mod lGCD
  If lRest <> 0 Then
    sbEuklid = CVErr(xlErrNA) 'There is no solution
  Else
    If bAllNonNegative And (lInput1 < 0 Or lInput2 < 0 Or lDesiredResult < 0) Then
      sbEuklid = CVErr(xlErrValue) 'bAllNonNegative but not all inputs are non-negative
    Else
      'See https://www.arndt-bruenner.de/mathe/scripts/erweitertereuklid.htm
      vR = [{1, 0; 0, 1}]
      vT = [{0, 1; 1, 0}]
      lT1 = lInput1
      lT2 = lInput2
      Do
        lDiv = lT1 \ lT2
        lRest = lT1 Mod lT2
        lT1 = lT2
        lT2 = lRest
        vT(2, 2) = -lDiv
        vR = .MMult(vR, vT)
      Loop While lRest <> 0
      vR = .MMult(Array(lDesiredResult \ lGCD, 0), .Transpose(vR))
      Debug.Assert lDesiredResult = vR(1) * lInput1 + vR(2) * lInput2 'Just assuring
      sbEuklid = vR
      If bAllNonNegative Then
        If lInput1 > lInput2 Then
          lT1 = lDesiredResult \ lInput1 + 1
          Do While lT1 > 0
            lT1 = lT1 - 1
            If (lDesiredResult - lInput1 * lT1) Mod lInput2 = 0 Then GoTo Success1
          Loop
          GoTo ErrorExit
Success1: vR(1) = lT1
          vR(2) = (lDesiredResult - lInput1 * lT1) \ lInput2
        Else
          lT2 = lDesiredResult \ lInput2 + 1
          Do While lT2 > 0
            lT2 = lT2 - 1
            If (lDesiredResult - lInput2 * lT2) Mod lInput1 = 0 Then GoTo Success2
          Loop
          GoTo ErrorExit
Success2: vR(2) = lT2
          vR(1) = (lDesiredResult - lInput2 * lT2) \ lInput1
        End If
        sbEuklid = vR
      End If
    End If
  End If
  'Debug.Assert lDesiredResult = vR(1) * lInput1 + vR(2) * lInput2 'Just testing
Exit Function
ErrorExit:
  sbEuklid = CVErr(xlErrNum)
End With

End Function
```

Time Representations

Calculate Working Hours Between Two Time Points – *sbTimeDiff*

Name

sbTimeDiff() - Calculate time between two time points but count only time as specified for week days and for holidays subtracted by break times if working time exceeds specified time.

Synopsis

sbTimeDiff(*dtFrom*, *dtTo*, *vwh* [, *vHolidays*] [, *vBreaks*])

Description

Calculate time between two time points but count only time as specified for week days and for holidays subtracted by break times if given for specified working time.

Parameters

dtFrom - Datetime to count from

dtTo - Datetime to count to

vwh – 8 by 2 matrix defining start time and end time for each weekday and for holidays, first row for Mondays, 8th row for holidays

vHolidays - Optional. List of holidays (integer datetime). If a day is in the holiday list its time will not be counted for any weekday - just for the time defined in row 8 of parameter *vwh*

vBreaks - Optional. N x 2 matrix specifying working time (sorted in ascending order) and break time to subtract if corresponding time for a day has been worked

Example

C9 =sbTimeDiff(A9,B9,$B$12:$C$19,,$B$22:$C$23)

	A	B	C	D	E	F	G
1		sbTimeDiff Examples					UK Holidays
2	From	To	Result	Formula	Comment		Fri 19-Apr-2019
3	Tue 24-Dec-2019 16:00:00	Sun 29-Dec-2019 10:00:00	34:00:00	=sbTimeDiff(A3,B3,$B$12:$C$19)	No holidays		Mon 22-Apr-2019
4	Tue 24-Dec-2019 16:00:00	Sun 29-Dec-2019 10:00:00	18:00:00	=sbTimeDiff(A4,B4,$B$12:$C$19,$G$2:$G$128)	With holidays		Mon 06-May-2019
5	Sun 05-Apr-2020 12:59:00	Sun 05-Apr-2020 19:32:00	0:01:00	=sbTimeDiff(A5,B5,$B$12:$C$19)	No holidays		Mon 27-May-2019
6	Sun 05-Apr-2020 11:30:00	Sun 05-Apr-2020 16:30:00	1:30:00	=sbTimeDiff(A6,B6,$B$12:$C$19,$G$2:$G$128)	With holidays		Mon 26-Aug-2019
7	Sun 05-Apr-2020 06:29:00	Mon 06-Apr-2020 08:32:00	2:32:00	=sbTimeDiff(A7,B7,$B$12:$C$19)	No holidays		Wed 25-Dec-2019
8	Tue 24-Dec-2019 16:00:00	Sun 29-Dec-2019 10:00:00	18:00:00	=sbTimeDiff(A8,B8,$B$12:$C$19,$G$2:$G$128)	With holidays		Thu 26-Dec-2019
9	Tue 15-Sep-2020 05:30:00	Fri 18-Sep-2020 00:31:00	27:45:00	=sbTimeDiff(A9,B9,$B$12:$C$19,,$B$22:$C$23)	With breaks, no hols		Wed 01-Jan-2020
10							Fri 10-Apr-2020
11	Working Hours	Start	End				Mon 13-Apr-2020
12	Monday	8:00	18:00				Mon 04-May-2020
13	Tuesday	8:00	18:00				Mon 25-May-2020
14	Wednesday	8:00	18:00				Mon 31-Aug-2020
15	Thursday	8:00	18:00				Fri 25-Dec-2020
16	Friday	8:00	18:00				Mon 28-Dec-2020
17	Saturday	10:00	12:00				Fri 01-Jan-2021
18	Sunday	11:00	13:00				Fri 02-Apr-2021
19	Holidays	12:00	14:00				Mon 05-Apr-2021
20							Mon 03-May-2021
21	Breaks	Limit	Duration				Mon 31-May-2021
22	First	06:00	00:30				Mon 30-Aug-2021
23	Second	09:00	00:15				Mon 27-Dec-2021

sbTimeDiff Code

```
Enum mc_Macro_Categories
    mcFinancial = 1
    mcDate_and_Time
    mcMath_and_Trig
    mcStatistical
    mcLookup_and_Reference
    mcDatabase
    mcText
    mcLogical
    mcInformation
    mcCommands
    mcCustomizing
    mcMacro_Control
    mcDDE_External
    mcUser_Defined
    mcFirst_custom_category
    mcSecond_custom_category 'and so on
End Enum 'mc_Macro_Categories

Function sbTimeDiff(dtFrom As Date, dtTo As Date, _
    vwh As Variant, _
    Optional vHolidays As Variant, _
    Optional vBreaks As Variant) As Date
'Returns time between dtFrom and dtTo but counts only
'dates and hours given in table vwh: for example
'09:00   17:00  'Monday
'09:00   17:00  'Tuesday
'09:00   17:00  'Wednesday
'09:00   17:00  'Thursday
'09:00   17:00  'Friday
'00:00   00:00  'Saturday
'00:00   00:00  'Sunday
'00:00   00:00  'Holidays
'This table defines hours to count for each day of the
'week (starting with Monday, 2 columns) and for holidays.
'Holidays given in vHolidays overrule week days.
'If you define a break table with break limits greater zero
'then the duration of each break exceeding the applicable
'time for this day will be subtracted from each day's time,
'but only down to the limit time, table needs to be sorted
'by limits in increasing order:
'Break table example
'Limit Duration (title row is not part of the table)
'6:00  0:30
'9:00  0:15
'
'(C) (P) by Bernd Plumhoff 28-Aug-2020 PB V1.3
Dim dt2 As Date, dt3 As Date, dt4 As Date, dt5 As Date
```

```
Dim i As Long, lTo As Long, lFrom As Long
Dim lWDFrom As Long, lWDTo As Long, lWDi As Long
Dim objHolidays As Object, objBreaks As Object, v As Variant

With Application.WorksheetFunction
sbTimeDiff = 0#
If dtTo <= dtFrom Then Exit Function
Set objHolidays = CreateObject("Scripting.Dictionary")
If Not IsMissing(vHolidays) Then
    For Each v In vHolidays
        objHolidays(v.Value) = 1
    Next v
End If
If Not IsMissing(vBreaks) Then
    vBreaks = .Transpose(.Transpose(vBreaks))
    Set objBreaks = CreateObject("Scripting.Dictionary")
    For i = LBound(vBreaks, 1) To UBound(vBreaks, 1)
        objBreaks(CDate(vBreaks(i, 1))) = CDate(vBreaks(i, 2))
    Next i
End If
lFrom = Int(dtFrom): lWDFrom = Weekday(lFrom, vbMonday)
lTo = Int(dtTo): lWDTo = Weekday(lTo, vbMonday)
If lFrom = lTo Then
    lWDi = lWDTo: If objHolidays(lTo) Then lWDi = 8
    dt3 = lTo + CDate(vwh(lWDi, 2))
    If dt3 > dtTo Then dt3 = dtTo
    dt2 = lTo + CDate(vwh(lWDi, 1))
    If dt2 < dtFrom Then dt2 = dtFrom
    If dt3 > dt2 Then
        dt2 = dt3 - dt2
    Else
        dt2 = 0#
    End If
    If Not IsMissing(vBreaks) Then
        dt2 = sbBreaks(dt2, objBreaks)
    End If
    sbTimeDiff = dt2
    Set objHolidays = Nothing
    Set objBreaks = Nothing
    Exit Function
End If
lWDi = lWDFrom: If objHolidays(lFrom) Then lWDi = 8
If dtFrom - lFrom >= CDate(vwh(lWDi, 2)) Then
    dt2 = 0#
Else
    dt2 = lFrom + CDate(vwh(lWDi, 1))
    If dt2 < dtFrom Then dt2 = dtFrom
    dt2 = lFrom + CDate(vwh(lWDi, 2)) - dt2
    If Not IsMissing(vBreaks) Then
        dt2 = sbBreaks(dt2, objBreaks)
    End If
End If
lWDi = lWDTo: If objHolidays(lTo) Then lWDi = 8
If dtTo - lTo <= CDate(vwh(lWDi, 1)) Then
    dt4 = 0#
Else
    dt4 = lTo + CDate(vwh(lWDi, 2))
    If dt4 > dtTo Then dt4 = dtTo
    dt4 = dt4 - lTo - CDate(vwh(lWDi, 1))
    If Not IsMissing(vBreaks) Then
        dt4 = sbBreaks(dt4, objBreaks)
    End If
End If
dt3 = 0#
For i = lFrom + 1 To lTo - 1
    lWDi = Weekday(i, vbMonday)
    If objHolidays(i) Then lWDi = 8
    dt5 = CDate(vwh(lWDi, 2)) - CDate(vwh(lWDi, 1))
    If Not IsMissing(vBreaks) Then
        dt5 = sbBreaks(dt5, objBreaks)
    End If
    dt3 = dt3 + dt5
Next i
Set objHolidays = Nothing
Set objBreaks = Nothing
sbTimeDiff = dt2 + dt3 + dt4
End With
End Function

Private Function sbBreaks(ByVal dt As Date, objBreaks As Object) As Date
'Subtract break durations from dt as long as it exceeds the break limit,
'but not below break limit.
'(C) (P) by Bernd Plumhoff 22-Mar-2020 PB V1.00
Dim dtTemp As Date
Dim k As Long
k = 0
Do While k <= UBound(objBreaks.keys)
    If dt > objBreaks.keys()(k) + objBreaks.items()(k) - dtTemp Then
        dt = dt - objBreaks.items()(k)
        dtTemp = dtTemp + objBreaks.items()(k)
```

```
    ElseIf dt > objBreaks.keys()(k) - dtTemp Then
        dt = objBreaks.keys()(k) - dtTemp
        Exit Do
    End If
    k = k + 1
Loop
sbBreaks = dt
End Function

Sub DescribeFunction_sbTimeDiff()

'Run this only once, then you will see this description in the function menu

Dim FuncName As String
Dim FuncDesc As String
Dim Category As String
Dim ArgDesc(1 To 5) As String

FuncName = "sbTimeDiff"
FuncDesc = "Returns time between dtFrom and dtTo but counts only " & _
            "time given in table vwh. Holidays given in vHolidays " & _
            "overrule week days, all breaks given in vBreaks are " & _
            "subtracted if corresponding time has been worked"
Category = mcDate_and_Time
ArgDesc(1) = "Start date and time where to count from"
ArgDesc(2) = "End date and time to count to"
ArgDesc(3) = "Range or array which defines which time to count during the week starting from Monday, " & _
            "8 by 2 matrix defining start time and end time for each weekday (8th row for holidays)"
ArgDesc(4) = "Optional list of holidays which overrule week days, define time to count in 8th row of vwh"
ArgDesc(5) = "Optional. N x 2 matrix specifying working limit times (sorted in ascending order) and break"
& _
             " durations to subtract if corresponding time for a day has been worked (but not below limit
time)"

Application.MacroOptions _
    Macro:=FuncName, _
    Description:=FuncDesc, _
    Category:=Category, _
    ArgumentDescriptions:=ArgDesc

End Sub
```

Add Working Hours to a Time Point – *sbTimeAdd*

Name

sbTimeAdd() - Add positive hours to a timepoint but count only time as specified for week days and for holidays increased by break time if working time exceeds specified time.

Synopsis

sbTimeAdd(*dt*, *dh*, *vwh* [, *vHolidays*] [, *dtBreakLimit*] [, *dtBreakDuration*])

Description

Add positive hours to a timepoint but count only time as specified for week days and for holidays increased by break time if working time exceeds specified time.

Parameters

dt - Datetime to add hours to

dh - Hours (plus minutes) of type Double to add to *dt*

vwh - 8 by 2 matrix defining start time and end time for each weekday and for holidays, first row for Mondays, 8th row for holidays

vHolidays - Optional. List of holidays (integer datetime). If a day is in the holiday list its time will not be counted for any weekday - just for the time defined in row 8 of parameter *vwh*

dtBreakLimit - Optional. Daily working time, if reached then *dtBreakDuration* will be subtracted for that day

dtBreakDuration - Optional. Break time. Will be subtracted from total time if daily working time exceeds *dtBreakLimit*

<u>Example</u>

C3 =sbTimeAdd(A3,B3,$B$13:$C$20)

	A	B	C	D	E	F	G
1			sbTimeAdd Examples				UK Holidays
2	Date	+ Hours	Result	Formula	Comment		Fri 19-Apr-2019
3	Tue 24-Dec-2019 16:00:00	18:00	Thu 26-Dec-2019 14:00:00	=sbTimeAdd(A3,B3,$B$13:$C$20)	No holidays		Mon 22-Apr-2019
4	Tue 24-Dec-2019 16:00:00	18:00	Sat 28-Dec-2019 12:00:00	=sbTimeAdd(A4,B4,$B$13:$C$20,$G$2:$G$128)	With holidays		Mon 06-May-2019
5	Fri 10-Apr-2020 11:30:00	0:00	Fri 10-Apr-2020 11:30:00	=sbTimeAdd(A5,B5,$B$13:$C$20)	No holidays		Mon 27-May-2019
6	Fri 10-Apr-2020 11:30:00	0:00	Fri 10-Apr-2020 12:00:00	=sbTimeAdd(A6,B6,$B$13:$C$20,$G$2:$G$128)	With holidays		Mon 26-Aug-2019
7	Thu 09-Apr-2020 18:00:00	16:00	Mon 13-Apr-2020 10:00:00	=sbTimeAdd(A7,B7,$B$13:$C$20)	No holidays		Wed 25-Dec-2019
8	Thu 09-Apr-2020 18:00:00	16:00	Tue 14-Apr-2020 16:00:00	=sbTimeAdd(A8,B8,$B$13:$C$20,$G$2:$G$128)	With holidays		Thu 26-Dec-2019
9	Tue 02-Jun-2020 10:30:00	12:00	Wed 03-Jun-2020 12:30:00	=sbTimeAdd(A9,B9,$B$13:$C$20)	No break		Wed 01-Jan-2020
10	Tue 02-Jun-2020 10:30:00	12:00	Wed 03-Jun-2020 13:00:00	=sbTimeAdd(A10,B10,$B$13:$C$20,,$B$23,$C$23)	With break		Fri 10-Apr-2020
11							Mon 13-Apr-2020
12	Working Hours	Start	End				Mon 04-May-2020
13	Monday	8:00	18:00				Mon 25-May-2020
14	Tuesday	8:00	18:00				Mon 31-Aug-2020
15	Wednesday	8:00	18:00				Fri 25-Dec-2020
16	Thursday	8:00	18:00				Mon 28-Dec-2020
17	Friday	8:00	18:00				Fri 01-Jan-2021
18	Saturday	10:00	12:00				Fri 02-Apr-2021
19	Sunday	11:00	13:00				Mon 05-Apr-2021
20	Holidays	12:00	14:00				Mon 03-May-2021
21							Mon 31-May-2021
22		Limit	Duration				Mon 30-Aug-2021
23	Break	06:00	00:30				Mon 27-Dec-2021

sbTimeAdd Code

```
Enum mc_Macro_Categories
    mcFinancial = 1
    mcDate_and_Time
    mcMath_and_Trig
    mcStatistical
    mcLookup_and_Reference
    mcDatabase
    mcText
    mcLogical
    mcInformation
    mcCommands
    mcCustomizing
    mcMacro_Control
    mcDDE_External
    mcUser_Defined
    mcFirst_custom_category
    mcSecond_custom_category 'and so on
End Enum 'mc_Macro_Categories

Function sbTimeAdd(dt As Date, dh As Double, _
    vwh As Variant, _
    Optional vHolidays As Variant, _
    Optional dtBreakLimit As Date, _
    Optional dtBreakDuration As Date) As Date
'Returns end date from start date dt and positive duration
'dh in hours (and minutes and seconds) but counts only
'time as given in table vwh: for example
```

```
'09:00   17:00  'Monday
'09:00   17:00  'Tuesday
'09:00   17:00  'Wednesday
'09:00   17:00  'Thursday
'09:00   17:00  'Friday
'00:00   00:00  'Saturday
'00:00   00:00  'Sunday
'00:00   00:00  'Holidays
'This table defines hours to count for each day of the
'week (starting with Monday, 2 columns) and for holidays.
'You can also define a break limit and a break duration.
'If the working hour for a day is exceeding the limit
'then the duration will be subtracted from its time.
'(C) (P) by Bernd Plumhoff 02-Feb-2019 PB V0.7
Dim dt1 As Date, dt2 As Date
Dim ldt1 As Long, lWDi As Long, v As Variant
Dim objHolidays As Object, objBreaks As Object

If dh < 0# Then
    sbTimeAdd = CVErr(xlErrValue)
    Exit Function
End If
If Not IsMissing(vHolidays) Then
    Set objHolidays = CreateObject("Scripting.Dictionary")
    For Each v In vHolidays
        objHolidays(Int(v.Value)) = 1
    Next v
End If
ldt1 = Int(dt)
lWDi = Weekday(ldt1, vbMonday)
If Not IsMissing(vHolidays) Then
    If objHolidays(ldt1) Then
        lWDi = 8
    End If
End If
dt1 = ldt1 + CDate(vwh(lWDi, 1)) 'start time of this day
If dt1 < dt Then dt1 = dt
dt2 = ldt1 + CDate(vwh(lWDi, 2)) 'end time of this day
If dt2 < dt1 Then dt2 = dt1
Do While Round2Sec(dt1 + dh - (dh >= dtBreakLimit) * _
    dtBreakDuration) > Round2Sec(dt2)
    'go ahead as long as our duration exceeds this day
    If dt1 < ldt1 + CDate(vwh(lWDi, 2)) Then
        dh = dh - dt2 + dt1 - (dh >= dtBreakLimit) * dtBreakDuration
    End If
    ldt1 = ldt1 + 1
    lWDi = Weekday(ldt1, vbMonday)
    If Not IsMissing(vHolidays) Then
        If objHolidays(ldt1) Then
            lWDi = 8
        End If
    End If
    dt1 = ldt1 + CDate(vwh(lWDi, 1)) 'start time of this day
    dt2 = ldt1 + CDate(vwh(lWDi, 2)) 'end time of this day
Loop
sbTimeAdd = dt1 + dh - (dh >= dtBreakLimit) * dtBreakDuration
End Function

Function Round2Sec(dt As Date) As Date
Round2Sec = Int(0.5 + dt * 24 * 60 * 60) / 24 / 60 / 60
End Function

Sub DescribeFunction_sbTimeAdd()

'Run this only once, then you will see this description in the function menu

Dim FuncName As String
Dim FuncDesc As String
Dim Category As String
Dim ArgDesc(1 To 6) As String

FuncName = "sbTimeAdd"
FuncDesc = "Add positive hours to a timepoint but count only time as specified for week days" & _
           " and for holidays increased by break time if working time exceeds specified time"
Category = mcDate_and_Time
ArgDesc(1) = "Start date and time where to count from"
ArgDesc(2) = "Hours to add"
ArgDesc(3) = "Range or array which defines which time to count during the week starting from Monday, " & _
            "8 by 2 matrix defining start time and end time for each weekday (8th row for holidays)"
ArgDesc(4) = "Optional list of holidays which overrule week days, define time to count in 8th row of vwh"
ArgDesc(5) = "Optional. Daily working time limit. If exceeded dtBreakDUration will be subtracted from total
time"
ArgDesc(6) = "Optional. Break time. Will be subtracted from total time if daily working time exceeds
dtBreakLimit"

Application.MacroOptions _
    Macro:=FuncName, _
    Description:=FuncDesc, _
    Category:=Category, _
    ArgumentDescriptions:=ArgDesc
```

Convert a Time to a Different Time Zone – *ConvertTime*

Julian Hess and Patrick Honorez have suggested a very good time conversion program for MS Access as well as for MS Excel to convert a date and time from one time zone into an other one:

https://stackoverflow.com/questions/3120915/get-timezone-information-in-vba-excel/20489651#20489651

Please note that this solution requires MS Outlook to be installed properly.

Check Digits

Check digits are used to prevent manual input errors or transfer errors of numbers.

Calculate or Check a European Article Number – *sbEAN*

If you need to calculate or to check an international article number (also known as European article number or EAN):

	A	B	C
1	**Input**	**Output**	**Comment**
2	1234567	0	Just the EAN-8 check digit
3	1234567	12345670	Full EAN-8
4	12345670	TRUE	EAN-8 is correct
5	12345678	FALSE	EAN-8 is incorrect
6	123456789012	8	Just the EAN-13 check digit
7	123456789012	1234567890128	Full EAN-13
8	1234567890128	TRUE	EAN-13 is correct
9	1234567890129	FALSE	EAN-13 is incorrect
10	12345678901234567	5	Just the EAN-18 / NVE / SSCC check digit
11	12345678901234567	123456789012345675	Full EAN-18
12	123456789012345675	TRUE	EAN-18 is correct
13	123456789012345676	FALSE	EAN-18 is incorrect
14	9023671254823	0	Just the EAN-14 / GTIN check digit
15	9023671254823	90236712548230	Full EAN-14 / GTIN
16	90236712548230	TRUE	EAN-14 / GTIN is correct
17	90236712548231	FALSE	EAN-14 / GTIN is incorrect
18	1234567890123456789	#VALUE!	Wrong length of input
19	123456789O12	#NUM!	Not a number (character 'O')

sbEAN Code

```
Function sbEAN(s As String, _
  Optional bFullEAN As Boolean = True, _
  Optional bEAN14 As Boolean = False) As Variant
'Calculate or check EAN check digit. Works for EAN-8,
'EAN-13, EAN-14 / GTIN, and for EAN-18 / NVE / SSCC.
'If EAN is given without check digit, it is calculated
'and returned (full EAN if bFullEAn is True or just the
'check digit if False). If full EAN is entered the
'result of the check (True or False) will be returned.
'(C) (P) by Bernd Plumhoff 31-Mar-2024 PB V0.3
Dim i As Long, d As Long, m As Long, w As Long
Dim bCheck As Boolean
m = Len(s)
For i = 1 To m
  w = Asc(Mid(s, i, 1))
  If w < 48 Or w > 57 Then
    sbEAN = CVErr(xlErrNum)
    Exit Function
  End If
Next i
If bEAN14 Then
  If m = 13 Then
    bCheck = False
  ElseIf m = 14 Then
    bCheck = True
    m = m - 1 'Calculate checksum without check digit
  Else
    sbEAN = CVErr(xlErrValue)
    Exit Function
  End If
Else
  Select Case m
  Case 7, 12, 17
    bCheck = False
  Case 8, 13, 18
    bCheck = True
    m = m - 1 'Calculate checksum without check digit
  Case Else
    sbEAN = CVErr(xlErrValue)
    Exit Function
  End Select
End If
w = 3
For i = m To 1 Step -1
    d = d + Mid(s, i, 1) * w
    w = 4 - w 'Alternate between 3 and 1
Next i
d = (10 - d Mod 10) Mod 10
If bCheck Then
  sbEAN = Right(s, 1) = d
ElseIf bFullEAN Then
  sbEAN = s & d
Else
  sbEAN = d
End If
End Function
```

Ordinal Numbers

Excel has no built-in function for ordinal numbers such as 1st, 2nd, 3rd, 4th, … 100th.

Formula solutions:

```
=A1&MID("thstndrd",(LEFT(RIGHT("0"&A1,2))<>"1")*(MOD(A1,10)<4)*MOD(A1,10)*2+1,2)
```

or

```
=A1&MID("thstndrd"&REPT("th",16)&REPT("thstndrdththt hththt hth",8),2*MOD(A1,100)+1,2)
```

Simple Math with Formulas

How to Analyze Spreadsheet Formulas

With the F9 function key, you can evaluate spreadsheet formulas from the inside out or from left to right:

Example:

Input in cell A2: 07.01.1901

Spreadsheet formula: `=QUOTIENT(DAY(EOMONTH(A2,0))+13-WEEKDAY(EOMONTH(A2,0),2),7)`

We select the first occurrence of `EOMONTH(A2,0)` and press F9.

We get `=QUOTIENT(DAY(397)+13-WEEKDAY(EOMONTH(A2,0),2),7)`.

Next, we select the part `DAY(397)` and press F9 again.

It becomes `=QUOTIENT(31+13-WEEKDAY(EOMONTH(A2,0),2),7)`.

Now we select the part `EOMONTH(A2,0)` and press F9.

We get `=QUOTIENT(31+13-WEEKDAY(397,2),7)`.

Next, we select the part `WEEKDAY(397,2)` and press F9.

It results in `=QUOTIENT(31+13-4,7)`.

Now we select the part `31+13-4` and press F9.

The result is `=QUOTIENT(40,7)`, and then the result is `5`, since the QUOTIENT function calculates an integer result without remainder.

Many tasks in Excel can be solved with simple mathematics. Here are a few examples.

Testing Date Formulas

How can I ensure that an Excel date formula is correct? I need to test it!

This works best:

- I generate all days of the relevant date range in column A. 1-Jan-1901 to 31-Dec-2099, for example.
- I enter all possible formulas in columns B, C, D, … and copy down.
- I compare all results to the reference formula in column B.

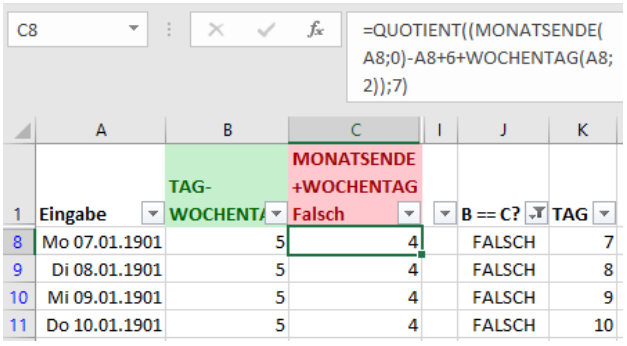
<u>Example</u>

How many ISO calendar weeks does a given month touch?

The formulas:

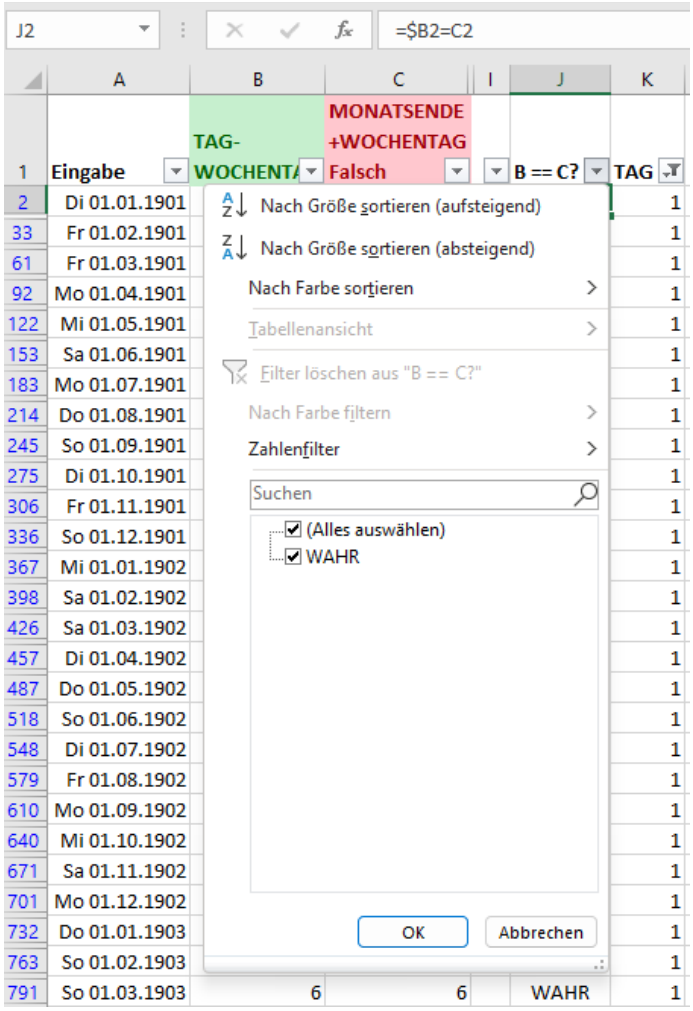
	B	C	J	K
1	**DAY-WEEKDAY**	**EOMONTH+WEEKDAY Wrong!**	**B==C?**	**DAY**
2	=QUOTIENT(DAY(EOMONTH(A2,0))+13-WEEKDAY(EOMONTH(A2,0),2),7)	=QUOTIENT((EOMONTH(A2,0)-A2+6+WEEKDAY(A2,2)),7)	=$B2=C2	=DAY(A2)

Copy row 2 down. If you apply a filter in row 1 and filter column J for the value **FALSE**, then you can see quickly for which date values column C differs from reference formula in column B:



	A	B	C	I	J	K
1	Eingabe	TAG-WOCHENT/	MONATSENDE +WOCHENTAG Falsch		B == C?	TAG
8	Mo 07.01.1901	5	4		FALSCH	7
9	Di 08.01.1901	5	4		FALSCH	8
10	Mi 09.01.1901	5	4		FALSCH	9
11	Do 10.01.1901	5	4		FALSCH	10

If you filter column K for day 1 (first day of month) then you will see that the formula in column C is correct at month start (but unfortunately only then):



First or last Weekday of a Month

You need to find the second Saturday in September 2009, or you want to determine the third last Saturday of the same month – by the way the same day?

	A	B	C	D
1	Any day in month	i	Weekday	i-th weekday in month
2	Di 01-Sep-2009	2	7	Sa 12-Sep-2009
3	Di 01-Sep-2009	2	7	Sa 12-Sep-2009
4	Any day in month	i	Weekday	Last but (i-1)-th weekday in month
5	Mi 30-Sep-2009	3	7	Sa 12-Sep-2009
6	Mi 30-Sep-2009	3	7	Sa 12-Sep-2009

Two general solutions, one a worksheet formula, and the other a user-defined function:

Worksheet Formulas		
Range	Formula	
D2	**D2**	=FLOOR(A2+(MOD(A2,7)>MOD(C2,7))*(6-MOD(C2,7))+(B2-1)*7,7)+MOD(C2,7)
D3	**D3**	=fwim(A3,C3,B3)
D5	**D5**	=CEILING(A5-(MOD(A5,7)<MOD(C5,7))*MOD(C5,7)-(B5-1)*7,7)+MOD(C5,7)-7
D6	**D6**	=lwim(A6,C6,B6)

fwim Program Code

```
Function fwim(d As Date, wd As Integer, Optional i As Integer = 1) As Date
'Fwim [First weekday in month] returns i-th weekday wd of month d:
'd  - date of first day of desired month
'wd - weekday: 1-Sunday, 2-Monday, ..., 7-Saturday
'i  - 1 will return first, 2 second, etc.
'(C) (P) by Bernd Plumhoff 12-Sep-2009 PB V0.20
Dim dr As Date, wd0 As Integer
dr = DateSerial(Year(d), Month(d), 1) 'Ensure to start with first day of month
wd0 = wd Mod 7 'Saturday = 0
fwim = Int((dr + ((dr Mod 7) > wd0) * (wd0 - 6) + (i - 1) * 7) / 7) * 7 + wd0
End Function
```

lwim Program Code

```
Function lwim(d As Date, wd As Integer, Optional i As Integer = 1) As Date
'Lwim [Last weekday in month] returns last but (i-1)-th weekday wd of month d:
'd  - date of last day of desired month
'wd - weekday: 1-Sunday, 2-Monday, ..., 7-Saturday
'i  - 1 will return last, 2 last but one, etc.
'(C) (P) by Bernd Plumhoff 12-Sep-2009 PB V0.20
Dim dr As Date, wd0 As Integer
dr = DateSerial(Year(d), Month(d) + 1, 0) 'Ensure to start with last day of month
wd0 = wd Mod 7 'Saturday = 0
lwim = Int((dr + ((dr Mod 7) < wd0) * wd0 - (i - 1) * 7 + 6) / 7) * 7 + wd0 - 7
End Function
```

Same Weekday and Calendarweek last year

You want to determine a day about one year ago but it should be the same weekday? You have two reasonable choices, I think:

The same weekday and the same calendarweek of previous year:

```
=B3 - (52 + (CALENDARWEEK(B3 - MONTH(B3) + 6, 21) <> CALENDARWEEK(B3 -
MONTH(B3) - 358, 21))) * 7
```

Please note that this formula logically extends values if same calendarweek does not exist.

Closest same weekday to direct day of source year:

```
=DATE(F$2, MONTH(B3), DAY(B3)) + CHOOSE(WEEKDAY(B3) - WEEKDAY(DATE(F$2,
MONTH(B3), DAY(B3))) + 7, 1, 2, 3, -1, -2, -1, 0, 1, 2, 3, -3, -2, -1)
```

B2 | 2021

	A	B	C	D	E	F	G	H
1			Calendar-week	Same weekday and calendarweek	Calendar-week	Closest same weekday to direct day of source year		
2	Day \ Year	2021	2021	Previous year	Previous year	2020		
3	1	Fri 01/01/2021	53	Fri 27/12/2019	52	Fri 03/01/2020		
4	2	Sat 02/01/2021	53	Sat 28/12/2019	52	Sat 04/01/2020		
5	3	Sun 03/01/2021	53	Sun 29/12/2019	52	Sun 05/01/2020		
6	4	Mon 04/01/2021	1	Mon 30/12/2019	1	Mon 06/01/2020		
7	5	Tue 05/01/2021	1	Tue 31/12/2019	1	Tue 07/01/2020		
8	6	Wed 06/01/2021	1	Wed 01/01/2020	1	Wed 08/01/2020		
9	7	Thu 07/01/2021	1	Thu 02/01/2020	1	Thu 09/01/2020		
10	8	Fri 08/01/2021	1	Fri 03/01/2020	1	Fri 10/01/2020		
11	9	Sat 09/01/2021	1	Sat 04/01/2020	1	Sat 11/01/2020		
12	10	Sun 10/01/2021	1	Sun 05/01/2020	1	Sun 12/01/2020		
13	11	Mon 11/01/2021	2	Mon 06/01/2020	2	Mon 13/01/2020		
14	12	Tue 12/01/2021	2	Tue 07/01/2020	2	Tue 14/01/2020		
353	351	Fri 17/12/2021	50	Fri 11/12/2020	50	Fri 18/12/2020		
354	352	Sat 18/12/2021	50	Sat 12/12/2020	50	Sat 19/12/2020		
355	353	Sun 19/12/2021	50	Sun 13/12/2020	50	Sun 20/12/2020		
356	354	Mon 20/12/2021	51	Mon 14/12/2020	51	Mon 21/12/2020		
357	355	Tue 21/12/2021	51	Tue 15/12/2020	51	Tue 22/12/2020		
358	356	Wed 22/12/2021	51	Wed 16/12/2020	51	Wed 23/12/2020		
359	357	Thu 23/12/2021	51	Thu 17/12/2020	51	Thu 24/12/2020		
360	358	Fri 24/12/2021	51	Fri 18/12/2020	51	Fri 25/12/2020		
361	359	Sat 25/12/2021	51	Sat 19/12/2020	51	Sat 26/12/2020		
362	360	Sun 26/12/2021	51	Sun 20/12/2020	51	Sun 27/12/2020		
363	361	Mon 27/12/2021	52	Mon 21/12/2020	52	Mon 28/12/2020		
364	362	Tue 28/12/2021	52	Tue 22/12/2020	52	Tue 29/12/2020		
365	363	Wed 29/12/2021	52	Wed 23/12/2020	52	Wed 30/12/2020		
366	364	Thu 30/12/2021	52	Thu 24/12/2020	52	Thu 31/12/2020		
367	365	Fri 31/12/2021	52	Fri 25/12/2020	52	Fri 01/01/2021		

Not same calendarweek but logical extention

08/01/2020 is closer to 06/01 but not same calendarweek

Budget Control

"If you want creativity, take a zero off your budget. If you want sustainability, take off two zeros." [Jaime Lerner]

This is a simple budget plan and budget control.

	A	B	C	D	E	F	G	H	I	J
1		**Budget / Cost [$]**					**Weights (seasonal)**			
2										
3		**Sales 1**	**Sales 2**	**Purchase**	**Personell**	**TOTAL**	**Sales 1**	**Sales 2**	**Purchase**	**Personell**
4										
5		**400.000**	**100.000**	**200.000**	**500.000**	**1.200.000**	Easter / Xmas	Summer / Swimming		
6										
7	**January**	8.163	1.961	24.000	40.816	74.940	1	1	6	4
8	**February**	40.816	1.961	16.000	40.816	99.593	5	1	4	4
9	**March**	81.633	3.922	12.000	51.020	148.575	10	2	3	5
10	**April**	48.980	3.922	16.000	40.816	109.717	6	2	4	4
11	**May**	0	11.765	20.000	45.000	76.765	0	6	5	4
12	**June**	0	15.686	20.000	40.219	75.905	0	8	5	4
13	**July**	50.000	19.608	16.000	40.219	125.827	0	10	4	4
14	**August**	0	19.608	15.000	40.219	74.827	0	10	4	4
15	***September***	*12.623*	*11.765*	*20.333*	*40.219*	*84.940*	2	6	5	4
16	***October***	*37.868*	*3.922*	*24.400*	*40.219*	*106.409*	6	2	6	4
17	***November***	*56.803*	*1.961*	*4.067*	*40.219*	*103.049*	9	1	1	4
18	***December***	*63.114*	*3.922*	*12.200*	*40.219*	*119.454*	10	2	3	4
19										
20	**Total Budget/*Forecast***	**400.000**	**100.000**	**200.000**	**500.000**	**1.200.000**				
21										
22		**Legende:**								
23		Actual exceeds plan								
24		Actual is less than plan								

In cells B5:E5, enter the annual planned budgets for your departments or cost centers. In cells G7:J18, enter the seasonal weights for the departments. Examples include fixed items (Easter and Christmas) as well as bath items. Purchasing incurs costs 2 months in advance. The general formula `=(B$5-SUM(B$6:B6))*G7/SUM(G7:G$19)` is initially copied into all cells in the range B7:F18. Please note that rows 6 and 19 are intentionally left blank.

Throughout the year, you overwrite the planned costs with the actual costs incurred. The remaining budget for the year will then automatically adjust to maintain the total planned costs for each department.

Once these remaining values for a department become unrealistic, they should be overwritten with realistic forecast values, and, for example, a variance information should be displayed at the bottom.

Increment Lowest Significant Digit

How can you increment the lowest significant digit of a number?

Example: If you have the number 123.4567 you will need a function which calculates 0.0001 to add.

	A	B
1	**Input**	**Output**
2	0,005	0,001
3	0,09	0,01
4	1,23	0,01
5	1,1	0,1
6	123	1
7	1027	1
8	10000	10000
9	1000	1000
10	100	100
11	10	10
12	1	1
13	0,1	0,1
14	0,01	0,01
15	0,001	0,001
16	0,0001	0,0001
17	0,333333333	1E-15
18	3,141592654	1E-14
19	-123	1
20	-1027	1
21	-10000	10000
22	-1000	1000
23	-100	100
24	-10	10
25	-1	1

Worksheet Formulas		
Range	Formula	
B2:B25	**B2**	=10^(--RIGHT(TEXT(ABS(A2),"#,###############E+000"), 4) + 7 -LEN(TEXT(ABS(A2),"#,###############E+000")))

See also Next Floating Point Number – *sbNextFloat*, page 40.

Linear Breakdown

How can you break down annual data like

	A	B
1	**Year**	**Annual Data**
2	2019	29.063.000,00
3	2020	31.388.000,00
4	2021	24.826.000,00
5	2022	27.436.000,00

into monthly data which is piece-wise linear

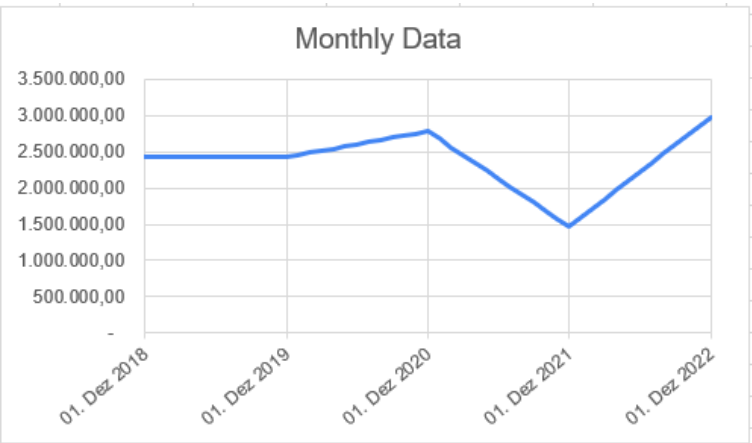
	A	B	C	D
7	**Month Ends**	**Monthly Data**	**Rounded preserving the sum**	**StDev(Data)**
8	31. Dez 2018	1.971.555,79		296.897,83
9	31. Jan 2019	2.040.842,08	2.040.842,08	
10	28. Feb 2019	2.110.128,37	2.110.128,37	
11	31. Mrz 2019	2.179.414,66	2.179.414,66	
12	30. Apr 2019	2.248.700,95	2.248.700,95	
13	31. Mai 2019	2.317.987,23	2.317.987,23	
14	30. Jun 2019	2.387.273,52	2.387.273,53	
15	31. Jul 2019	2.456.559,81	2.456.559,81	
16	31. Aug 2019	2.525.846,10	2.525.846,10	
17	30. Sep 2019	2.595.132,39	2.595.132,38	
18	31. Okt 2019	2.664.418,67	2.664.418,68	
19	30. Nov 2019	2.733.704,96	2.733.704,96	
20	31. Dez 2019	2.802.991,25	2.802.991,25	
21	31. Jan 2020	2.774.172,08	2.774.172,08	
22	29. Feb 2020	2.745.352,92	2.745.352,92	
23	31. Mrz 2020	2.716.533,75	2.716.533,75	
24	30. Apr 2020	2.687.714,58	2.687.714,58	
25	31. Mai 2020	2.658.895,42	2.658.895,42	
26	30. Jun 2020	2.630.076,25	2.630.076,25	
27	31. Jul 2020	2.601.257,08	2.601.257,08	
28	31. Aug 2020	2.572.437,92	2.572.437,92	
29	30. Sep 2020	2.543.618,75	2.543.618,75	
30	31. Okt 2020	2.514.799,58	2.514.799,58	
31	30. Nov 2020	2.485.980,42	2.485.980,42	
32	31. Dez 2020	2.457.161,25	2.457.161,25	
33	31. Jan 2021	2.397.418,49	2.397.418,49	
34	28. Feb 2021	2.337.675,74	2.337.675,74	
35	31. Mrz 2021	2.277.932,98	2.277.932,98	
36	30. Apr 2021	2.218.190,22	2.218.190,22	
37	31. Mai 2021	2.158.447,47	2.158.447,47	
38	30. Jun 2021	2.098.704,71	2.098.704,71	
39	31. Jul 2021	2.038.961,96	2.038.961,96	
40	31. Aug 2021	1.979.219,20	1.979.219,20	
41	30. Sep 2021	1.919.476,44	1.919.476,44	
42	31. Okt 2021	1.859.733,69	1.859.733,69	
43	30. Nov 2021	1.799.990,93	1.799.990,93	
44	31. Dez 2021	1.740.248,17	1.740.248,17	
45	31. Jan 2022	1.824.261,27	1.824.261,27	
46	28. Feb 2022	1.908.274,38	1.908.274,38	
47	31. Mrz 2022	1.992.287,48	1.992.287,48	
48	30. Apr 2022	2.076.300,58	2.076.300,58	
49	31. Mai 2022	2.160.313,68	2.160.313,68	
50	30. Jun 2022	2.244.326,78	2.244.326,78	
51	31. Jul 2022	2.328.339,88	2.328.339,89	
52	31. Aug 2022	2.412.352,99	2.412.352,98	
53	30. Sep 2022	2.496.366,09	2.496.366,09	
54	31. Okt 2022	2.580.379,19	2.580.379,19	
55	30. Nov 2022	2.664.392,29	2.664.392,29	
56	31. Dez 2022	2.748.405,39	2.748.405,39	

?

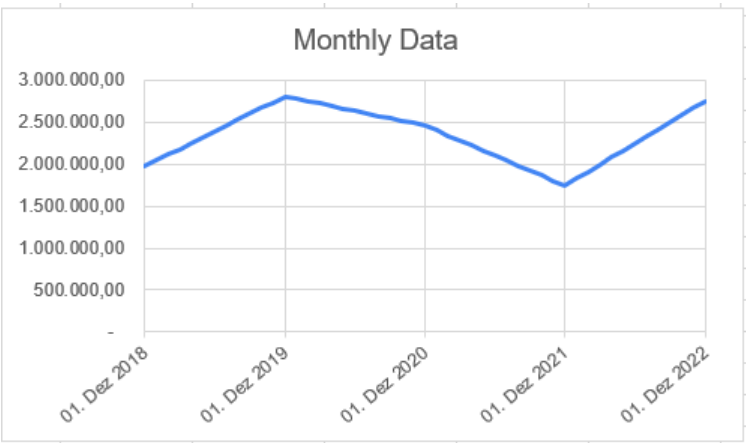
Enter your annual data into cells A2:B5. In cells A8:A56 you enter your month end dates, sorted ascending. As a start value you enter into cell B8 `=B2/12`. In cell B9 you enter `=VLOOKUP DATE(YEAR($A9)- 1,12,31),A$8:B8,2) + (MONTH($A9)) *`

`(VLOOKUP(YEAR($A9),$A$2:$B$5,2) - VLOOKUP(DATE(YEAR($A9) - 1,12,31),A$8:B8,2) * 12) / 78` and copy down. In cell D8 enter `=STDEV.P(B9:B56)`. If your monthly data needs to sum up to the annual values exactly, enter into cell C9 `=ROUND(B9,2)` and into D10 `=ROUND(SUM(B$9:B10),2)-SUM(C$9:C9)` and copy down again. This is just a pragmatic rounding method without VBA. In case you want to correctly round I suggest to use RoundToSum.

Now you get this piece-wise linear output:



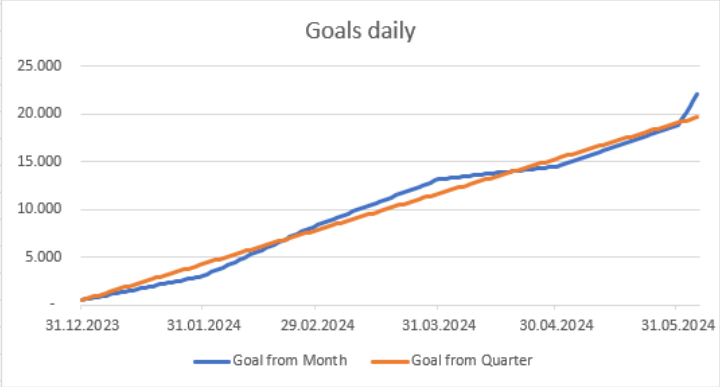
The start value in cell B8 offers one degree of freedom. If you enter 2,000,000 here you get:



Another example would be a breakdown of quarterly or monthly data into daily data with this formula in cell L2:

```
=VLOOKUP(DATE(YEAR($K3),MONTH($K3),0),K$2:L2,2) +
(DAY($K3))*(VLOOKUP(DATE(YEAR($K3),MONTH($K3),1),$A$2:$C$7,3) -
VLOOKUP(DATE(YEAR($K3),MONTH($K3),0),K$2:L2,2) *
VLOOKUP(DATE(YEAR($K3),MONTH($K3),1),$A$2:$C$7,2)) /
(VLOOKUP(DATE(YEAR($K3),MONTH($K3),1),$A$2:$C$7,2) *
(1+VLOOKUP(DATE(YEAR($K3),MONTH($K3),1),$A$2:$C$7,2))/2)
```

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	**Month**	**Days**	**Goal**	**% of Total**		**Quarter**	**Days**	**Total**			**Date**	**Goal from Month**	**Goal from Quarter**
2	01.01.2024	31	56.000	3,5%		01.01.2024	91	560000	35%		31.12.2023	500	500
3	01.02.2024	29	168.000	10,5%		01.04.2024	66	1040000	65%		01.01.2024	582	623
4	01.03.2024	31	336.000	21,0%							02.01.2024	663	746
5	01.04.2024	30	416.000	26,0%							03.01.2024	745	869
6	01.05.2024	31	520.000	32,5%							04.01.2024	827	992
7	01.06.2024	5	104.000	6,5%							05.01.2024	908	1.115
8	**Total**	**157**	**1.600.000**	**100,0%**							06.01.2024	990	1.237
9											07.01.2024	1.072	1.360
10	**Forecast**										08.01.2024	1.153	1.483
11	**Month**	**Days**	**Goal**	**% of Total**							09.01.2024	1.235	1.606
12	Jan	31	56.000	2,6%							10.01.2024	1.317	1.729
13	Feb	29	168.000	7,9%							11.01.2024	1.398	1.852
14	Mar	31	336.000	15,8%							12.01.2024	1.480	1.975
15	Apr	30	416.000	19,6%							13.01.2024	1.561	2.098
16	May	31	520.000	24,5%							14.01.2024	1.643	2.221
17	Jun	30	624.000	29,4%							15.01.2024	1.725	2.344
18	**Total**	**182**	**2.120.000**	**100,0%**							16.01.2024	1.806	2.467
19											17.01.2024	1.888	2.589
20											18.01.2024	1.970	2.712
21											19.01.2024	2.051	2.835
22											20.01.2024	2.133	2.958
23											21.01.2024	2.215	3.081
24											22.01.2024	2.296	3.204
25											23.01.2024	2.378	3.327
26											24.01.2024	2.460	3.450
27											25.01.2024	2.541	3.573
28											26.01.2024	2.623	3.696
29											27.01.2024	2.705	3.819
30											28.01.2024	2.786	3.941
31											29.01.2024	2.868	4.064
32											30.01.2024	2.950	4.187
33											31.01.2024	3.031	4.310
34											01.02.2024	3.215	4.433



If this approach does not satisfy you and if you have seasonal weights of your data at hand, you can apply the better method budget control (page 91).

Merge two columns justified into one

If you need to merge two columns with data justified into one column, you can use a monospaced font (`Courier New` or `Lucida San Typewriter`) and a worksheet formula:

	A	B	C	D	E
1	There are two monospaced TrueType fonts supplied by Microsoft:				
2	1. Courier New				
3	2. Lucida Sans Typewriter				
4					
5	Now let us try to simultaneously left-align text and right-align text and numbers for a table in a single column with the least possible width:				
6					
7				**Without Overlap**	**With Overlap**
8	**Sample Input Table Column 1**	**Sample Input Table Column 2**	**Helper Cells**	Lucida Sans Typewriter Output	Lucida Sans Typewriter Output
9	alpha	1,00	Max text length	alpha 1,00	alpha 1,00
10	beta	123.456,78	7	beta 123.456,78	beta 123.456,78
11	gamma	123,00	Max formatted number length	gamma 123,00	gamma 123,00
12	delta	123,45	10	delta 123,45	delta 123,45
13	epsilon	0,00	Max total length	epsilon 0,00	epsilon 0,00
14			14		
15			Number format		
16			#.##0,00		
17			(change to '#,##0.00' in UK or US Excel version)		

Worksheet Formulas		
Range	Formula	
C10	**C12**	=MAX(LEN(TEXT(B9:B13,"#.##0,00")))
C12	**C12**	=MAX(LEN(TEXT(B9:B13,"#.##0,00")))
C14	**C14**	=MAX(LEN(A9:A13)+LEN(TEXT(B9:B13,"#.##0,00")))
D9:D13	**D9**	=$A9&REPT(" ",$C$10+$C$12+1-LEN($A9)-LEN(TEXT($B9,$C$16)))&TEXT($B9,$C$16)
E9:E13	**E9**	=A9&REPT(" ",$C$14+1-LEN(A9)-LEN(TEXT($B9,$C$16))) & TEXT($B9,$C$16)

Minimum Truck Load Problem

Suppose you have a supplier who will only sell items in certain quantities to make shipping manageable. So you have to buy a full truck load of items to begin with, and after that anywhere between a half and full load.

A full load is 10 items. So that's the minimum order. After that you have to order at least another half load, so 5 more items. Between that and the next full load, they don't care. They just don't want to ship a truck that's less than half full.

So the item quantities you can buy are: 0, 15, 16, 17, 18, 19, 20, 25, 26, 27, 28, 29, 30, 35 …

Which formula will tell me for any number of required items the minimum shipping order?

Here is my general solution:

	A	B
1	**Truck capacity [Pieces]**	10
2	**Minimum order [Trucks]**	1
3	**Minimum truck load [%]**	50%
4		
5	**Input [Pieces]**	**Output [Accepted order of Pieces]**
6	0	0
7	1	10
8	2	10
9	3	10
10	4	10
11	5	10
12	6	10
13	7	10
14	8	10
15	9	10
16	10	10
17	11	19
18	12	19
19	13	19
20	14	19
21	15	19
22	16	19
23	17	19
24	18	19
25	19	19
26	20	20
27	21	29
28	22	29
29	23	29
30	24	29
31	25	29
32	26	29
33	27	29
34	28	29
35	29	29

Worksheet Formulas		
Range	Formula	
B6:B35	**B6**	=ROUNDUP(IF(A6:A35<=0, 0, IF(A6:A35<=$B$1*$B$2, $B$1*$B$2, $B$1*INT(A6:A35/$B$1) + IF(MOD(A6:A35, $B$1) > 0, MAX(MOD(A6:A35, $B$1), $B$1*$B$3), 0))), 0)

Count Trailing Zeros

You need to count trailing zeros of an integer? Here you go:

	A	B
1	Count trailing zeros	
2	Input	Output
3	300	2
4	6.000	3
5	1.200.000.000.000	11
6	-300	2
7	-6.000	3
8	-1.200.000.000.000	11

Worksheet Formulas		
Range	Formula	
B3:B5	**B3**	=(A3<>0) * SUM(--(A3=ROUND(A3, -SEQUENCE(15))))
B6:B9	**B6**	=LEN(ABS(A6)) - LEN(SUM(MID(ABS(A6), LEN(ABS(A6)) + 1 - SEQUENCE(LEN(ABS(A6))), 1) * 10 ^ (LEN(ABS(A6)) - SEQUENCE(LEN(ABS(A6)))))

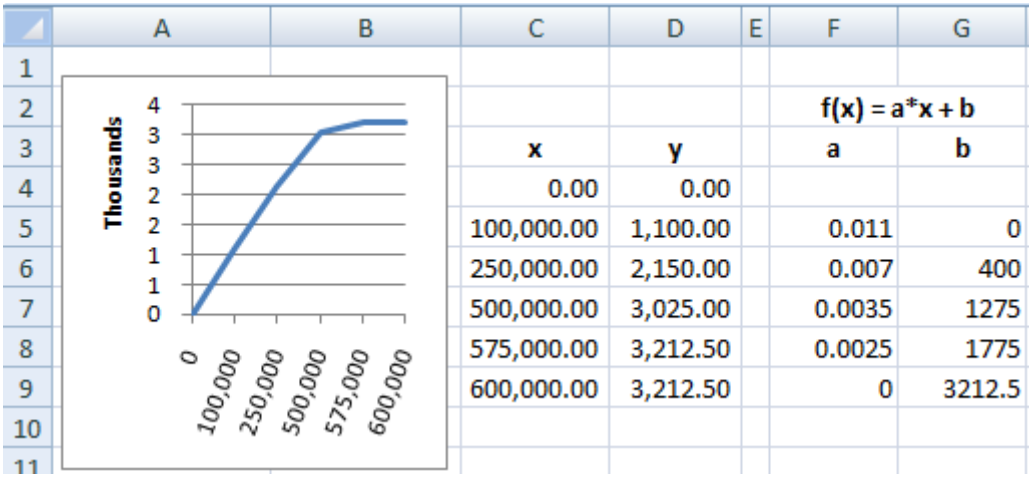
The formula `=(A3<>0) * SUM(--(A3=ROUND(A3, -SEQUENCE(15))))` in cell B3 is shorter and more elegent by far.

Minimax Interpolation

Alexander W. came up with yet another interpolation formula for problems which you can often encounter with taxes or other fee structures:

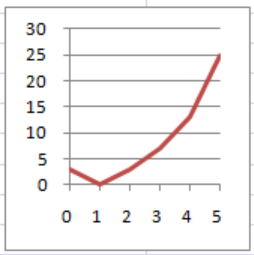
```
=MIN(C6%%*{110,70,35,25}+{0,400,1275,1775},3212.5)
```

See below, for the input value 250,000 this would result in 2,150:

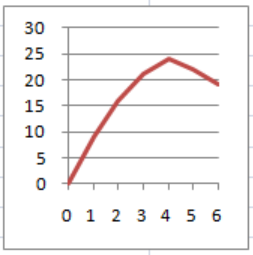


	A	B	C	D	E	F	G
1							
2						f(x) = a*x + b	
3			x	y		a	b
4			0.00	0.00			
5			100,000.00	1,100.00		0.011	0
6			250,000.00	2,150.00		0.007	400
7			500,000.00	3,025.00		0.0035	1275
8			575,000.00	3,212.50		0.0025	1775
9			600,000.00	3,212.50		0	3212.5
10							
11							

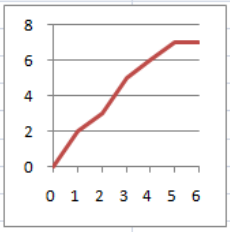
How do you calculate his function parameters and what are the prerequisites or the conditions under which his function can be applied? His formula can be applied if and only if the slopes between adjacent points are monotonously increasing or decreasing:



12					f(x) = a*x + b				
13			x	y	a	b		Formula	Test
14			0.00	3.00					
15			1.00	0.00	-3	3			
16			2.00	3.00	3	-3		=MAX(C16*{-3,3,4,6,12}+{3,-3,-5,-11,-35})	3
17			3.00	7.00	4	-5			
18			4.00	13.00	6	-11			
19			5.00	25.00	12	-35			
20									
21									
22					f(x) = a*x + b				
23			x	y	a	b		Formula	Test
24			0.00	0.00					
25			1.00	9.00	9	0		=MIN(C25*{9,7,5,3,-2,-3}+{0,2,6,12,32,37})	9
26			2.00	16.00	7	2			
27			3.00	21.00	5	6			
28			4.00	24.00	3	12			
29			5.00	22.00	-2	32			
30			6.00	19.00	-3	37			
31									



An example for which you cannot apply this solution:



34					f(x) = a*x + b			
35			x	y	a	b		Formula
36			0.00	0.00				
37			1.00	2.00	2	0		
38			2.00	3.00	1	1		
39			3.00	5.00	2	-1		Slopes need to increase or to decrease monotonously!
40			4.00	6.00	1	2		
41			5.00	7.00	1	2		
42			6.00	7.00	0	7		
43								

If this approach is applicable it is pretty fast. But if the number of interpolations and the number of given points increase (let us say some hundreds) then the runtime of this approach is not considerably faster than my user-defined function *sbInterp* (see page 129).

I recommend to use the Minimax approach not with a manual calculation but with a semi-automated one:

C3 =IF(A3<>"",(B3-B2)/(A3-A2),"")

	A	B	C	D	E	F
1	x	y	y = f(x) = a*x + b		# data points	Test input
2	0.00	0.00	a	b	7	2.50
3	1.00	9.00	9	0	Slope increasing?	Minimax formula
4	2.00	16.00	7	2	FALSE	
5	3.00	21.00	5	6	Slope decreasing?	Minimax formula
6	4.00	24.00	3	12	TRUE	=MIN(F2*{9,7,5,3,-2,-3}+{0,2,6,12,32,37})
7	5.00	22.00	-2	32		
8	6.00	19.00	-3	37		Test output
9						18.50
10						All-in-one with helper columns
11						=MIN(F2*{9,7,5,3,-2,-3}+{0,2,6,12,32,37})
12						All-in-one formula with helper columns
13						{=IF(AND(C3:INDEX(C:C,COUNT(A:A))<=C4:INDEX(C:C,COUNT(A:A)+1)),"=MAX(F2*{"&sbCat(C3:INDEX(C:C,COUNT(A:A)+1))&"}+{"&sbCat(D3:INDEX(D:D,COUNT(A:A)+1))&"})",IF(AND(C3:INDEX(C:C,COUNT(A:A))>=C4:INDEX(C:C,COUNT(A:A)+1)),"=MIN(F2*{"&sbCat(C3:INDEX(C:C,COUNT(A:A)+1))&"}+{"&sbCat(D3:INDEX(D:D,COUNT(A:A)+1))&"})","No solution: slopes need to increase or to decrease monotonously!"))}

These are the necessary worksheet formulas:

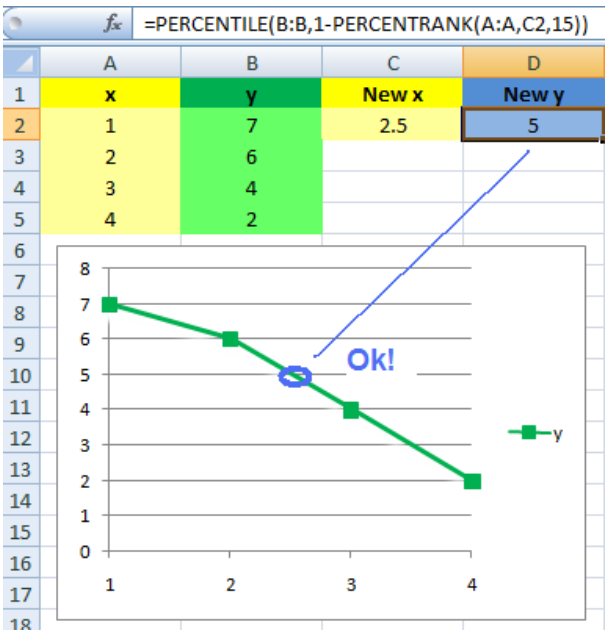
```
C3 =IF(A3<>"",(B3-B2)/(A3-A2),"") [copy down as far as necessary]
D3 =IF(A3<>"",B2-C3*A2,"") [copy down]
E2 =COUNT(A:A)
E4 =AND(C3:INDEX(C:C,E2)<=C4:INDEX(C:C,E2+1)) [Array formula]
E6 =AND(C3:INDEX(C:C,E2)>=C4:INDEX(C:C,E2+1)) [Array formula]
F4 =REPT("=MAX(F2*{"&sbCat(C3:INDEX(C:C,E2+1))&"}+{"&sbCat(D3:INDEX(D:D,E2+1))&"})",E4)
F6 =REPT("=MIN(F2*{"&sbCat(C3:INDEX(C:C,E2+1))&"}+{"&sbCat(D3:INDEX(D:D,E2+1))&"})",E6)
F7 =REPT("No solution: slopes need to increase or to decrease monotonously!",NOT(E4+E6))
F9 =IF(E4,sbEval(F4),IF(E6,sbEval(F6),"")) [sbEval is a UDF:
Function sbEval(s As String) As Variant
    Application.Volatile
    sbEval = Evaluate(s)
End Function
]
```

Interpolation with PERCENTILE / PERCENTRANK

"Everything will be okay in the end. If it's not okay, then it's not the end." [Ed Sheeran]

Lorimer M. came up with a nice and quick but limited linear worksheet function interpolation which works for monotonic functions:

So far everything seems to be ok and you start to think that PERCENTILE and PERCENTRANK offer a nice and attractive way to interpolate linearly.



But:

=PERCENTILE(B:B,1-PERCENTRANK(A:A,C2,15))

	A	B	C	D
1	x	y	New x	New y
2	1	1	2.5	3
3	2	6		
4	3	4		
5	4	2		

Now we realize that this approach only works if x values are increasing and y values are decreasing (increasing if you omit "1-" before PERCENTRANK). So this approach is not only not extrapolating, but also not informing if its results are nonsense!

I did not and I still do not like it too much but at least I would like to show how (better: when) it could and - if ever - how it maybe should be applied:

=PERCENTILE(B23:B27,E24+SIGN(E23-E24)*PERCENTRANK(A23:A27,D27,15))/(E22*(E23+E24))

	A	B	C	D	E	F
1	Linear Interpolation					
2						
3		y = ln(x)			Test	Test Formula
4	X values	Y Values		X monotonically increasing?	1	{=PRODUCT(--(A5:A8<=A6:A9))}
5	1	0		Y monotonically increasing?	1	{=PRODUCT(--(B5:B8<=B6:B9))}
6	2	0.693147		Y monotonically decreasing?	0	{=PRODUCT(--(B5:B8>=B6:B9))}
7	3	1.098612				
8	4	1.386294		New X	New Y	
9	5	1.609438		2.5	0.8958797	=PERCENTILE(B5:B9,E6+SIGN(E5-
10						E6)*PERCENTRANK(A5:A9,D9,15))/(E4*(E5+E6))
11						
12		y = 1/x			Test	Test Formula
13	X values	Y Values		X monotonically increasing?	1	{=PRODUCT(--(A14:A17<=A15:A18))}
14	1	1		Y monotonically increasing?	0	{=PRODUCT(--(B14:B17<=B15:B18))}
15	2	0.5		Y monotonically decreasing?	1	{=PRODUCT(--(B14:B17>=B15:B18))}
16	3	0.333333				
17	4	0.25		New X	New Y	
18	5	0.2		2.5	0.4166667	=PERCENTILE(B14:B18,E15+SIGN(E14-
19						E15)*PERCENTRANK(A14:A18,D18,15))/(E13*(E14+E15))
20						
21		y = (x-3)^2			Test	Test Formula
22	X values	Y Values		X monotonically increasing?	1	{=PRODUCT(--(A23:A26<=A24:A27))}
23	1	4		Y monotonically increasing?	0	{=PRODUCT(--(B23:B26<=B24:B27))}
24	2	1		Y monotonically decreasing?	0	{=PRODUCT(--(B23:B26>=B24:B27))}
25	3	0				
26	4	1		New X	New Y	
27	5	4		2.5	#DIV/0!	=PERCENTILE(B23:B27,E24+SIGN(E23-
28						E24)*PERCENTRANK(A23:A27,D27,15))/(E22*(E23+E24))

I recommend to use my user-defined function sbInterp (see page129).

Interpolation with TREND

"Work and struggle and never accept an evil that you can change." [André Gide]

If you cannot use macros you might be tempted to use the worksheet function TREND for interpolation. Of course you can't just feed all y-values, x-values (if there are more than 2) and the new x for the unknown y because TREND will just be piecewise accurate.

So you need to identify the surrounding x-values (or the first two resp. the last two in case of extrapolation) for the new x with their corresponding y-values first and then you feed them to TREND.

I don't like this approach and I won't recommend to use it but if you do want to, use it correctly:

=TREND(INDEX(B5:B9,E6):INDEX(B5:B9,E7),INDEX(A5:A9,E6):INDEX(A5:A9,E7),D9)/E4

	A	B	C	D	E	F
1	Linear Interpolation					
2						
3		y = ln(x)			Test	Test Formula
4	X values	Y Values		X monotonically increasing?	1	{=PRODUCT(--(A5:A8<=A6:A9))}
5	1	0			Index	
6	2	0.693147		Lower border index	2	=IF(D9<A5,A5,MATCH(D9,A5:A8,1))
7	3	1.098612		Upper border index	3	=E6+1
8	4	1.386294		New X	New Y	
9	5	1.609438		2.5	0.8958797	=TREND(INDEX(B5:B9,E6):INDEX(B5:B9,E7),INDEX(A5:A9
10						,E6):INDEX(A5:A9,E7),D9)/E4
11						
12		y = 1/x			Test	Test Formula
13	X values	Y Values		X monotonically increasing?	1	{=PRODUCT(--(A14:A17<=A15:A18))}
14	1	1			Index	
15	2	0.5		Lower border index	2	=IF(D18<A14,A14,MATCH(D18,A14:A17,1))
16	3	0.333333		Upper border index	3	=E15+1
17	4	0.25		New X	New Y	
18	5	0.2		2.5	0.4166667	=TREND(INDEX(B14:B18,E15):INDEX(B14:B18,E16),INDE
19						X(A14:A18,E15):INDEX(A14:A18,E16),D18)/E13
20						
21		y = (x-3)^2			Test	Test Formula
22	X values	Y Values		X monotonically increasing?	1	{=PRODUCT(--(A23:A26<=A24:A27))}
23	1	4			Index	
24	2	1		Lower border index	2	=IF(D27<A23,A23,MATCH(D27,A23:A26,1))
25	3	0		Upper border index	3	=E24+1
26	4	1		New X	New Y	
27	5	4		2.5	0.5	=TREND(INDEX(B23:B27,E24):INDEX(B23:B27,E25),INDE
28						X(A23:A27,E24):INDEX(A23:A27,E25),D27)/E22

I deem this approach as inappropriate because you do not really need TREND here - you can easily apply the direct interpolation formula. I think its just too complex and therefore inferior to the (also poor) PERCENTILE / PERCENTRANK approach (see page 100), despite the fact that it can even extrapolate and that it can cope with non-monotonic y-values.

REFA Time Slicing

REFA, the Association for Work Design, Business Organization and Business Development is Germany's oldest organization for work design, business organization and business development. They offer training similar to Six Sigma or Lean Management. If you like to perform a relatively accurate analysis of one month's hotline calls, you can apply this approach:

	A	B	C	D	E	F
1	**Timeslicing according to REFA**					
2						
3	**Hotline Example**				**Comment**	**Possible Warnings and Error Messages**
4	**Relative Confidence (%)**		2		Confidence intervall is typically between 0.1 and 5%	
5	**Length of calculation period (min)**		10080	A typical work month	Monitoring period larger than largest possible base time?	
6	**Shortest possible base time (min)**		2	Shortest Hotline call		
7	**Longest possible base time (min)**		60	Longest Hotline call	Longest possible base time greater than shortest?	
8	**Lower border of smallest time slice (min)**		1		Lower border of smallest time slice 50 - 80% of shortest possible base time?	
9						
10	**Warnings and Error Messages**		**All fine**			
11	Explanation: If you like to methodologically analyze durations of hotline calls, for example ("Which topics were dealt with in which time?") then this example tells you for one month of time logging at a confidence intervall of 1% that hotline staff should roughly estimate the durations of calls by telling to which duration timeslice the call belongs (1-4) and which topic was discussed.					
12	**Helper Calc**	**Timeslice #**	**Width**	**Lower Border**	**Midth**	**Upper Border**
13	4,861392064	1	10	1	5	11
14	8,974179072	2	18	11	9	29
15	13,01564799	3	26	29	13	55
16	17,04345008	4	34	55	17	89

Worksheet Formulas		
Range	Formula	
F4	**F4**	=IF(OR($C$4<0.1,$C$4>5),"Confidence intervall is typically between 0.1 and 5%.","")
F4	**F4**	=IF(OR($C$4<0.1,$C$4>5),"Confidence intervall is typically between 0.1 and 5%.","")
F5	**F5**	=IF($C$5<$C$7,"Calculation period must be larger than largest possible base time.","")
F7	**F7**	=IF($C$7<=$C$6,"Longest possible base time has to be greater than shortest possible base time.","")
F8	**F8**	=IF(OR($C$8<0.5*$C$6,$C$8>0.8*$C$6),"Lower border of smallest time slice should be 50 - 80% of shortest possible base time.","")
C10	**C10**	=IF($F$4&$F$5&$F$6&$F$7&$F$8="","All fine",$F$4&$F$5&$F$6&$F$7&$F$8)
A13	**A13**	=($C$4%)^2*$C$5/2+((($C$4%)^2*$C$5/2)^2+($C$4%)^2*$C$5*D13)^0.5
B13	**B13**	1
C13	**C13**	=ROUND(A13*2,0)
D13	**D13**	=$C$8
E13	**E13**	=(F13-D13)/2
F13	**F13**	=C13+D13
A14:A22	**A14**	=IF(F13<$C$7,($C$4%)^2*$C$5/2+((($C$4%)^2*$C$5/2)^2+($C$4%)^2*$C$5*D14)^0.5,"")
B14:B22	**B14**	=IF(F13<$C$7,B13+1,"")
C14:C22	**C14**	=IF(F13<$C$7,ROUND(A14*2,0),"")
D14:D22	**D14**	=IF(F13<$C$7,F13,"")
E14:E22	**E14**	=IF(F13<$C$7,(F14-D14)/2,"")
F14:F22	**F14**	=IF(F13<$C$7,C14+D14,"")

Roles and Rights

Given a table of roles and which rights are associated with them, and a table of forbidden right combinations - i.e. defining which rights are not to be granted together, how can you now derive the table of forbidden role combinations - i.e. a table which shows which roles cannot be granted together?

Role 1 owns right 1, role 2 contains rights 2 and 4, and role 4 contains rights 4 and 5.

If we do not allow right 1 to be granted together with right 4, then the interesting conclusion is that role 1 can never be given together with role 2 nor role 4.

The grey area in the second table mirrors the values of the white cells in the same table.

	A	B	C	D	E	F
1	Input - Roles and Rights					
2		Role_1	Role_2	Role_3	Role_4	
3	Right_1	x				
4	Right_2		x			
5	Right_3			x		
6	Right_4		x		x	
7	Right_5				x	
8						
9	Input - Forbidden Right Combinations					
10		Right_1	Right_2	Right_3	Right_4	Right_5
11	Right_1				x	
12	Right_2	0				
13	Right_3	0	0			
14	Right_4	x	0	0		
15	Right_5	0	0	0	0	
16						
17	Output - Forbidden Role Combinations					
18		Role_1	Role_2	Role_3	Role_4	
19	Role_1		x		x	
20	Role_2	x				
21	Role_3					
22	Role_4	x				

Worksheet Formulas		
Range	Formula	
B12	**B12**	=TRANSPOSE(C11:F11)
C13	**C13**	=TRANSPOSE(D12:F12)
D14	**D14**	=TRANSPOSE(E13:F13)
E15	**E15**	=F14
B12	**B12**	=TRANSPOSE(C11:F11)
B19	**B19**	=REPT("x", SIGN(MMULT(--(TRANSPOSE(B3:E7)="x"),MMULT(--(B11:F15="x"), --(B3:E7="x")))))

Rounding is Fun

	A	B	C
1	**Rounding is Fun**	**Input**	**Output**
2	Round a number to an integer	3,141592654	3
3	Round a number to two digits	3,141592654	3,14
4	Round a number to next thousand	2020	2000
5	Round a number to 50p	5,672	5,5
6	Round down if 35p or less, else round up	2,36	3
7	Round to 1/64	2,1718	2,171875
8	Show bond price in US T-Note format (see Bloomberg © for example)	100,578125	100-18,5
9	Show bond price in US T-Note format (see Bloomberg © for example) 2nd version	100,578125	100-18½
10	Round a date time to an hour	26.04.2010 16:37:12	26.04.2010 17:00:00
11	Round a date time to next 10 min point	26.04.2010 16:37:12	26.04.2010 16:40:00
12	Round to 1/8 but show 1/4, 1/2, 3/4 instead of 2/8, 4/8, 6/8	-2,5	-2 1/2

Worksheet Formulas		
Range	Formula	
C2	**C2**	=ROUND(B2,0)
C3	**C3**	=ROUND(B3,2)
C4	**C4**	=ROUND(B4,-3)
C5	**C5**	=ROUND(B5 * 2,0) / 2
C6	**C6**	=CEILING(B6 - 0.35,1)
C7	**C7**	=ROUND(B7 * 64,0) / 64
C8	**C8**	=INT(B8) & REPT("-" & MOD(B8,1) * 32, SIGN(MOD(B8,1)))
C9	**C9**	=INT(B9) & REPT("-" & INT(MOD(B9,1) * 32) & REPT(CHAR(187 + INT((MOD(B9,1) - INT(MOD(B9,1) * 32) / 32) * 128)), INT((MOD(B9,1) - INT(MOD(B9,1) * 32) / 32) * 128) > 0), INT(MOD(B9,1) * 128) > 0)
C10	**C10**	=ROUND(B10 * 24,0) / 24
C11	**C11**	=ROUND(B11 * 24 * 6,0) / 24 / 6
C12	**C12**	=CHOOSE(2 + SIGN(B12),"-","0","") & IF(INT(ABS(B12)) <> 0,INT(ABS(B12)) & " ","") & CHOOSE(1 + ROUND((ABS(B12) - INT(ABS(B12))) * 8,0),"", "1/8","1/4","3/8","1/2","5/8","3/4","7/8")

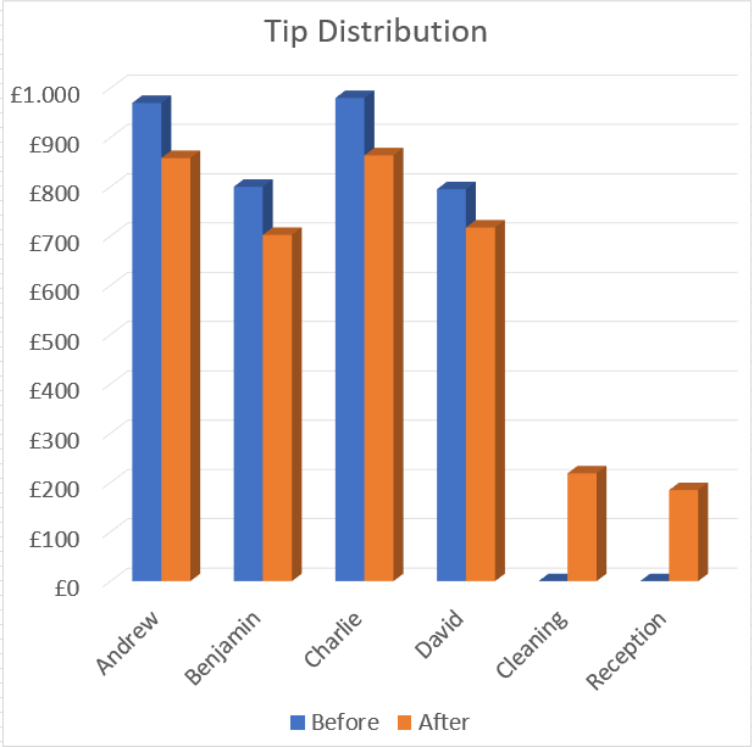
Please note: Cell D5: =ROUND(B5 * 2, 0) / 2 is better than =ROUND(B5 / 5, 1) * 5, because the internal binary representation of floating-point numbers has fewer calculation inaccuracies when multiplied or divided by 2. Cell C8: This result is the maximum number in T-Note format that is less than or equal to the value entered in cell B9.

Tip Distribution

"Flattery is nice but tips will suffice."

You want to split tips fairly between your employees? They may keep 70% of all tips they get directly but 30% should be distributed based on hours worked so that cleaners and reception staff would receive a share, too?

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Date	Total		Andrew		Benjamin		Charlie		David		Cleaning		Reception	
2		€	Hours	€	Hours	€	Hours	€	Hours	€	Hours	€	Hours	€	Hours
3	1	120,00	26,0	50,00	5,0	35,00	7,0	20,00	2,0	15,00	3,0		8,0		1,0
4	2	230,00	29,0	50,00	5,0	80,00	8,0	20,00	2,0	80,00	8,0		6,0		0,0
5	3	105,00	28,0	10,00	1,0	10,00	2,0	15,00	3,0	70,00	7,0		7,0		8,0
6	4	140,00	26,0	0,00	0,0	70,00	7,0	50,00	5,0	20,00	4,0		8,0		2,0
7	5	110,00	17,0	10,00	1,0	80,00	8,0	0,00	0,0	20,00	2,0		2,0		4,0
8	6	55,00	10,0	10,00	1,0	15,00	3,0	30,00	3,0	0,00	0,0		1,0		2,0
9	7	90,00	19,0	5,00	1,0	0,00	0,0	15,00	3,0	70,00	7,0		0,0		8,0
10	8	110,00	32,0	30,00	6,0	30,00	6,0	40,00	4,0	10,00	2,0		6,0		8,0
11	9	95,00	20,0	50,00	5,0	0,00	0,0	25,00	5,0	20,00	2,0		1,0		7,0
12	10	140,00	37,0	60,00	6,0	30,00	6,0	35,00	7,0	15,00	3,0		7,0		8,0
13	11	90,00	11,0	0,00	0,0	30,00	3,0	20,00	2,0	40,00	4,0		0,0		2,0
14	12	115,00	22,0	20,00	2,0	0,00	0,0	25,00	5,0	70,00	7,0		3,0		5,0
15	13	100,00	25,0	30,00	6,0	30,00	3,0	0,00	0,0	40,00	8,0		8,0		0,0
16	14	70,00	17,0	35,00	7,0	30,00	3,0	0,00	0,0	5,00	1,0		2,0		4,0
17	15	80,00	21,0	35,00	7,0	20,00	2,0	0,00	0,0	25,00	5,0		1,0		6,0
18	16	145,00	33,0	35,00	7,0	20,00	4,0	60,00	6,0	30,00	6,0		8,0		2,0
19	17	140,00	17,0	60,00	6,0	40,00	4,0	10,00	1,0	30,00	3,0		0,0		3,0
20	18	160,00	27,0	40,00	8,0	20,00	2,0	40,00	4,0	60,00	6,0		3,0		4,0
21	19	140,00	23,0	0,00	0,0	30,00	6,0	50,00	5,0	60,00	6,0		5,0		1,0
22	20	50,00	20,0	0,00	0,0	0,00	0,0	20,00	4,0	30,00	3,0		7,0		6,0
23	21	90,00	17,0	10,00	1,0	0,00	0,0	10,00	2,0	70,00	7,0		6,0		1,0
24	22	105,00	18,0	10,00	1,0	25,00	5,0	50,00	5,0	20,00	2,0		0,0		5,0
25	23	120,00	34,0	40,00	4,0	35,00	7,0	35,00	7,0	10,00	2,0		6,0		8,0
26	24	180,00	26,0	10,00	2,0	80,00	8,0	20,00	4,0	70,00	7,0		0,0		5,0
27	25	60,00	21,0	0,00	0,0	5,00	1,0	30,00	6,0	25,00	5,0		8,0		1,0
28	26	130,00	29,0	10,00	1,0	80,00	8,0	40,00	4,0	0,00	0,0		8,0		8,0
29	27	20,00	8,0	0,00	0,0	15,00	3,0	0,00	0,0	5,00	1,0		2,0		2,0
30	28	155,00	29,0	50,00	5,0	35,00	7,0	0,00	0,0	70,00	7,0		6,0		4,0
31	29	120,00	29,0	30,00	3,0	40,00	8,0	15,00	3,0	35,00	7,0		6,0		2,0
32	30	105,00	20,0	0,00	0,0	25,00	5,0	10,00	1,0	70,00	7,0		7,0		0,0
33	31	60,00	13,0	0,00	0,0	10,00	2,0	30,00	3,0	20,00	2,0		0,0		6,0
34	**Sum**	3.430,00	704,0	690,00	91,0	920,00	128,0	715,00	96,0	1.105,00	134,0	0,00	132,0	0,00	123,0
35															
36	**Keep**	2.401,00		483,00		644,00		500,50		773,50		0,00		0,00	
37	**Give**		1.029,00		207,00		276,00		214,50		331,50		0,00		0,00
38	**Receive**	1.029,00		133,01		187,09		140,32		195,86		192,94		179,78	
39															
40	**Total**	3.430,00		616,01		831,09		640,82		969,36		192,94		179,78	



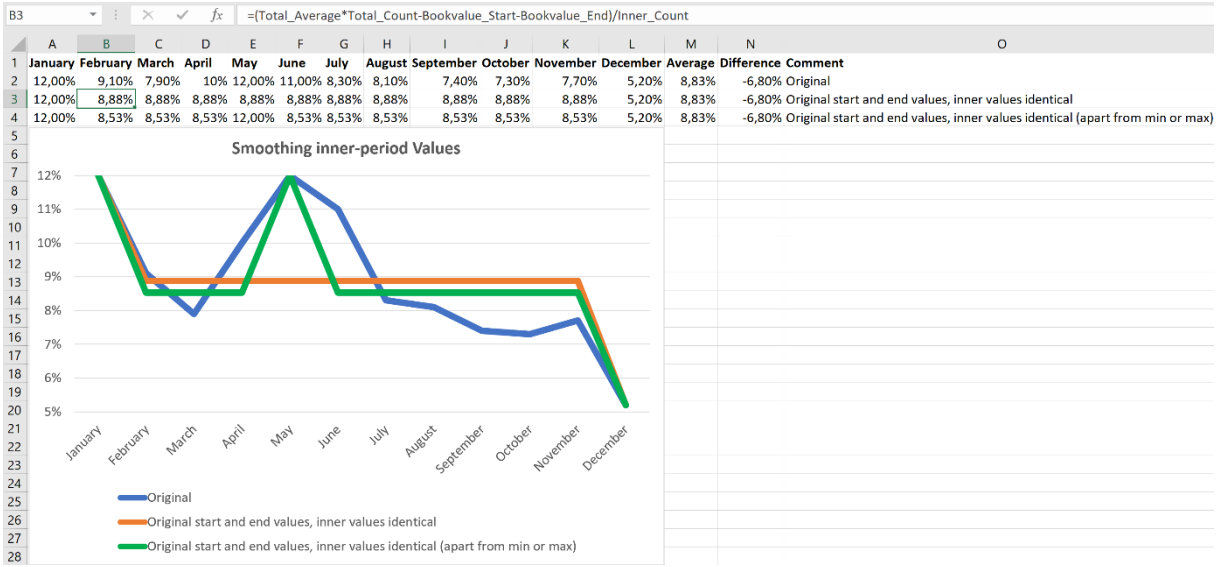
Worksheet Formulas		
Range	Formula	
B3:B34;B36;B38;B40	**B3**	=SUM(D3,F3,H3,J3,L3,N3)
C3:C34;C36;C38;C49	**C3**	=SUM(E3,G3,I3,K3,M3,O3)
B34:O34	**B34**	=SUM(B3:B33)
D36;F36;H36;J36;L36;N36	**D36**	=D34*Keep_Percentage
E37;G37;I37;K37;M37;O37	**E37**	=D34-D36
D38;F38;H38;J38;L38;N38	**D38**	=$C$37*E34/$C$34
D40;F40;H40;J40;L40;N40	**D40**	=D36+D38

Named Ranges		
Name	Scope	Refers to
Keep_Percentage	Workbook	=0.7

Smoothing Inner Period Values

You need to smoothen inner-period values, keeping the book-end values and the overall (total) average?

Two simple approaches:



January	February	March	April	May	June	July	August	September	October	November	December	Average	Difference	Comment
12,00%	9,10%	7,90%	10%	12,00%	11,00%	8,30%	8,10%	7,40%	7,30%	7,70%	5,20%	8,83%	-6,80%	Original
12,00%	8,88%	8,88%	8,88%	8,88%	8,88%	8,88%	8,88%	8,88%	8,88%	8,88%	5,20%	8,83%	-6,80%	Original start and end values, inner values identical
12,00%	8,53%	8,53%	8,53%	12,00%	8,53%	8,53%	8,53%	8,53%	8,53%	8,53%	5,20%	8,83%	-6,80%	Original start and end values, inner values identical (apart from min or max)

Define these names:

Name	Refers to
All_Values	=Sheet1!$A$2:$K$2
Bookvalue_End	=Sheet1!$L$2
Bookvalue_Start	=Sheet1!$A$2
Inner_Count	=(Total_Count-2)
Inner_Values	=Sheet1!$B$2:$K$2
Total_Average	=AVERAGE(All_Values)
Total_Count	=COLUMN(Bookvalue_End)-COLUMN(Bookvalue_Start)+1

If all inner values should be identical, enter into

```
A3: =Bookvalue_Start
B3: =(@Total_Average*@Total_Count-Bookvalue_Start-Bookvalue_End)/@Inner_Count
```
and copy to the right until cell K3

```
L3: =Bookvalue_End
```

If all inner values (with the exception of one or more most extreme values) should be identical, enter into

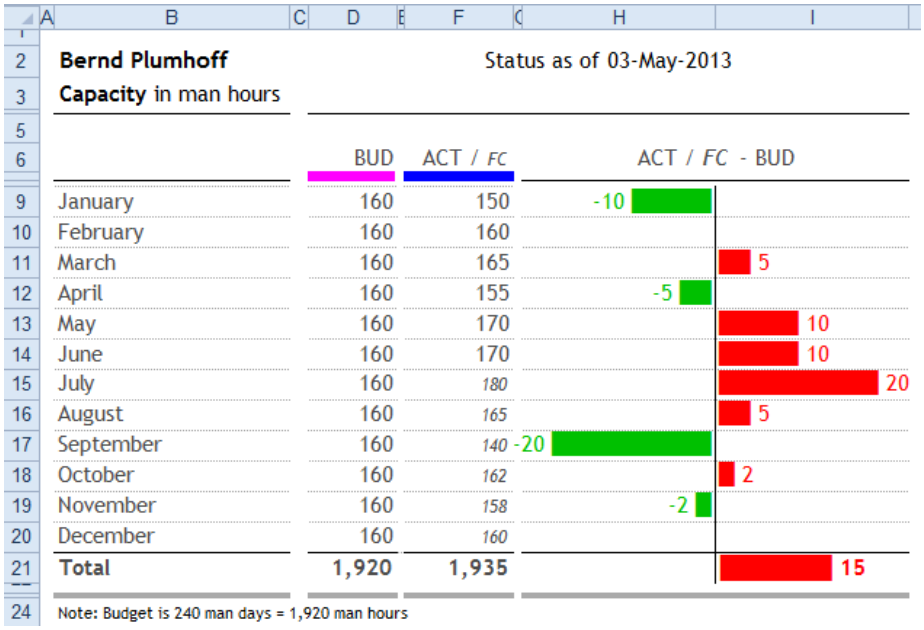
```
A3: =Bookvalue_Start
B3: =IF(ABS(@Total_Average-MAX(Inner_Values)) >= ABS(@Total_Average - MIN(Inner_Values)), IF(B$2 =
MAX(Inner_Values), B$2, (@Total_Average * @Total_Count-Bookvalue_Start-Bookvalue_End - MAX(Inner_Values) *
COUNTIF(Inner_Values, MAX(Inner_Values))) / (@Inner_Count - COUNTIF(Inner_Values, MAX(Inner_Values)))),
IF(B$2 = MIN(Inner_Values), B$2, (@Total_Average * @Total_Count - Bookvalue_Start - Bookvalue_End -
MIN(Inner_Values) * COUNTIF(Inner_Values, MIN(Inner_Values))) / (@Inner_Count - COUNTIF(Inner_Values,
MIN(Inner_Values))))) 
```
and copy to the right until cell K3

```
L3: =Bookvalue_End
```

Cell Based Charts

Dietmar P. came up with a nice cell based chart at Controller Akademie. I applied some changes and enhancements like introducing auto-scaling and adding two parameters:

- A number format to allow for a general format change if necessary.
- A Boolean parameter to control whether to switch the green and red colouring of the chart. If you hire a contractor then you might like to show staying below the budget with a green colour, but if you like to present sales figures then it might be the other way round



	A	B	C	D	E	F	G	H	I
2		**Bernd Plumhoff**				Status as of 03-May-2013			
3		**Capacity** in man hours							
5									
6				BUD		ACT / *FC*		ACT / *FC* - BUD	
9		January		160		150		-10	
10		February		160		160			
11		March		160		165			5
12		April		160		155		-5	
13		May		160		170			10
14		June		160		170			10
15		July		160		*180*			20
16		August		160		*165*			5
17		September		160		*140*		-20	
18		October		160		*162*			2
19		November		160		*158*		-2	
20		December		160		*160*			
21		**Total**		**1,920**		**1,935**			**15**
24		Note: Budget is 240 man days = 1,920 man hours							

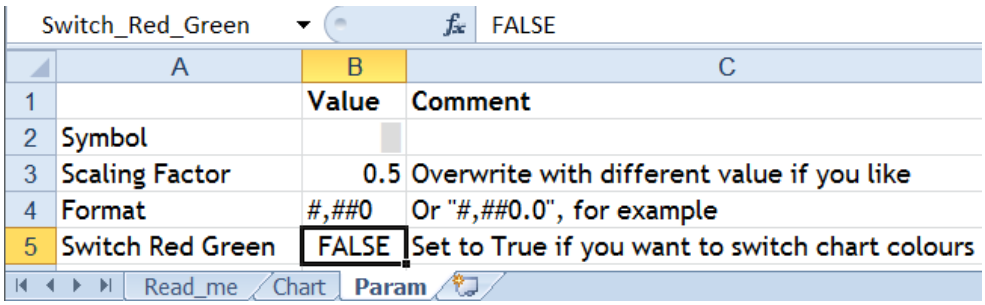
Formula in H9:

```
=REPT(TEXT(F9-D9,Format)&" ",F9-D9<0)&REPT(Symbol,(F9-D9<0)*ABS(ROUND(F9-D9,0))*Scaling_Factor)
```

In I9:

```
=REPT(Symbol,(F9-D9>0)*ABS(ROUND(F9-D9,0))*Scaling_Factor)&REPT(" "&TEXT(F9-D9,Format),F9-D9>0)
```

In sheet Param all parameters have been collected:



Switch_Red_Green | fx FALSE

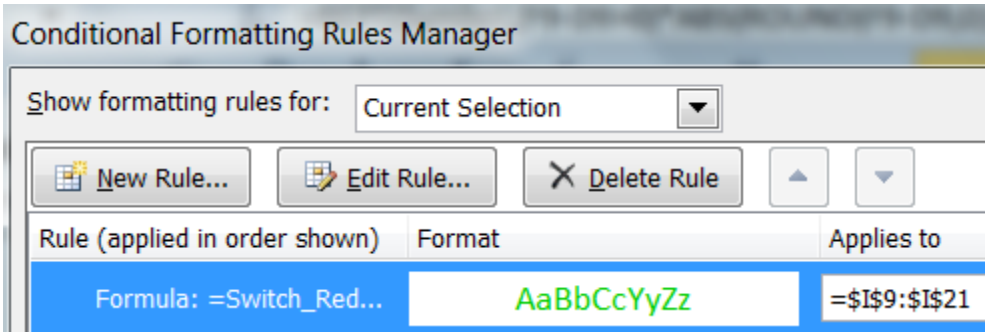
	A	B	C
1		**Value**	**Comment**
2	Symbol		
3	Scaling Factor	0.5	Overwrite with different value if you like
4	Format	#,##0	Or "#,##0.0", for example
5	Switch Red Green	FALSE	Set to True if you want to switch chart colours

Read_me | Chart | Param

These parameters have been named as shown in the Name Manager window:

Name	Value	Refers To	Scope
Format	#,##0	=Param!$B$4	Workbook
Scaling_Factor	0.5	=Param!$B$3	Workbook
Switch_Red_Green	FALSE	=Param!$B$5	Workbook
Symbol		=Param!$B$2	Workbook

A possible switch between colours green and red is implemented via a conditional format, for example in column I:



Simple VBA Programs

Abstract

With simple VBA programs you can create solutions for missing Excel functionalities.

Sum up Numbers with same Number Format – *sbSumMyFormat*

If you have an area filled with numbers of different currencies (i.e. different number formats) and if you need to calculate totals for all of these:

B5 =sbSumMyFormat(Area)

	A	B	C	D
1	Area:	200.00 €	£20.00	-$6.00
2		£10.00	-300.00 €	500.00 €
3		$4.00	$9.00	-£55.00
4		Euros	Pounds Sterling	US Dollars
5	Totals:	400.00 €	-£25.00	$7.00

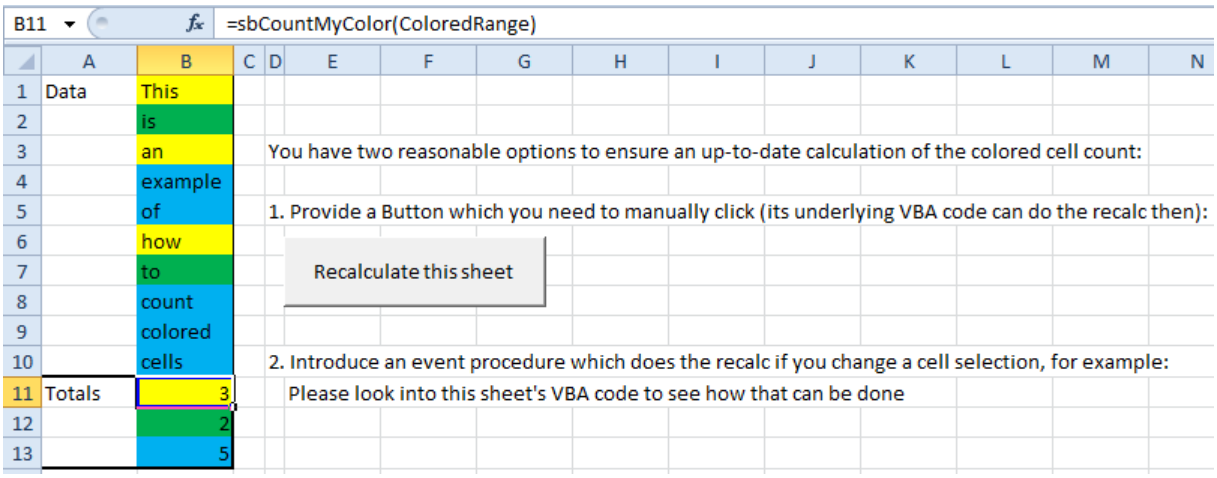
Please note that changing the number format or the color of a cell does not trigger its recalculation (nor of dependent cells). You would need to invoke a recalculation manually with CTRL + ALT + F9, for example. Other reasonable options to ensure an up-to-date calculation of dependent cells are a manual button which starts the recalculation, or an event procedure which starts this whenever a different cell is being selected, for example.

sbSumMyFormat Program Code

```
Function sbSumMyFormat(r As Range)
'Sums up all values in r which have the same number format
'as calling cell (where this function is called from).
'Please keep in mind that this function is NOT automatically
'updated if an input cell format changes because that does
'not trigger a recalculation event! To ensure an up-to-date
'calculation you need to do this yourself
'(with CTRL + ALT + F9, for example).
'(C) (P) by Bernd Plumhoff 10-Jul-2011 PB V0.21
Dim v

For Each v In r
    If v.NumberFormat = Application.Caller.NumberFormat Then
        sbSumMyFormat = sbSumMyFormat + v
    End If
Next v

End Function
```

Count Cells with Certain Color – *sbCountMyColor*

With this VBA code you can count colored cells. Please notice the same restrictions as in previous chapter regarding recalculation. In this example cells B1:B10 are named ColoredRange:



sbCountMyColor Program Code

```
Function sbCountMyColor(r As Range)
'Counts all cells of r which have the same color as calling
'cell (where this function is called from).
'Please keep in mind that this function is NOT automatically
'updated if an input cell format changes because that does
'not trigger a recalculation event! To ensure an up-to-date
'calculation you need to do this yourself
'(with CTRL + ALT + F9, for example).
'(C) (P) by Bernd Plumhoff 10-Jul-2011 PB V0.11
Dim v

For Each v In r
    If v.Interior.Color = Application.Caller.Interior.Color Then
        sbCountMyColor = sbCountMyColor + 1
    End If
Next v

End Function
```

This Sub would need to be inserted into the code section of the related worksheet:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    Dim state As SystemState
    Set state = New SystemState 'Please look into class module
                 'SystemState: Events are being disabled here
    Call RecalculateColoredRangeDependents
End Sub

Sub RecalculateColoredRangeDependents()
    Range("ColoredRange").DirectDependents.Calculate
End Sub
```

Allocate Assets to a Portfolio – *sbAllocate*

If you need to generate random portfolios with a given total sum and lower boundaries and upper boundaries for each asset, you can take the function *sbAllocate* shown below.

	A	B	C	D
1			**Budget:**	
2			**11.000,00**	
3		Lower Boundaries	Result Vector	Upper Boundaries
4	Asset 1	700,00	1.299,84	1.300,00
5	Asset 2	1.300,00	1.399,96	1.400,00
6	Asset 3	-1.000,00	-369,29	1.000,00
7	Asset 4	-2.000,00	-1.074,74	-1.000,00
8	Asset 5	0,00	1.708,16	5.000,00
9	Asset 6	-300,00	-26,05	0,00
10	Asset 7	1.700,00	8.062,13	9.000,00
11	Sum	400,00	11.000,00	16.700,00

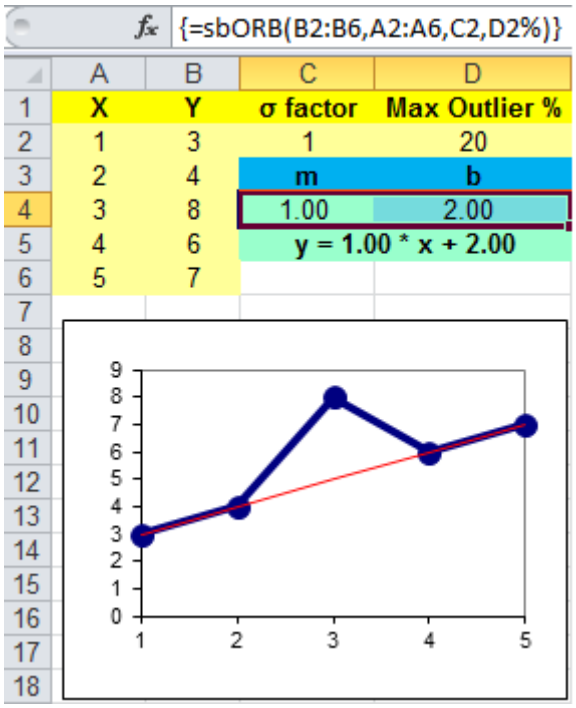
Worksheet Formulas		
Range	Formula	
C4	**C4**	=sbAllocate(C2,B4:B10,D4:D10)

sbAllocate Program Code

```
Function sbAllocate(db As Double, _
    vlb As Variant, _
    vub As Variant) As Variant
'Generate a portfolio of assets x1..xN
'x1..xN being random numbers (double) with:
'x1+x2+..xN = db 'budget
'xi >= vlb(v)     'lower bound vector
'xi <= vub(v)     'upper bound vector
'(C) (P) by Bernd Plumhoff 22-Feb-2025 PB V0.2
Dim dcumx     As Double
Dim dcumlb    As Double
Dim dcumub    As Double
Dim dxlb      As Double
Dim dxub      As Double
Dim n         As Long
Dim v         As Variant
Dim vR        As Variant

With Application.WorksheetFunction
dcumlb = .Sum(vlb)
dcumub = .Sum(vub)
If dcumlb > db Or dcumub < db Then
  sbAllocate = CVErr(xlErrValue)
  Exit Function
End If
n = vlb.Count
vR = .Transpose(vlb)
dcumx = 0#
For Each v In UniqRandInt(n, n)
  If vlb(v) > vub(v) Then
    sbAllocate = CVErr(xlErrValue)
    Exit Function
  End If
  dcumlb = dcumlb - vlb(v)
  dcumub = dcumub - vub(v)
  dxlb = .Max(vlb(v), db - dcumx - dcumub)
  dxub = .Min(vub(v), db - dcumx - dcumlb)
  vR(v) = dxlb + Rnd() * (dxub - dxlb)
  dcumx = dcumx + vR(v)
Next v
sbAllocate = .Transpose(vR)
End With
End Function
```

Eliminate Outliers – sbORB

Real life data sometimes includes extreme (outlier) values which you might want to ignore or to exclude. This way you cab create an outlier resistant beta, for example:



sbORB Program Code

```
Function sbORB(rY As Range, rX As Range, _
    Optional dSigmaFactor As Double = 3#, _
    Optional dMaxOutlierPercentage As Double = 0.5) As Variant
'sbORB() = outlier resistant beta returns a beta and
'an alpha where y = beta * x + alpha is most accurate
'for (almost) all given x in rX and y in rY.
'"Almost" means that we successively (one by one) throw out outliers
'which have a distance of > dSigmaFactor * STDEV_of_all_Distances
'from the least square (LS) proxy.
'(C) (P) by Bernd Plumhoff 24-Jun-2007 PB V0.9
Dim vLinEst As Variant 'store LinEst() result of recent LS proxy during iterations
Dim dm2 As Double 'ortogonal slope to recent LS proxy
Dim dc As Double 'Constant c in: y2=m2*x2+c which is ortogonal to LS proxy through a given point
Dim dx2 As Double 'x2 in: y2 = m2 * x2 + c which is ortogonal to LS proxy through a given point
Dim dy2 As Double 'y2 in: y2 = m2 * x2 + c which is ortogonal to LS proxy through a given point
Dim i As Long, j As Long
Dim lcount As Long 'holds current number of live points
Dim lcount_orig As Long 'original (starting) number of points
Dim lcount_old As Long 'holds number of live points of previous iteration
Dim daverage As Double 'average of distances to LS proxy of current iterations' live points
Dim dstdev As Double 'Stdev of distances to LS proxy of current iterations' live points
Dim dDistMax As Double
Dim lDistMaxIdx As Long

lcount = rX.Rows.Count
If rX.Columns.Count > lcount Then
    lcount = rX.Columns.Count
End If
lcount_orig = lcount
lcount_old = lcount + 1

ReDim dDist(1 To lcount) As Double 'store distances of live points to recent LS proxy (line)
ReDim dX(1 To lcount) As Double
ReDim dY(1 To lcount) As Double    'store coordinates of "live" points during iterations

'read data row-wise or column-wise
If rX.Rows.Count > rX.Columns.Count Then
    For i = 1 To lcount
        dX(i) = rX.Cells(i, 1)
```

```
        dY(i) = rY.Cells(i, 1)
    Next i
Else
    For i = 1 To lcount
        dX(i) = rX.Cells(1, i)
        dY(i) = rY.Cells(1, i)
    Next i
End If

Do
    lcount_old = lcount
    ReDim Preserve dDist(1 To lcount) As Double 'Store distances of live points to last LS proxy
    ReDim Preserve dX(1 To lcount) As Double
    ReDim Preserve dY(1 To lcount) As Double 'Store coordinates of "live" points during iterations
    vLinEst = Application.WorksheetFunction.LinEst(dY, dX, True, True)
    dDistMax = 0#
    lDistMaxIdx = 1
    For i = 1 To lcount
        'Calculate distances of live points to recent LS proxy
        dm2 = -1# / vLinEst(1, 1)
        dc = dY(i) - dX(i) * dm2
        dx2 = (dc - vLinEst(1, 2)) / (vLinEst(1, 1) - dm2)
        dy2 = dm2 * dx2 + dc
        dDist(i) = Sqr((dX(i) - dx2) * (dX(i) - dx2) + (dY(i) - dy2) * (dY(i) - dy2))
        'remember largest distance and its index
        If dDist(i) > dDistMax Then
            dDistMax = dDist(i)
            lDistMaxIdx = i
        End If
    Next i
    'calculate average and standard deviation of live points' distances to LS proxy
    daverage = Application.WorksheetFunction.Average(dDist)
    dstdev = Application.WorksheetFunction.StDev(dDist)
'    'kill points with distance > dSigmaFactor * dstdev 'Attention: might erase too many points
'    j = 1
'    For i = 1 To lcount
'        If dDist(i) <= dstdev * dSigmaFactor Then
'            dX(j) = dX(i)
'            dY(j) = dY(i)
'            j = j + 1
'        Else
'            Debug.Print "Lcount: " & lcount & ". Throwing out (" & dX(i) & ";" & dY(i) & ")"
'        End If
'    Next i
'    lcount = j - 1
    'kill point with largest distance > dSigmaFactor * dstdev
    If dDist(lDistMaxIdx) >= dstdev * dSigmaFactor Then
        Debug.Print "Lcount: " & lcount & ". Throwing out (" & dX(lDistMaxIdx) & _
                    ";" & dY(lDistMaxIdx) & ")"
        dX(lDistMaxIdx) = dX(lcount)
        dY(lDistMaxIdx) = dY(lcount)
        lcount = lcount - 1
    End If
Loop While lcount_old > lcount And lcount / lcount_orig > 1# - dMaxOutlierPercentage

If lcount < lcount_old Then
    vLinEst = Application.WorksheetFunction.LinEst(dY, dX, True, True)
End If

sbORB = vLinEst

End Function
```

Fair Distribution of a Limited Budget – sbDistBudget

"Givers have to set limits because takers rarely do." [Irma Kurtz]

Suppose your company is in the middle of its annual process to plan its revenues and expenses. You are head of a division with 6 departments (A to F). Your department heads requested 2,000, 1,900, 2,000, 2,000, 600 and 2,000 € but you only got a budget total of 9,500 € from your company. Your departments' weighted contributions to your company's revenues are 30%, 20%, 15%, 15%, 10% and 10%.

How do you distribute your budget? You surely would not give them more than they have asked for, and if the you cannot give everybody what they asked you want to take their revenue contribution into account:

C5 =sbDistBudget(B4;C2:H2;C3:H3)

	A	B	C	D	E	F	G	H
1		Total	A	B	C	D	E	F
2	Request	10.500,00	2.000,00	1.900,00	2.000,00	2.000,00	600,00	2.000,00
3	Weight%	100,0%	30,0%	20,0%	15,0%	15,0%	10,0%	10,0%
4	Budget weighted	9.500,00	2.850,00	1.900,00	1.425,00	1.425,00	950,00	950,00
5	Budget abs. (Macro)	9.500,00	2.000,00	1.900,00	1.875,00	1.875,00	600,00	1.250,00
6	Budget rel.	100,0%	21,1%	20,0%	19,7%	19,7%	6,3%	13,2%

sbDistBudget Program Code

```
Function sbDistBudget(dBudget As Double, _
  vRequest As Variant, _
  vWeight As Variant) As Variant
'Distribute a budget fairly upon Ubound(vRequest)
'requestors according to their weight vWeight(i)
'but do not give them more than they requested.
'Iterative solution.
'(C) (P) by Bernd Plumhoff 03-Dec-2012 PB V0.22
Dim dSumRequest As Double
Dim dSumWeight As Double
Dim dSumDist As Double
Dim dBudgetRest As Double
Dim dMinRest As Double
Dim i As Long, lc As Long, lgtNull As Long
With Application
lc = vRequest.Count
If lc <> vWeight.Count Then
  sbDistBudget = CVErr(xlErrValue)
  Exit Function
End If
ReDim dWeight(1 To lc) As Double
ReDim vR(1 To lc) As Variant 'Result vector
ReDim vT(1 To lc) As Variant 'Temp vector
dSumRequest = .Sum(vRequest)
If dSumRequest <= dBudget Then
  'Easy case: budget >= requests
  For i = 1 To lc
    vR(i) = vRequest(i)
  Next i
  sbDistBudget = vR
  Exit Function
End If
'Initialize budget distribution
dBudgetRest = dBudget
For i = 1 To lc
  dWeight(i) = vWeight(i)
Next i
'Distribute budget
```

```
Do While dBudget > dSumDist
  dSumWeight = .Sum(dWeight)
  If dSumWeight > 0# Then
    For i = 1 To lc
      vT(i) = dWeight(i) * dBudgetRest / dSumWeight
      If vT(i) + vR(i) >= vRequest(i) Then
        vT(i) = vRequest(i) - vR(i)
        dWeight(i) = 0#
      End If
      vR(i) = vR(i) + vT(i)
    Next i
  Else
    lgtNull = 0
    dMinRest = dBudgetRest
    For i = 1 To lc
      vT(i) = .Max(vRequest(i) - vR(i), 0#)
      If vT(i) > 0# Then
        lgtNull = lgtNull + 1
        If dMinRest > vT(i) Then
          dMinRest = vT(i)
        End If
      End If
    Next i
    If lgtNull = 0 Then Exit Do
    If dMinRest > dBudgetRest / lgtNull Then
      dMinRest = dBudgetRest / lgtNull
    End If
    For i = 1 To lc
      If vT(i) > 0# Then
        vR(i) = vR(i) + dMinRest
        vT(i) = dMinRest
      End If
    Next i
  End If
  dBudgetRest = dBudgetRest - .Sum(vT)
  dSumDist = .Sum(vR)
Loop
End With
sbDistBudget = vR
End Function
```

Compute Collatz Length – sbCollatz

"Mathematics is not yet ready for such problems." [Paul Erdös]

The Collatz conjecture states for positive integers:

- If the number is even, halve it.
- If the number is odd, triple it and add one.

If you apply above rules repetitively, you will always reach 1.

Example: If you start with 5, you will get 5, 16, 8, 4, 2, 1 which results in the Collatz length of 6.

sbCollatz Program Code

```
Function sbCollatz(s As String) As Long
'Calculates the Collatz length of a positive integer =
'returns count of iterations until result is 1.
'Excel is not the best tool to implement this but here we are:
'(C) (P) by Bernd Plumhoff 17-Jul-2022 PB V0.2
Dim b As Boolean, c As Integer
Dim i As Long, j As Long, k As Long, m As Long, n As Long, p As Long
n = Len(s)
m = n + 20 'We assume 20 additional digits will suffice
ReDim t(1 To m) As Integer
For i = 1 To n
    t(m - n + i) = Mid(s, i, 1)
Next i
b = False
For j = 1 To m - 1
  If t(j) <> 0 Then Exit For
Next j
k = 1
If j = m And t(m) < 2 Then
  t(m) = 1
  b = True
End If
Do While Not b
  k = k + 1
  Select Case t(m)
  Case 0, 2, 4, 6, 8
    'Divide by 2
    c = 0
    For j = 1 To m
      p = 5 * (t(j) Mod 2)
      t(j) = t(j) \ 2 + c
      c = p
    Next j
  Case 1, 3, 5, 7, 9
    'Multiply by 3 and add 1
    c = 1
    For j = m To 1 Step -1
      p = 3 * t(j) + c
      t(j) = p Mod 10
      c = p \ 10
    Next j
    Debug.Assert c = 0 'If we fail here the number of additional digits was too small
  Case Else
    Debug.Assert False
  End Select
  For j = 1 To m - 1
    If t(j) <> 0 Then Exit For
  Next j
  If j = m And t(m) = 1 Then b = True
  'If you like you can print out t() here
Loop
sbCollatz = k
End Function
```

Rank Item Uniquely – sbUniqRank

You need a rank function which returns a unique rank, even if duplicates occur? One possible approach:

You need to press ALT + F11, insert a new module and copy the program code below into the new module, then return to your spreadsheet, select cells A8:C11 and enter `=sbUniqRank($A$2:$C$5,D13,FALSE,1)`.

Worksheet Formulas		
Range	Formula	
A8	**A8**	=RANK(A2:C5,A2:C5)
A14,A20,A26, A32,A38,A44, A50,A56	**A14**	=sbUniqRank($A$2:$C$5, D13,FALSE,1)

	A	B	C	D
1	Input			
2	4	5	6	
3	5	4	6	
4	4	4	6	
5	1	2	3	
6				
7	Normal Rank			
8	6	4	1	
9	4	6	1	
10	6	6	1	
11	12	11	10	
12				
13	sbUniqRank			1
14	4	8	10	
15	9	5	11	
16	6	7	12	
17	1	2	3	
18				
19	sbUniqRank			2
20	4	8	10	
21	9	5	11	
22	7	6	12	
23	1	2	3	
24				
25	sbUniqRank			3
26	7	9	12	
27	8	6	11	
28	4	5	10	
29	1	2	3	
30				
31	sbUniqRank			4
32	7	9	12	
33	8	6	11	
34	5	4	10	
35	1	2	3	
36				
37	sbUniqRank			5
38	4	9	10	
39	8	6	11	
40	5	7	12	
41	1	2	3	
42				
43	sbUniqRank			6
44	5	9	12	
45	8	7	11	
46	4	6	10	
47	1	2	3	
48				
49	sbUniqRank			7
50	6	8	10	
51	9	4	11	
52	7	5	12	
53	1	2	3	
54				
55	sbUniqRank			8
56	7	8	12	
57	9	5	11	
58	6	4	10	
59	1	2	3	

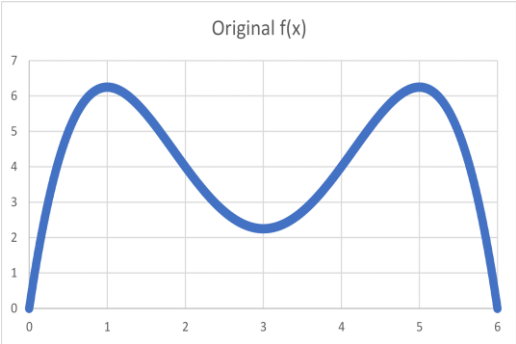
sbUniqRank Program Code

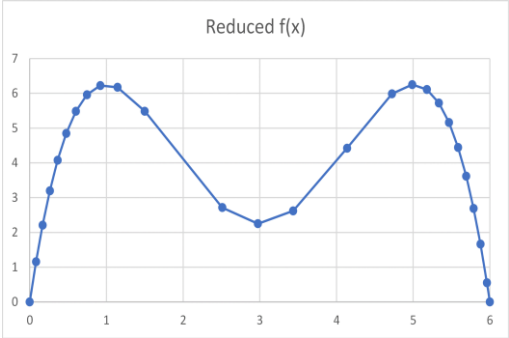
```
Function sbUniqRank(r As Range, _
    Optional vCountFrom As Variant = 1, _
    Optional bJustNumeric As Boolean = True, _
    Optional lOrder As Long = 0) As Variant
'Array function to rank a range with unique ranks.
'vCountFrom determines from where you count in case of duplicates:
'1 = first rows (1 to count), then columns (1 to count), i. e. top left to top right (tltr)
'2 = starting with top right to top left, then downwards (trtl)
'...
'8 = starting with bottom right to top right, then to the left (brtr)
'If bJustNumeric is True then Rank will be used to rank, if False then Countif will be used.
'lOrder is like Rank's order: 0 = Descending, 1 = Ascending
'(C) (P) by Bernd Plumhoff 25-Oct-2018 PB V0.6
Dim obj As Object
Dim bSwap As Boolean
Dim i As Long, i1 As Long, i2 As Long, i3 As Long
Dim j As Long, j1 As Long, j2 As Long, j3 As Long
Dim sComp As String
Dim vI As Variant, vR As Variant
vI = r: vR = vI
Set obj = CreateObject("Scripting.Dictionary")
Select Case vCountFrom
  Case 1, "tltr", "olor"
    i1 = 1: i2 = UBound(vI, 1): i3 = 1: j1 = 1: j2 = UBound(vI, 2): j3 = 1: bSwap = False
  Case 2, "trtl", "orol"
    i1 = 1: i2 = UBound(vI, 1): i3 = 1: j1 = UBound(vI, 2): j2 = 1: j3 = -1: bSwap = False
  Case 3, "blbr", "ulur"
    i1 = UBound(vI, 1): i2 = 1: i3 = -1: j1 = 1: j2 = UBound(vI, 2): j3 = 1: bSwap = False
  Case 4, "brbl", "urul"
    i1 = UBound(vI, 1): i2 = 1: i3 = -1: j1 = UBound(vI, 2): j2 = 1: j3 = -1: bSwap = False
  Case 5, "tlbl", "olul"
    i1 = 1: i2 = UBound(vI, 2): i3 = 1: j1 = 1: j2 = UBound(vI, 1): j3 = 1: bSwap = True
  Case 6, "bltl", "ulol"
    i1 = 1: i2 = UBound(vI, 2): i3 = 1: j1 = UBound(vI, 1): j2 = 1: j3 = -1: bSwap = True
  Case 7, "trbr", "orur"
    i1 = UBound(vI, 2): i2 = 1: i3 = -1: j1 = 1: j2 = UBound(vI, 1): j3 = 1: bSwap = True
  Case 8, "brtr", "uror"
    i1 = UBound(vI, 2): i2 = 1: i3 = -1: j1 = UBound(vI, 1): j2 = 1: j3 = -1: bSwap = True
  Case Else
    sbUniqRank = CVErr(xlErrValue)
    Exit Function
End Select
sComp = ">": If lOrder = 1 Then sComp = "<"
If bSwap Then
  'column - wise
  For i = i1 To i2 Step i3
    For j = j1 To j2 Step j3
      If bJustNumeric Then
        vR(j, i) = Application.WorksheetFunction.Rank(vI(j, i), r, lOrder) _
                   + obj.Item(vI(j, i))
      Else
        vR(j, i) = Application.WorksheetFunction.CountIf(r, _
                   sComp & vI(j, i)) + obj.Item(vI(j, i)) + 1
      End If
      obj.Item(vI(j, i)) = obj.Item(vI(j, i)) + 1
    Next j
  Next i
Else
  'row - wise
  For i = i1 To i2 Step i3
    For j = j1 To j2 Step j3
      If bJustNumeric Then
        vR(i, j) = Application.WorksheetFunction.Rank(vI(i, j), r, lOrder) _
                   + obj.Item(vI(i, j))
      Else
        vR(i, j) = Application.WorksheetFunction.CountIf(r, _
                   sComp & vI(i, j)) + obj.Item(vI(i, j)) + 1
      End If
      obj.Item(vI(i, j)) = obj.Item(vI(i, j)) + 1
    Next j
  Next i
End If
sbUniqRank = vR
End Function
```

Eliminate Points of a Graph with Small Slope Changes – sbReducePoints

If you need to reduce the number of curve points to those only showing a significant slope change:

	A	B	C	D	E
1	**Maximal slope delta:**		**0,9**		
2					
3	**Original Curve**			**Reduced Curve**	
4	**x**	**Original f(x)**		**x**	**Reduced f(x)**
5	0	0		0	0
6	0,002	0,029954024		0,082	1,154316801
7	0,004	0,059816192		0,168	2,209449747
8	0,006	0,089586648		0,262	3,193370184
9	0,008	0,119265535		0,364	4,076592834
10	0,01	0,148852997		0,476	4,845092357
11	0,012	0,178349179		0,602	5,484021459
12	0,014	0,207754222		0,746	5,958121293
13	0,016	0,237068272		0,92	6,22336576
14	0,018	0,26629147		1,144	6,172920473
15	0,02	0,29542396		1,5	5,484375
16	0,022	0,324465885		2,51	2,715787998
17	0,024	0,353417389		2,976	2,251151917
18	0,026	0,382278614		3,436	2,621157878
19	0,028	0,411049702		4,138	4,42080326
20	0,03	0,439730797		4,724	5,985894456
21	0,032	0,468322042		4,988	6,249427451
22	0,034	0,496823578		5,18	6,10847356
23	0,036	0,525235548		5,336	5,719363511
24	0,038	0,553558095		5,468	5,156904658
25	0,04	0,58179136		5,586	4,444475685
26	0,042	0,609935486		5,692	3,614468541
27	0,044	0,637990615		5,788	2,691223393
28	0,046	0,665956889		5,878	1,664226161
29	0,048	0,693834449		5,962	0,553558095
30	0,05	0,721623438		6	6,57963E-12
31	0,052	0,749323996			





The original graph of the function f(x) = -0.25*(A5-3)^4 + 2*(A5-3)^2 + 2.25 here contained 3,001 points. By calling *sbReducePoints*(A5:A3005, B5:B3005, 0,9) we could reduce this count to 26 points.

Worksheet Formulas		
Range	Formula	
A6:A3005	**A6**	=A5+0.002
B5:B3005	**B5**	=-0.25*(A5-3)^4 + 2*(A5-3)^2 + 2.25
D5	**D5**	=sbReducePoints(A5:A3005,B5:B3005,C1)

If this simple slope change approach is not successful, use the Ramer–Douglas–Peucker algorithm: https://en.wikipedia.org/wiki/Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm

sbReducePoints Program Code

```
Function sbReducePoints(rX As Range, rY As Range, _
    Optional dMaxSlopeDelta As Double = 0.001) As Variant
'sbReducePoints eliminates points from a given set
'in case the slopes between these points do not differ
'too much.
'(C) (P) by Bernd Plumhoff 29-Mar-2023 PB V0.1

Dim bNewSlope               As Boolean

Dim dSlope12                As Double
Dim dSlope13                As Double
Dim dSlope23                As Double

Dim i                       As Long
Dim k                       As Long
Dim lcount                  As Long

With Application.WorksheetFunction

lcount = rX.Rows.Count
If rX.Columns.Count > lcount Then
    lcount = rX.Columns.Count
End If

ReDim dX(1 To lcount) As Double
ReDim dY(1 To lcount) As Double

'read data row-wise or column-wise
If rX.Rows.Count > rX.Columns.Count Then
    For i = 1 To lcount
        dX(i) = rX.Cells(i, 1)
        dY(i) = rY.Cells(i, 1)
    Next i
Else
    For i = 1 To lcount
        dX(i) = rX.Cells(1, i)
        dY(i) = rY.Cells(1, i)
    Next i
End If

ReDim vR(1 To 2, 1 To lcount) As Variant

vR(1, 1) = dX(1)
vR(2, 1) = dY(1)
vR(1, 2) = dX(2)
vR(2, 2) = dY(2)
k = 2
bNewSlope = True
For i = 3 To lcount
    If bNewSlope Then dSlope12 = (vR(2, k) - vR(2, k - 1)) / (vR(1, k) - vR(1, k - 1))
    dSlope13 = (dY(i) - vR(2, k - 1)) / (dX(i) - vR(1, k - 1))
    dSlope23 = (dY(i) - vR(2, k)) / (dX(i) - vR(1, k))
    If Abs(dSlope13 - dSlope12) > dMaxSlopeDelta Or _
        Abs(dSlope13 - dSlope23) > dMaxSlopeDelta Then
        k = k + 1
        bNewSlope = True
    Else
        bNewSlope = False
    End If
    vR(1, k) = dX(i)
    vR(2, k) = dY(i)
Next i

ReDim Preserve vR(1 To 2, 1 To k) As Variant

If rX.Rows.Count > rX.Columns.Count Then
    sbReducePoints = .Transpose(vR)
Else
    sbReducePoints = vR
End If

End With

End Function
```

Birthday List – sbBirthdayList

"Nice to be here? At my age, it's nice to be anywhere." [George Burns]

You like to know when it's time for cake in your team? Then create a birthday list:

	A	B	C	D	E
1	George Washington	22. Feb	Month	#	(Day) Names
2	John Adams	30. Okt	Januar	4	(7) Millard Fillmore, (9) Richard Milhous Nixon, (29) William McKinley, (30) Franklin Delano Roosevelt
3	Thomas Jefferson	13. Apr	Februar	4	(6) Ronald Wilson Reagan, (9) William Henry Harrison, (12) Abraham Lincoln, (22) George Washington
4	James Madison	16. Mrz	März	4	(15) Andrew Jackson, (16) James Madison, (18) Grover Cleveland, (29) John Tyler
5	James Monroe	28. Apr	April	4	(13) Thomas Jefferson, (23) James Buchanan, (27) Ulysses S. Grant, (28) James Monroe
6	John Quincy Adams	11. Jul	Mai	2	(8) Harry S. Truman, (29) John Fitzgerald Kennedy
7	Andrew Jackson	15. Mrz	Juni	2	(12) George Herbert Walker Bush, (14) Donald John Trump
8	Martin Van Buren	05. Dez	Juli	4	(4) Calvin Coolidge, (6) George Walker Bush, (11) John Quincy Adams, (14) Gerald Rudolph Ford
9	William Henry Harrison	09. Feb	August	5	(4) Barack Hussein Obama, (10) Herbert Clark Hoover, (19) William Jefferson (Bill) Clinton, (20) Benjamin Harrison, (27) Lyndon Baines Johnson
10	John Tyler	29. Mrz	September	1	(15) William Howard Taft
11	James Knox Polk	02. Nov	Oktober	6	(1) James Earl (Jimmy) Carter, (4) Rutherford Birchard Hayes, (5) Chester A. Arthur, (14) Dwight David Eisenhower, (27) Theodore Roosevelt, (30) John Adams
12	Zachary Taylor	24. Nov	November	6	(2) James Knox Polk, Warren Gamaliel Harding, (19) James Abram Garfield, (20) Joseph Robinette Biden, (23) Franklin Pierce, (24) Zachary Taylor
13	Millard Fillmore	07. Jan	Dezember	3	(5) Martin Van Buren, (28) Woodrow Wilson, (29) Andrew Johnson
14	Franklin Pierce	23. Nov			
15	James Buchanan	23. Apr			
16	Abraham Lincoln	12. Feb			
43	Barack Hussein Obama	04. Aug			
44	Donald John Trump	14. Jun			
45	Joseph Robinette Biden	20. Nov			

Worksheet Formulas		
Range	Formula	
C1	**C1**	=sbBirthdayList(A1:B45)

sbBirthdayList Program Code

```
Function sbBirthdayList(r As Range) As Variant
'Create monthly birthday list.
'(C) (P) by Bernd Plumhoff 15-Sep-2010 PB V0.10
Dim vR(1 To 13, 1 To 3) As Variant
Dim i As Long, j As Long
Dim sNames(101 To 1231) As String

'Fill temporary array
For i = 1 To r.Rows.Count
  If IsDate(r.Cells(i, 2)) Then
    j = Month(r.Cells(i, 2))
    vR(j + 1, 2) = vR(j + 1, 2) + 1 'Increasing DOB counter for month
    j = j * 100 + Day(r.Cells(i, 2))
    If sNames(j) <> "" Then sNames(j) = sNames(j) & ", "
    sNames(j) = sNames(j) & r.Cells(i, 1)
  End If
Next i

'Fill output area
vR(1, 1) = "Month"
vR(1, 2) = "#"
vR(1, 3) = "(Day) Names"
For i = 1 To 12
  vR(i + 1, 1) = Format(DateSerial(1900, i, 1), "MMMM")
  vR(i + 1, 3) = ""
  For j = 1 To 31
    If sNames(i * 100 + j) <> "" Then
      If vR(i + 1, 3) <> "" Then vR(i + 1, 3) = vR(i + 1, 3) & ", "
      vR(i + 1, 3) = vR(i + 1, 3) & "(" & j & ") " & sNames(i * 100 + j)
    End If
  Next j
Next i

sbBirthdayList = vR

End Function
```

Accumulated Trade Blotter – sbAccumulatedTadeBlotter

If you like to create an accumulated trade blotter:

I3 =G3*F3

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1						Input Data						Output Data			
		Bought/	Order-			Position Size/		Conversion	Traded	Traded Value		Create Output Data		Position Size/	Traded Value
2	Currency	Sold	ID	Trade Date	Value Date	Amount	Price	Rate	Value	in EUR		Currency	Long/Short	Amount	in EUR
3	EURGBP	Sold	1	14/01/2011	17/01/2011	26,100	0.89	1.17137265	23,317.14	27,313.07		EURGBP	Enter short	26100	27,313.07
4	EURCHF	Sold	2	14/01/2011	17/01/2011	26,100	1.46	0.72345243	38,160.77	27,607.50		EURCHF	Enter short	26100	27,607.50
5	EURGBP	Bought	3	14/01/2011	17/01/2011	26,100	0.89	1.17045492	23,309.96	27,283.25		EURGBP	Exit short - flat	0	0.00
6	EURGBP	Sold	4	14/01/2011	17/01/2011	26,100	0.89	1.17137265	23,317.14	27,313.07		EURGBP	Enter short	26100	27,313.07
7	EURCHF	Bought	5	14/01/2011	17/01/2011	26,100	1.46	0.71429161	38,155.45	27,254.12		EURCHF	Exit short - flat	0	0.00
8	EURGBP	Bought	6	14/01/2011	17/01/2011	26,100	0.89	1.16996567	23,311.29	27,273.41		EURGBP	Exit short - flat	0	0.00
9	EURGBP	Sold	7	14/01/2011	17/01/2011	26,100	0.89	1.17137265	23,317.14	27,313.07		EURGBP	Enter short	26100	27,313.07
10	EURCHF	Bought	8	14/01/2011	17/01/2011	26,100	1.46	0.72345243	38,151.46	27,600.76		EURCHF	Enter long	26100	27,600.76
11	EURCHF	Sold	9	14/01/2011	17/01/2011	26,100	1.46	0.71446362	38,160.77	27,264.48		EURCHF	Exit long - flat	0	0.00
12	EURGBP	Bought	10	14/01/2011	17/01/2011	26,100	0.89	1.16996567	23,311.29	27,273.41		EURGBP	Exit short - flat	0	0.00
13	EURGBP	Sold	11	14/01/2011	17/01/2011	26,100	0.89	1.17137265	23,317.14	27,313.07		EURGBP	Enter short	26100	27,313.07
14	EURCHF	Bought	12	14/01/2011	17/01/2011	26,100	1.46	0.72345243	38,154.12	27,602.69		EURCHF	Enter long	26100	27,602.69
15	EURGBP	Bought	13	14/01/2011	17/01/2011	26,100	0.89	1.1738292	23,312.62	27,365.03		EURGBP	Exit short - flat	0	0.00
16	EURGBP	Bought	14	14/01/2011	17/01/2011	26,100	0.89	1.17137265	23,311.29	27,306.21		EURGBP	Enter long	26100	27,306.21
17	EURCHF	Bought	15	14/01/2011	17/01/2011	26,100	1.46	0.72345243	38,148.79	27,598.84		EURCHF	Enter long	52200	55,201.53
18	EURGBP	Sold	16	14/01/2011	17/01/2011	26,100	0.89	1.1740558	23,315.81	27,374.07		EURGBP	Exit long - flat	0	0.00
19	EURCHF	Sold	17	14/01/2011	17/01/2011	26,100	1.46	0.71241701	38,154.12	27,181.64		EURCHF	Exit long	26100	28,019.88
20	EURCHF	Sold	18	14/01/2011	17/01/2011	26,100	1.46	0.73571458	38,154.12	28,070.54		EURCHF	Exit long - flat	0	0.00

sbAccumulatedTradeBlotter Program Code

```
Enum trade_sheet_columns
  ts_ccy = 1
  ts_bought_sold
  ts_order_id
  ts_trade_date
  ts_value_date
  ts_amount
  ts_price
  ts_fxrate
  ts_traded_value
  ts_traded_value_EUR
  ts_EMPTY
  ts_output_ccy
  ts_output_long_short
  ts_output_amount_accumulated
  ts_output_traded_value_EUR
End Enum 'trade_sheet_columns

Private Enum xlCI 'Excel Color Index
: xlCIBlack = 1: xlCIWhite: xlCIRed: xlCIBrightGreen: xlCIBlue '1 - 5
: xlCIYellow: xlCIPink: xlCITurquoise: xlCIDarkRed: xlCIGreen '6 - 10
: xlCIDarkBlue: xlCIDarkYellow: xlCIViolet: xlCITeal: xlCIGray25 '11 - 15
: xlCIGray50: xlCIPeriwinkle: xlCIPlum: xlCIIvory: xlCILightTurquoise '16 - 20
: xlCIDarkPurple: xlCICoral: xlCIOceanBlue: xlCIIceBlue: xlCILightBrown '21 - 25
: xlCIMagenta2: xlCIYellow2: xlCICyan2: xlCIDarkPink: xlCIDarkBrown '26 - 30
: xlCIDarkTurquoise: xlCISeaBlue: xlCISkyBlue: xlCILightTurquoise2: xlCILightGreen '31 - 35
: xlCILightYellow: xlCIPaleBlue: xlCIRose: xlCILavender: xlCITan '36 - 40
: xlCILightBlue: xlCIAqua: xlCILime: xlCIGold: xlCILightOrange '41 - 45
: xlCIOrange: xlCIBlueGray: xlCIGray40: xlCIDarkTeal: xlCISeaGreen '46 - 50
: xlCIDarkGreen: xlCIGreenBrown: xlCIBrown: xlCIDarkPink2: xlCIIndigo '51 - 55
: xlCIGray80 '56
End Enum

Sub Calculate_Accumulated_Blotter()
'(C) (P) by Bernd Plumhoff 15-Jan-2011 PB V0.1
Dim lRow As Long
Dim lColorIdx As Long
Dim dSign As Double
Dim dSum As Double
Dim vColors As Variant
Dim oCcyPairAcc As Object    'Stores accumulated amounts of ccy pairs
Dim state As SystemState     'Runtime optimisation - see class SystemState

Set state = New SystemState 'Runtime optimisation - see class SystemState
Set oCcyPairAcc = CreateObject("Scripting.Dictionary")

vColors = Array(xlCIRed, xlCIBlue, xlCIBrown, xlCIDarkGreen, _
          xlCIDarkYellow, xlCIOrange, xlCIDarkTeal, xlCIPlum)
lColorIdx = 0
Sheets("CurrencyPairs").Select        'Sheet to work on
lRow = 3                              'Start row
```

```
Do While Not IsEmpty(Cells(lRow, ts_ccy))
  If lRow Mod 10 = 0 Then Application.StatusBar = _
     "Calculate_Accumulated_Blotter: Processing row " & lRow & " ..."
  Cells(lRow, ts_output_ccy) = Cells(lRow, ts_ccy)
  If oCcyPairAcc.Item(Cells(lRow, ts_ccy).Text & "|FontColor") = 0# Then
    oCcyPairAcc.Item(Cells(lRow, ts_ccy).Text & "|FontColor") = vColors(lColorIdx)
    lColorIdx = lColorIdx + 1
    If lColorIdx > UBound(vColors) Then
      Call MsgBox("Row " & lRow & ": Not enough colors defined", vbOKOnly, "Error")
      Exit Sub
    End If
  End If
  Select Case Cells(lRow, ts_bought_sold)
  Case "Bought"
    dSign = 1#
  Case "Sold"
    dSign = -1#
  Case Else
    Call MsgBox("Row " & lRow & ": Illegal Bought/Sold Keyword """ & _
      Cells(lRow, ts_bought_sold) & """ in column " & ts_bought_sold, vbOKOnly, "Error")
    Exit Sub             'Stop at first error
  End Select
  dSum = oCcyPairAcc.Item(Cells(lRow, ts_ccy).Text) + dSign * Cells(lRow, ts_amount)
  Select Case Sgn(dSum)
  Case 1#
    If dSign = 1# Then
      Cells(lRow, ts_output_long_short) = "Enter long"
    Else
      Cells(lRow, ts_output_long_short) = "Exit long"
    End If
    oCcyPairAcc.Item(Cells(lRow, ts_ccy).Text & "|EUR") = _
        oCcyPairAcc.Item(Cells(lRow, ts_ccy).Text & "|EUR") + _
        dSign * Cells(lRow, ts_fxrate) * Cells(lRow, ts_traded_value)
  Case 0#
    If dSign = 1# Then
      Cells(lRow, ts_output_long_short) = "Exit short - flat"
    Else
      Cells(lRow, ts_output_long_short) = "Exit long - flat"
    End If
    oCcyPairAcc.Item(Cells(lRow, ts_ccy).Text & "|EUR") = 0#
  Case -1#
    If dSign = 1# Then
      Cells(lRow, ts_output_long_short) = "Exit short"
    Else
      Cells(lRow, ts_output_long_short) = "Enter short"
    End If
    oCcyPairAcc.Item(Cells(lRow, ts_ccy).Text & "|EUR") = _
        oCcyPairAcc.Item(Cells(lRow, ts_ccy).Text & "|EUR") + _
        dSign * Cells(lRow, ts_fxrate) * Cells(lRow, ts_traded_value)
  End Select
  Cells(lRow, ts_output_amount_accumulated) = Abs(dSum)
  oCcyPairAcc.Item(Cells(lRow, ts_ccy).Text) = dSum
  Cells(lRow, ts_output_traded_value_EUR) = Abs(oCcyPairAcc.Item(Cells(lRow, ts_ccy).Text & "|EUR"))
  Range(Cells(lRow, ts_output_ccy), Cells(lRow, ts_output_traded_value_EUR)).Font.ColorIndex = _
    oCcyPairAcc.Item(Cells(lRow, ts_ccy).Text & "|FontColor")
  lRow = lRow + 1
Loop
Set oCcyPairAcc = Nothing
End Sub
```

Most Frequent Pairs – sbMostFrequentPairs

Given a matrix of some club members and when they attended a club evening, which member duo showed up most frequently at these club meetings?

The input in sheet wsPresent (codename, see chapter *Fair Staff Selection Based on Team Size – sbFairStaffSelection)*:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1		Attended											
2	Member	01. Jan	01. Feb	01. Mrz	01. Apr	01. Mai	01. Jun	01. Jul	01. Aug	01. Sep	01. Okt	01. Nov	01. Dez
3	Adam	x	x	x		x	x	x	x	x		x	
4	Ben		x	x	x	x		x	x		x	x	x
5	Conrad	x		x		x	x	x		x			x
6	Dora	x	x		x	x	x	x	x		x	x	
7	Eric	x	x	x	x	x	x	x	x	x		x	x
8	Fred	x	x	x	x	x	x	x			x		x
9	George	x	x	x	x				x	x		x	

The output in sheet wsPairs (codename):

	A	B	C
1	Rank	Duo	Frequency
2	1	Adam & Eric	9
3	2	Ben & Eric	8
4		Dora & Eric	8
5		Eric & Fred	8
6	5	Adam & Dora	7
7		Ben & Dora	7
8		Ben & Fred	7
9		Conrad & Eric	7
10		Dora & Fred	7
11		Eric & George	7
12	11	Adam & Ben	6
13		Adam & Conrad	6
14		Adam & Fred	6
15		Adam & George	6
16		Conrad & Fred	6
17	16	Ben & George	5
18		Dora & George	5
19	18	Ben & Conrad	4
20		Conrad & Dora	4
21		Fred & George	4
22	21	Conrad & George	3

The solution is based on a matrix multiplication.

sbMostFrequentPairs Program Code

This programm needs (uses) class SystemState (see page 17).

```
Sub sbMostFrequentPairs(vNames As Variant, _
    vInputArea As Variant, _
    rOutput As Range)
'(C) (P) by Bernd Plumhoff  09-Nov-2019 PB V1.0

Dim vT As Variant
Dim i As Long, j As Long, k As Long
Dim sName As String
Dim state As SystemState

Set state = New SystemState
```

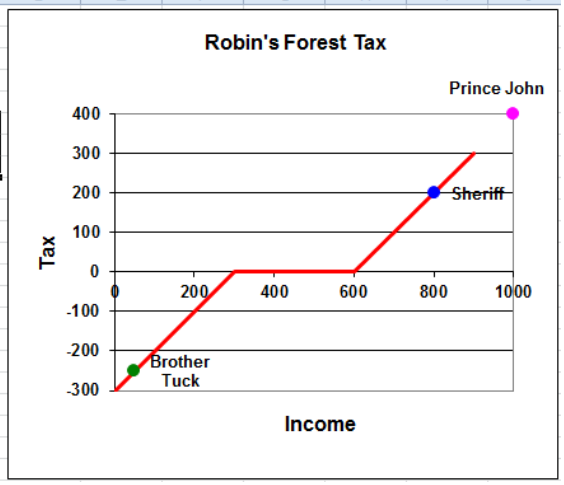
```
With Application.WorksheetFunction

'Get LBound und UBound address with vT(x, y):
vT = .Transpose(.Transpose(vInputArea))

For i = LBound(vT, 1) To UBound(vT, 1)
    For j = LBound(vT, 2) To UBound(vT, 2)
        If vT(i, j) = "x" Or vT(i, j) = "X" Then
            vT(i, j) = 1
        Else
            'Detect non-empty other cells:
            If vT(i, j) <> "" Then Debug.Print i, j, "'" & vT(i, j) & "'"
            vT(i, j) = 0
        End If
    Next j
Next i

vT = .MMult(vT, .Transpose(vT)) 'This is the core calculation

Range(rOutput, rOutput.Offset(0, 2)).FormulaArray = _
    Array("Rank", "Duo", "Frequency")
k = 1
For i = 2 To UBound(vT, 1)
    For j = 1 To i - 1 'We just need the lower left triangular matrix
        'Sort the names:
        sName = vNames(i) & " & " & vNames(j)
        If vNames(i) > vNames(j) Then sName = vNames(j) & " & " & vNames(i)
        Range(rOutput.Offset(k, 0), rOutput.Offset(k, 2)).FormulaArray = _
            Array("", sName, vT(i, j))
        k = k + 1
    Next j
Next i

'Sort by frequency and then by name
With rOutput.Worksheet.Sort
    .SortFields.Clear
    .SortFields.Add Key:=rOutput.Worksheet.Range("C2:C" & k), _
        Order:=xlDescending
    .SortFields.Add Key:=rOutput.Worksheet.Range("B2:B" & k), _
        Order:=xlAscending
    .SetRange rOutput.Worksheet.Range("A1:C" & k)
    .Header = xlYes
    .Apply
End With

'Add the rank
k = 1
Do While Not IsEmpty(rOutput.Offset(k, 2))
    If rOutput.Offset(k, 2) <> rOutput.Offset(k - 1, 2) Then
        'Some fancy top border
        With Range(rOutput.Offset(k, 0), _
            rOutput.Offset(k, 2)).Borders(xlEdgeTop)
            .LineStyle = xlContinuous
            .Weight = xlThin
        End With
        rOutput.Offset(k, 0) = k
    End If
    k = k + 1
Loop

Range(rOutput.Offset(k, 0), _
    rOutput.Offset(k, 2)).EntireColumn.AutoFit

End With

End Sub

Sub sbGenerateListOfPairs()

Dim state As SystemState
Set state = New SystemState

wsPairs.Cells.EntireRow.Delete
Call sbMostFrequentPairs(Range(wsPresent.Range("A3"), _
    wsPresent.Range("A2").End(xlDown)), _
    Range(wsPresent.Range("A2").End(xlDown).Offset(-1), _
        wsPresent.Range("A2").End(xlToRight).Offset(0, -1)).Offset(1, 1), _
    wsPairs.Range("A1"))

End Sub
```

Interpolate – sbInterp

If you have a given set of known (x, y) points and you need to know y-parts of other given x parts then you need to interpolate. It is similar to filling gaps of a table:

C6 {=sbInterp(B2:B5,C2:C5,B6:B8)}

	A	B	C
1	Tax Table	Income	Tax
2		0	-300
3		300	0
4		600	0
5		900	300
6	Prince John	1000	400
7	Sheriff	800	200
8	Brother Tuck	50	-250



C10 {=sbInterp($A$2:$A$5,$B$2:$B$5,$A10:$A18,C7,C8,C9)}

	A	B	C	D	E	F	G	H	I
1	X	Y							
2	1	0							
3	2	1							
4	3	2							
5	4	3							
6									
7		sType:	Const	Linear	LinearInVariance				
8		bExtrapolate:	TRUE	TRUE	TRUE				
9	New X	sExtraType:	Const	Linear	LinearInVariance		Expected correct results:		
10	0	New Y:	0	-1	0		0	-1	0
11	1		0	0	0		0	0	0
12	1.5		0	0.5	0.707106781		0	0.5	0.707107
13	2		1	1	1		1	1	1
14	2.5		1	1.5	1.58113883		1	1.5	1.581139
15	3		2	2	2		2	2	2
16	3.5		2	2.5	2.549509757		2	2.5	2.54951
17	4		3	3	3		3	3	3
18	5		3	4	3.741657387		3	4	3.741657
19									
20		sType:	Const	Linear	LinearInVariance				
21		bExtrapolate:	FALSE	FALSE	FALSE				
22	New X	sExtraType:	Const	Linear	LinearInVariance		Expected correct results:		
23	0	New Y:	#NUM!	#NUM!	#NUM!		#NUM!	#NUM!	#NUM!
24	1		0	0	0		0	0	0
25	1.5		0	0.5	0.707106781		0	0.5	0.707107
26	2		1	1	1		1	1	1
27	2.5		1	1.5	1.58113883		1	1.5	1.581139
28	3		2	2	2		2	2	2
29	3.5		2	2.5	2.549509757		2	2.5	2.54951
30	4		3	3	3		3	3	3
31	5		#NUM!	#NUM!	#NUM!		#NUM!	#NUM!	#NUM!

sbInterp Program Code

```
Function sbInterp(vX As Variant, vY As Variant, _
           vT As Variant, _
           Optional ByVal sType As String = "Linear", _
           Optional bExtrapolate As Boolean = True, _
           Optional ByVal sExtraType As String) As Variant
'Interpolates y-values for target values vT with known
'y-values vY and known x-values vX with type sType.
'sType can be:
'Const or C
'Linear or L
'LinearInVariance or LIV
'Extrapolation will be done if bExtrapolate is TRUE.
'Extrapolation type sExtraType defaults to sType if empty.
'Values in vX must be in ascending order. #VALUE! error
'indicates illegal sType, #NUM! error indicates that
'extrapolation has been switched off and #N/A tells you
'that x-values are not given in increasing order, or
'y-value count differs from x-value count.
'(C) (P) by Bernd Plumhoff 25-Dec-2023 PB V0.7
Dim i As Long, iX As Long, iY As Long, iT As Long, k As Long
Dim vTk, vXi
Dim sT As String 'Type of inter- or extrapolation
Dim sEType As String 'Extrapolation type
With Application
On Error Resume Next
iX = vX.Count
iX = UBound(vX)
iY = vY.Count
iY = UBound(vY)
iT = vT.Count
iT = UBound(vT)
On Error GoTo 0
If iX <> iY Then
  sbInterp = CVErr(xlErrNA)
  Exit Function
End If
k = 0
ReDim vX1(1 To iX) As Variant, vY1(1 To iX) As Variant
For i = 1 To iX
  If vX(i) <> "" And vY(i) <> "" Then
    k = k + 1
    vX1(k) = vX(i)
    vY1(k) = vY(i)
  End If
Next i
iX = k
ReDim Preserve vX1(1 To iX) As Variant
ReDim Preserve vY1(1 To iX) As Variant
If iX < 2 Then
  sType = "Const"
  sExtraType = "Const"
Else
  For k = 2 To iX
    If vX1(k) <= vX1(k - 1) Then
      sbInterp = CVErr(xlErrNA)
      Exit Function
    End If
  Next k
End If
ReDim vR(1 To iT) As Variant
If sExtraType = "" Then
  sEType = sType 'Same as interpolation type
Else
  sEType = sExtraType
End If
For k = 1 To iT
  i = 0
  vTk = 0
  vXi = 0
  On Error Resume Next
  i = .Match(vT(k), vX1, 1)
  vTk = vT(k)
  vXi = vX1(i)
  On Error GoTo 0
  If Not bExtrapolate And _
    (i = 0 Or (i = iX And vTk <> vXi)) Then
    vR(k) = CVErr(xlErrNum)
  Else
    sT = sType 'Set to interpolation type
    If i = 0 Then
      i = 1
      sT = sEType 'Set to extrapolation type
    End If
    If i = iX Then
      i = i - 1
      If vTk <> vXi Then
```

```
        sT = sEType 'Set to extrapolation type
      End If
      If sT = "C" Or sT = "Const" Then i = i + 1
    End If
    Select Case sT
    Case "C", "Const"
      vR(k) = .Index(vY1, i)
    Case "L", "Linear"
      vR(k) = .Index(vY1, i) + (vTk - .Index(vX1, i)) _
              * (.Index(vY1, i + 1) - .Index(vY1, i)) _
              / (.Index(vX1, i + 1) - .Index(vX1, i))
    Case "LIV", "LinearInVariance"
      On Error Resume Next
      vR(k) = Sqr(.Index(vY1, i) ^ 2# + (vTk - .Index(vX1, i)) _
              * (.Index(vY1, i + 1) ^ 2# - .Index(vY1, i) ^ 2#) _
              / (.Index(vX1, i + 1) - .Index(vX1, i)))
      On Error GoTo 0
    Case Else
      sbInterp = CVErr(xlErrValue)
      Exit Function
    End Select
  End If
Next k
If TypeName(vT) = "Range" Then
  If vT.Rows.Count > vT.Columns.Count Then
    vR = .Transpose(vR)
  End If
ElseIf TypeName(.Caller) = "Range" Then
  If .Caller.Rows.Count > .Caller.Columns.Count Then
    vR = .Transpose(vR)
  End If
End If
sbInterp = vR
End With
End Function
```

Combinations with Subsets k of n

This algorithm generates all subsets of n with k elements. The Excel function COMBIN(n, k) shows the count of all combinations, but the algorithm presented here lists all combinations individually so that you can examine them if necessary. The program is fast – please compare to the formula solutions you can find at: https://techcommunity.microsoft.com/discussions/excelgeneral/efficient-approach-to-generate-list-of-combinations-with-no-repetition/3786730

Example

Input:

	A	B
1	n	5
2	k	3

Output:

	A	B	C	D	E
1	1	1	1	0	0
2	1	0	1	1	0
3	0	1	1	1	0
4	1	1	0	1	0
5	1	0	0	1	1
6	0	1	0	1	1
7	0	0	1	1	1
8	1	0	1	0	1
9	0	1	1	0	1
10	1	1	0	0	1

Literature

Reingold, Nievergelt, Deo: Combinatorial Algorithms, 1977, Algorithm 5.9, p. 186, ISBN 0-13-152447-X

An interesting link: https://techcommunity.microsoft.com/discussions/excelgeneral/efficient-approach-to-generate-list-of-combinations-with-no-repetition/3786730

Combinations with k subsets of n Program Code

```
Public r As Long 'Output row

'Generates all combinations of array n with subsets k.
'See Reingold, Nievergelt, Deo: Combinatorial Algorithms, 1977, Algorithm 5.9, p. 186, ISBN 0-13-152447-X
'Version 0.2 12-Jul-2024

Sub Combinations_with_k_subsets_of_n(n As Long, k As Long)
Dim i As Long, j As Long, m As Long, t As Long, u As Long

With Application.WorksheetFunction
wsO.Cells.ClearContents
ReDim g(1 To n + 1) As Long
ReDim tau(1 To n + 1) As Long
ReDim res(1 To .Combin(n, k), 1 To n) As Variant
r = 1
For j = 1 To k
  g(j) = 1
  tau(j) = j + 1
Next j
For j = k + 1 To n + 1
  g(j) = 0
  tau(j) = j + 1
Next j
t = k
tau(1) = k + 1
i = 0
Do While i <> n + 1
  'Call Visit(g) instead of the next 4 rows if you need to analyze g().
  For u = 1 To UBound(g) - 1
    res(r, u) = g(u)
  Next u
  r = r + 1
  i = tau(1)
  tau(1) = tau(i)
  tau(i) = i + 1
  If g(i) = 1 Then
    If t <> 0 Then
      g(t) = 1 - g(t)
    Else
      g(i - 1) = 1 - g(i - 1)
    End If
    t = t + 1
  Else
    If t <> 1 Then
      g(t - 1) = 1 - g(t - 1)
    Else
      g(i - 1) = 1 - g(i - 1)
    End If
    t = t - 1
  End If
  g(i) = 1 - g(i)
  If t = i - 1 Or t = 0 Then
    t = t + 1
  Else
    t = t - g(i - 1)
    tau(i - 1) = tau(1)
    If t = 0 Then
      tau(1) = i - 1
    Else
      tau(1) = t + 1
    End If
  End If
Loop
Range(wsO.Cells(1, 1), wsO.Cells(r - 1, n)) = res
End With
End Sub

Sub Visit(g As Variant)
'Print current permutation in immediate window and on sheet Output.
'You can analyze the permutation or do other things as well.
Dim i As Long
For i = 1 To UBound(g) - 1
  wsO.Cells(r, i) = g(i)
  Debug.Print g(i);
Next i
Debug.Print
r = r + 1
End Sub

Sub test()
Debug.Print Now
Call Combinations_with_k_subsets_of_n(5, 3)
Debug.Print Now
End Sub
```

Lookup Variants

"The hardest thing of all is to find a black cat in a dark room, especially if there is no cat." [Confucius]

Here I present some LOOKUP variants I found useful:

Lookup Program Codes

```
Function sbLookup(vLookupValue As Variant, _
            rTableArray As Range, _
            Optional ByVal lOccurrence As Long = 1, _
            Optional lColumnOffset As Long, _
            Optional lRowOffset As Long) As Variant
'Reverse("moc.LiborPlus.www") PB 09-May-2010 V0.10
'Looks up lOccurrence'th occurrence of vLookupValue in rTableArray
'and returns found cell offset by lRowOffset rows and lColumnOffset
'columns. If lOccurrence is negative the search is done bottom-up
'(i.e. -1 finds the last value, -2 last but one, etc.).

Dim i As Long
Dim rFound As Range
Dim iSearchDir As Integer

If lOccurrence >= 0 Then
    iSearchDir = xlNext
Else
    iSearchDir = xlPrevious
    lOccurrence = -lOccurrence
End If

With rTableArray
    If rTableArray.Cells(1, 1) = vLookupValue And lOccurrence = 1 Then
        sbLookup = .Cells(1, 1)(1, lColumnOffset + 1)
        Exit Function
    Else
        Set rFound = .Cells(1, 1)
        For i = 1 To lOccurrence
            Set rFound = rTableArray.Find(What:=vLookupValue, After:=rFound, _
                    LookIn:=xlValues, LookAt:=xlWhole, SearchOrder:=xlRows, _
                    SearchDirection:=iSearchDir)
        Next i
    End If
End With

sbLookup = rFound.Offset(lRowOffset, lColumnOffset)

End Function


Function sbClosest(dSearchVal As Double, _
    rLookupRange As Range, _
    Optional dLower As Double = 0#, _
    Optional dUpper As Double = 0#) As Variant
'Looks for the closest value to dSearchVal in
'rLookupRange which is greater or equal to dSearchVal
'+ dLower and less or equal to dSearchVal + dUpper.
'Returns that value and the address of it. xlErrNum
'indicates that no relevant data was found.
'Reverse("moc.LiborPlus.www") V0.10 16-Oct-2010 PB
Dim dMin As Double, v, vR(1 To 2)
dMin = 1E+308
For Each v In rLookupRange
    If (dLower = 0# And dUpper = 0#) Or _
        (v >= dSearchVal + dLower And _
        v <= dSearchVal + dUpper) Then
            If Abs(v - dSearchVal) < dMin Then
                vR(1) = v
                vR(2) = v.Address(False, False)
                dMin = Abs(v - dSearchVal)
            End If
    End If
Next v
If dMin = 1E+308 Then
    sbClosest = CVErr(xlErrNum)
Else
    sbClosest = vR
End If
End Function
```

```
Function sbLookupAddress(vLookupValue As Variant, _
            rTableArray As Range, _
            Optional ByVal lOccurrence As Long = 1, _
            Optional lColumnOffset As Long, _
            Optional lRowOffset As Long) As String
'Reverse("moc.LiborPlus.www") PB 26-Aug-2010 V0.10
'Looks up lOccurrence'th occurrence of vLookupValue in rTableArray and
'returns address of found cell offset by lRowOffset rows and lColumnOffset
'columns. If lOccurrence is negative the search is done bottom-up
'(i.e. -1 finds the last value, -2 last but one, etc.).

Dim i As Long
Dim rFound As Range, rLast As Range
Dim iSearchDir As Integer

If lOccurrence >= 0 Then
    iSearchDir = xlNext
Else
    iSearchDir = xlPrevious
    lOccurrence = -lOccurrence + 1
End If

With rTableArray
    If rTableArray.Cells(1, 1) = vLookupValue Then lOccurrence = lOccurrence - 1
    If lOccurrence = 0 Then
        sbLookupAddress = .Cells(1, 1)(1, lColumnOffset + 1).Address(False, False)
        Exit Function
    Else
        Set rFound = .Cells(1, 1)
        Set rFound = rTableArray.Find(What:=vLookupValue, After:=rFound, _
                LookIn:=xlValues, LookAt:=xlWhole, SearchOrder:=xlRows, _
                SearchDirection:=iSearchDir)
        Set rLast = rFound
        Do
            lOccurrence = lOccurrence - 1
            If lOccurrence = 0 Then
                sbLookupAddress = rFound.Offset(lRowOffset, _
                                  lColumnOffset).Address(False, False)
                Exit Function
            End If
            Set rFound = rTableArray.Find(What:=vLookupValue, After:=rFound, _
                    LookIn:=xlValues, LookAt:=xlWhole, SearchOrder:=xlRows, _
                    SearchDirection:=iSearchDir)
        Loop While rLast.Address <> rFound.Address
        sbLookupAddress = CVErr(xlErrValue)
    End If
End With

End Function


Function vlookupall(sSearch As String, rRange As Range, _
    Optional lLookupCol As Long = 2, Optional sDel As String = ",") As String
'Vlookupall searches in first column of rRange for sSearch and returns
'corresponding values of column lLookupCol if sSearch was found. All these
'lookup values are being concatenated, delimited by sDel and returned in
'one string. If lLookupCol is negative then rRange must not have more than
'one column.
'Reverse("moc.LiborPlus.www") PB 16-Sep-2010 V0.20
Dim i As Long, sTemp As String
If lLookupCol > rRange.Columns.Count Or sSearch = "" Or _
    (lLookupCol < 0 And rRange.Columns.Count > 1) Then
    vlookupall = CVErr(xlErrValue)
    Exit Function
End If
vlookupall = ""
For i = 1 To rRange.Rows.Count
    If rRange(i, 1).Text = sSearch Then
        If lLookupCol >= 0 Then
            vlookupall = vlookupall & sTemp & rRange(i, lLookupCol).Text
        Else
            vlookupall = vlookupall & sTemp & rRange(i).Offset(0, lLookupCol).Text
        End If
        sTemp = sDel
    End If
Next i
End Function
```

```
Function vlookupallarr(sSearch As String, rRange As Range, _
    Optional lLookupCol As Long = 2) As Variant
'Vlookupall searches in first column of rRange for sSearch and returns
'corresponding values of column lLookupCol if sSearch was found. All
'values looked up are being returned in a vertical array.
'If lLookupCol is negative then rRange must not have more than
'one column.
'Reverse("moc.LiborPlus.www") PB 12-Jul-2012 V0.10
Dim i As Long, j As Long
If lLookupCol > rRange.Columns.Count Or sSearch = "" Or _
    (lLookupCol < 0 And rRange.Columns.Count > 1) Then
    vlookupallarr = CVErr(xlErrValue)
    Exit Function
End If
ReDim v(1 To rRange.Rows.Count)
For i = 1 To rRange.Rows.Count
    If rRange(i, 1).Text = sSearch Then
        j = j + 1
        If lLookupCol >= 0 Then
            v(j) = rRange(i, lLookupCol).Text
        Else
            v(j) = rRange(i).Offset(0, lLookupCol).Text
        End If
    End If
Next i
i = Application.Caller.Rows.Count
ReDim Preserve v(1 To i)
For j = j + 1 To i
    v(j) = ""
Next j
vlookupallarr = Application.WorksheetFunction.Transpose(v)
End Function


Function lookup2(vSV As Variant, vSA As Variant, vRA As Variant) As Variant
'Similar to lookup() but it looks up the biggest value in vSA which is less-equal than vSV
'vSA has to be sorted, lowest first!!
'Remember that lookup() looks up the smallest value in the search-array which is
'greater-equal than search-value.
Dim i As Long
i = 1
Do While i <= vSA.Count
    If vSV <= vSA(i) Then
        lookup2 = vRA(i)
        Exit Function
    End If
    i = i + 1
Loop
lookup2 = "OUT OF RANGE"
End Function
```

Minimal Number of Banknotes and Coins – sbMinCash

You need to find the minimal number of banknotes and coins to make up a given cash amount?

E4 =sbMinCash(C4;A4:B18)

	A	B	C	D	E	F
1	Input				Output	
2	Available		Wanted		Solution	
3	Notes & Coins	Count	Amount		Used Notes & Coins	Count
4	500	2	255,40		200	1
5	200	2			50	1
6	100	2			5	1
7	50	2			0,2	2
8	20	2				
9	10	2				
10	5	2				
11	2	2				
12	1	2				
13	0,5	2				
14	0,2	2				
15	0,1	2				
16	0,05	2				
17	0,02	2				
18	0,01	2				

The function below might return with an error value, though:
- xlErrNum - The input amount or a face value of a banknote or a coin has more than 2 decimal places
- xlErrValue - A face value of a banknote or a coin is negative or not valid
- xlErrNull - No banknotes nor coins given
- xlErrNA - There is no solution

Known limitation: In some less fortunate cases such as banknotes with face values 1, 6, and 10 the cash amount 13 will come out as 10, 1, 1, 1, and not as 6, 6, 1.
Good message: This will not happen for *normal* cases of face values 1000, 500, 200, 100, 50, 20, 10, 5, 2, 1, 0.50, 0.20, 0.10, 0.05, 0.02, and 0.01.

But just in case you need to check whether sbMinCash has produced the minimum number of banknotes and coins - you might need to multiply all inputs by 100 if your original input values have decimal places since this algorithm is an integer only one:

sbMinCoins Program Code

```
Function minCoins(V As Long, coins As Variant) As Variant
'Source: https://www.geeksforgeeks.org/find-minimum-number-of-coins-that-make-a-change/
'Adapted to VBA by Bernd Plumhoff  15-Jan-2023 PB V0.1
Dim i As Integer, j As Integer, sub_res As Integer
Dim vCoins As Variant
With Application.WorksheetFunction
If V <> Int(V) Then
    minCoins = CVErr(xlErrNum)
    Exit Function
End If
vCoins = .Transpose(.Transpose(coins))
ReDim tabl(0 To V + 1) As Long
tabl(V + 1) = UBound(vCoins)
tabl(0) = 0
```

```
For i = 1 To V
  tabl(i) = 65536 'Do not use type Long. 2147483647 you cannot wait for.
Next i
For i = 1 To V
  For j = 1 To UBound(vCoins, 1)
    If vCoins(j, 1) <= i Then
      sub_res = tabl(i - vCoins(j, 1))
      If sub_res <> 65536 And _
        sub_res + 1 < tabl(i) Then
        tabl(i) = sub_res + 1
      End If
    End If
  Next j
Next i

If tabl(V) = 65536 Then
  minCoins = CVErr(xlErrNA)
Else
  minCoins = tabl(V)
End If
End With
End Function
```

sbMinCash Program Code

```
Function sbMinCash(dAmount As Variant, vNotesCoins As Variant) As Variant
'Returns the minimum number of given notes and coins to make up dAmount
'in a two-dimensional vertical array, for example for dAmount = 255.40 and
'vNotesCoins = {500.2;200.2;100.2;50.2;20.2;10.2;5.2;2.2;1.2;0,5.2;0,2.2;0,1.2;0,05.2;0,02.2;0,01.2}
'the result would be:
'200   1
' 50   1
'  5   1
'  0.2 2
'If you omit the second dimension of vNotesCoins - that is, if you do not provide
'the number of available banknotes and coins (just their face value) then the
'program would assume unlimited supply.
'Return error values:
'xlErrNum -   dAmount or a face values in vNotesCoins have more than 2 decimal places
'xlErrValue - A face value in vNotesCoins is negative
'xlErrNull -  No value in vNotesCoins given
'xlErrNA -    There is no solution
'Known limitation: In some less fortunate cases such as banknotes with
'amounts 1, 6, and 10 the cash amount 13 will result in 10, 1, 1, 1,
'and not in 6, 6, 1.
'(C) (P) by Bernd Plumhoff  15-Jan-2023 PB V0.5
Dim lAmount100 As Long 'dAmount x 100 to be able to apply integer calc
Dim i As Long, j As Long, k As Long

With Application.WorksheetFunction
If dAmount * 100# <> Int(dAmount * 100#) Then
    sbMinCash = CVErr(xlErrNum)
    Exit Function
End If
vNotesCoins = .Transpose(.Transpose(vNotesCoins))
ReDim lNC100(1 To UBound(vNotesCoins, 1)) As Long

'Fill integer array with 100 x non-empty notes and coins set
i = 1
j = 1
Do While i <= UBound(vNotesCoins, 1)
  If vNotesCoins(i, 1) >= 0 Then
    lNC100(j) = Int(vNotesCoins(i, 1) * 100# + 0.5)
    If lNC100(j) / 100# <> vNotesCoins(i, 1) Then
      sbMinCash = CVErr(xlErrNum)
      Exit Function
    End If
    j = j + 1
  Else
    sbMinCash = CVErr(xlErrValue)
    Exit Function
  End If
  i = i + 1
Loop
If j = 1 Then
  sbMinCash = CVErr(xlErrNull)
  Exit Function
End If

ReDim Preserve lNC100(1 To j - 1) As Long
ReDim vR(0 To 1, 1 To j - 1) As Variant
'Sort notes and coins, highest value first
For i = 1 To UBound(lNC100, 1) - 1
  For j = i + 1 To UBound(lNC100, 1)
    If lNC100(i) < lNC100(j) Then
```

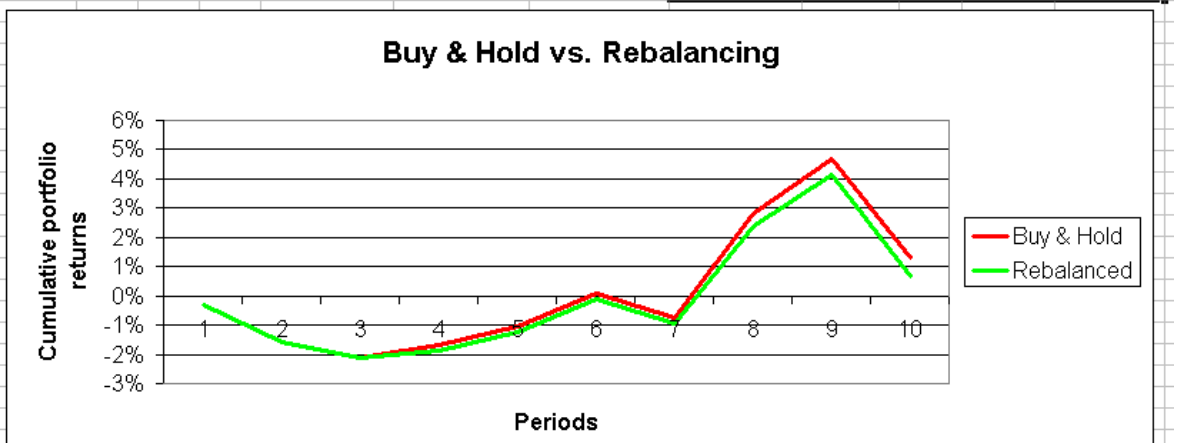
```
      k = lNC100(i)
      lNC100(i) = lNC100(j)
      lNC100(j) = k
    End If
  Next j
Next i

lAmount100 = Int(dAmount * 100# + 0.5)
j = 1
i = 1
Do While lAmount100 > 0 And lNC100(i) > 0
  k = lAmount100 \ lNC100(i)
  If UBound(vNotesCoins, 2) > 1 Then
    If k > vNotesCoins(i, 2) Then
      k = vNotesCoins(i, 2)
    End If
  End If
  If k > 0 Then
    vR(0, j) = lNC100(i) / 100#
    vR(1, j) = k
    j = j + 1
    lAmount100 = lAmount100 - k * lNC100(i)
  End If
  i = i + 1
  If i > UBound(lNC100, 1) Then Exit Do
Loop
If lAmount100 <> 0 Then
  sbMinCash = CVErr(xlErrNA)
Else
  ReDim Preserve vR(0 To 1, 1 To j - 1)
  sbMinCash = .Transpose(vR)
End If
End With
End Function
```

Rebalance AssetClass Weights of a Portfolio – *sbRebalancedReturn*

Portfolio rebalancing is the process of bringing the different asset classes back into proper relationship following a significant change in one or more assets. You return your portfolio to the proper mix of stocks, bonds, cash or other assets when they no longer conform to your plan/limits.

H3 {=RebalancedReturn(B3:D12,B1:D1,J1,L1)}

	A	B	C	D	E	F	G	H	I	J	K	L
1	Weights:	33.33%	33.33%	33.33%	Buy & Hold	Cum Buy & Hold	Cum rebalanced	Rebalanced	Frequency:	4	Tolerance:	3%
2	Period	A1	A2	A3	Portfolio return	Portfolio return	Portfolio return	Portfolio return	A1	A2	A3	Rebalanced?
3	1	-0.99%	-3.00%	3.00%	-0.33%	-0.33%	-0.33%	-0.33%	33.11%	32.44%	34.45%	TRUE
4	2	2.60%	-4.00%	-2.29%	-1.23%	-1.56%	-1.56%	-1.23%	34.39%	31.53%	34.08%	FALSE
5	3	1.45%	-5.00%	1.47%	-0.58%	-2.14%	-2.14%	-0.58%	35.10%	30.13%	34.78%	FALSE
6	4	5.00%	-3.00%	-1.16%	0.45%	-1.69%	-1.86%	0.28%	34.90%	32.24%	32.85%	TRUE
7	5	1.16%	0.06%	0.64%	0.66%	-1.03%	-1.24%	0.62%	33.51%	33.15%	33.34%	TRUE
8	6	5.00%	2.52%	-4.21%	1.13%	0.10%	-0.13%	1.11%	34.80%	33.61%	31.59%	FALSE
9	7	1.22%	0.22%	-4.21%	-0.83%	-0.73%	-0.96%	-0.83%	35.52%	33.97%	30.51%	FALSE
10	8	6.00%	1.16%	2.75%	3.55%	2.82%	2.40%	3.36%	36.43%	33.24%	30.33%	FALSE
11	9	3.00%	0.75%	1.42%	1.86%	4.68%	4.13%	1.72%	33.75%	33.01%	33.23%	TRUE
12	10	-2.70%	-4.51%	-3.21%	-3.38%	1.30%	0.66%	-3.47%	34.02%	32.66%	33.32%	FALSE



sbRebalancedReturn Program Code

```
Const CMaxDouble = 1.79769313486231E+308
Function sbRebalancedReturn(rARM As Range, _
    rIWV As Range, _
    Optional ByVal lRF As Long = 0, _
    Optional dDT As Double = CMaxDouble) As Variant
'RebalancedReturn calculates balanced returns for a
'portfolio with given
'rARM - asset return matrix (columns show different
'       assets, rows show returns per asset over time)
'rIWV - initial weight vector for the assets
'lRF  - rebalancing frequency (in time steps = rows)
'       If lRF > 0 then each lRF time step rebalancing
'       will take place
'       If lRF = 0 then no rebalancing will take place
'       If lRF < 0 then each -lRF time step after last
'       rebalance portfolio will be rebalanced again
'dDT  - drift tolerance %, if any asset has drifted by
'       by more than dDT (relative measure) then the
'       portfolio will be rebalanced AND the internal
'       rebalancing frequency count will be reset
'The output matrix shows portfolio returns % in first
'column, then end-of-period asset weights and finally
'boolean output values in last column, showing whether
'a rebalance happened.
'This function has been inspired by Andreas Steiner's
'similar function.
'(C) (P) by Bernd Plumhoff 19-Mar-2011 PB V0.2
Dim i As Long, j As Long, k As Long, n As Long, m As Long
Dim bDrifted As Boolean, bForceRB As Boolean

n = rARM.Rows.Count 'Number of observations
m = rARM.Columns.Count 'Number of assets

If m <> rIWV.Columns.Count Or _
  rIWV.Rows.Count <> 1 Then
```

```
  sbRebalancedReturn = CVErr(xlErrValue)
  Exit Function
End If

ReDim w0(1 To m) As Double, x(1 To m) As Double
ReDim r(1 To n, 1 To m) As Double

If lRF = 0 Then lRF = n
If lRF < 0 Then
  lRF = -lRF
  bForceRB = True
Else
  bForceRB = False
End If

ReDim vR(1 To n, 1 To m + 2)
For i = 1 To m
  x(i) = rIWV(i)
  w0(i) = x(i)
  For j = 1 To n
    r(j, i) = rARM(j, i)
  Next j
Next i

k = 1
'Model rebalancing tolerance
For i = 1 To n
  If bDrifted And bForceRB Then k = i
  'Calculate period start weights
  vR(i, m + 2) = (i - k) Mod lRF = 0 Or bDrifted
  If vR(i, m + 2) Then
    For j = 1 To m
      x(j) = w0(j)
    Next j
  Else
    For j = 1 To m
      x(j) = vR(i - 1, 1 + j)
    Next j
  End If
  'Calculate portfolio return
  For j = 1 To m
    vR(i, 1) = vR(i, 1) + x(j) * r(i, j)
  Next j
  'Calculate period end weights & check for drift
  bDrifted = False
  For j = 1 To m
    vR(i, 1 + j) = x(j) * (1# + r(i, j)) / (1# + vR(i, 1))
    bDrifted = bDrifted Or Abs(vR(i, 1 + j) - w0(j)) > dDT
  Next j
Next i
sbRebalancedReturn = vR
End Function
```

Optimal Pitstops

You manage a car racing team and you like to plan for optimal pitstops during a race?

Example:

	A	B	C	D	E	F	G	H	I	J
1										
2					Calculate optimal pit stops					
3										
4					**Anzahl Stopps**	**Gesamtzeit [s]**	**Stopps in Runde(n)**			
5		**Number of rounds**	20		0	1771,0				
6		**Time for Round 1 [s]**	80,0		1	1715,0	10; 11			
7		**Increment per Round [s]**	0,9		2	1713,0	7, 13; 7, 14; 8, 14			
8		**Pit Stop [s]**	25,0		3	1724,5	5, 10, 15; 6, 10, 15; 6, 11, 15; 6, 11, 16			
9		**Reset Time after Pit Stop**	80,9		4	1741,4	4, 8, 12, 16; 5, 8, 12, 16; 5, 9, 12, 16; 5, 9, 13, 16; 5, 9, 13, 17			
10					5	1761,0	4, 7, 10, 13, 16; 4, 7, 10, 13, 17; 4, 7, 10, 14, 17; 4, 7, 11, 14, 17; 4, 8, 11, 14, 17; 5, 8, 11, 14, 17			
11					6	1782,4	3, 6, 8, 11, 14, 17; 4, 6, 8, 11, 14, 17; 3, 6, 9, 11, 14, 17; 4, 6, 9, 11, 14, 17; 4, 7, 9, 11, 14, 17; 3, 6, 9, 12, 14, 17; 4, 6, 9, 1			
12					7	1804,7	3, 6, 8, 10, 12, 14, 17; 4, 6, 8, 10, 12, 14, 17; 3, 5, 7, 9, 12, 15, 17; 3, 5, 7, 10, 12, 15, 17; 3, 5, 8, 10, 12, 15, 17; 3, 6, 8, 10,			
13					8	1827,0	3, 6, 8, 10, 12, 14, 16, 18; 4, 6, 8, 10, 12, 14, 16, 18			
14					9	1850,2	2, 4, 6, 8, 10, 12, 14, 16, 18; 3, 4, 6, 8, 10, 12, 14, 16, 18; 3, 5, 6, 8, 10, 12, 14, 16, 18; 3, 5, 7, 8, 10, 12, 14, 16, 18; 3, 5, 7,			
15					10	1874,3	2, 4, 6, 7, 9, 10, 12, 14, 16, 18; 3, 4, 6, 7, 9, 10, 12, 14, 16, 18; 3, 5, 6, 7, 9, 10, 12, 14, 16, 18; 2, 4, 6, 8, 9, 10, 12, 14, 16, 1			
16					11	1898,4	2, 3, 4, 5, 7, 8, 10, 12, 14, 16, 18; 2, 3, 4, 6, 7, 8, 10, 12, 14, 16, 18; 2, 3, 5, 6, 7, 8, 10, 12, 14, 16, 18; 2, 4, 5, 6, 7, 8, 10, 12			
17					12	1922,5	2, 3, 4, 5, 6, 7, 9, 10, 12, 14, 16, 18; 2, 3, 4, 5, 6, 8, 9, 10, 12, 14, 16, 18; 2, 3, 4, 5, 7, 8, 9, 10, 12, 14, 16, 18; 2, 3, 4, 6, 7, 8			
18					13	1946,6	2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 18; 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 14, 16, 18; 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 14, 16, 18; 2, 3,			
19					14	1970,7	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 18; 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 16, 18; 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 16			
20					15	1994,8	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 18; 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 18; 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12			
21					16	2018,9	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18; 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18; 2, 3, 4, 5, 6, 7, 8, 9, 10			
22					17	2043,0	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18; 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19; 2, 3, 4, 5, 6, 7,			
23					18	2067,1	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19			
24					19	2092,1	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19; 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20			
25					20	2117,1	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20			

Optimal_Pitstops Program Code

This program requires (uses) class SystemState (see page 17).

```
Sub optimal_pitstops()
'Berechnet optimale Boxenstopps für ein Autorennteam.
'(C) (P) by Bernd Plumhoff 01-Jan-2023 PB V0.2

Dim i                           As Long
Dim j                           As Long
Dim k                           As Long
Dim m                           As Long
Dim t                           As Long
Dim lRunden                     As Long

Dim dStartzeit                  As Double
Dim dRundenZeit                 As Double
Dim dResetZeit                  As Double
Dim dZeitTotal                  As Double
Dim dZeitBest                   As Double
Dim dInkrement                  As Double
Dim dBoxenstopp                 As Double

Dim sBoxenstopps                As String
Dim sComma                      As String
Dim sSemiColon                  As String

Dim state                       As SystemState

Set state = New SystemState

lRunden = Range("Anzahl_Runden")
ReDim lIdx(1 To lRunden) As Long
dStartzeit = Range("Startzeit")
dResetZeit = Range("Resetzeit")
dInkrement = Range("Inkrement")
dBoxenstopp = Range("Boxenstopp")
Columns("E:G").ClearContents
Range("E4:G4").FormulaArray = Array("Anzahl Stopps", "Gesamtzeit [s]", "Stopps in Runde(n)")
```

```
For t = 0 To lRunden 'Anzahl der Boxenstopps
  dZeitBest = 1E+300
  ReDim c(1 To t + 2) As Long
  For j = 1 To t
    c(j) = j - 1
  Next j
  c(t + 1) = lRunden
  c(t + 2) = 0
  Do
    dZeitTotal = 0#
    dRundenZeit = dStartzeit
    For i = 1 To lRunden
      dZeitTotal = dZeitTotal + dRundenZeit
      For m = 1 To t
        If i = c(m) + 1 Then
          dZeitTotal = dZeitTotal + dBoxenstopp
          dRundenZeit = dResetZeit
          Exit For
        End If
      Next m
      If m > t Then dRundenZeit = dRundenZeit + dInkrement
    Next i
    If (dZeitBest > dZeitTotal) Or (Abs(dZeitBest - dZeitTotal) < 0.000000001) Then
      If dZeitBest > dZeitTotal Then
        dZeitBest = dZeitTotal
        sBoxenstopps = ""
        sSemiColon = ""
      End If
      sComma = ""
      sBoxenstopps = sBoxenstopps & sSemiColon
      For m = 1 To t
        sBoxenstopps = sBoxenstopps & sComma & c(m) + 1
        sComma = ", "
      Next m
      sSemiColon = "; "
    End If
    j = 1
    Do While c(j) + 1 = c(j + 1)
      c(j) = j - 1
      j = j + 1
    Loop
    c(j) = c(j) + 1
  Loop Until j > t
  Cells(t + 5, 5) = t
  Cells(t + 5, 6) = dZeitBest
  Cells(t + 5, 7) = sBoxenstopps
Next t

Columns("E:G").EntireColumn.AutoFit
If Columns("G:G").ColumnWidth > 70 Then Columns("G:G").ColumnWidth = 70

End Sub
```

Optimal Usage of Vacation Days

Each year we members of the working population are asking ourselves when we should take our vacation.

This program shows you the maximal achievable number of free days for your given vacation days:

	A	B	C	D	E	F	G	H	I	J	K
1	**Bank Holidays**		**Input**		**Output**	**Best**			**Second Best**		
2	Mo 01.01.2024		**# of Holidays**		**# of Holidays**	**# of free days**	**First Day**	**Last Day**	**# of free days**	**First Day**	**Last Day**
3	Sa 06.01.2024		30		1	6	Di 24.12.2024	So 29.12.2024	6	Sa 21.12.2024	Do 26.12.2024
4	Fr 29.03.2024				2	9	Sa 21.12.2024	So 29.12.2024	7	Fr 20.12.2024	Do 26.12.2024
5	Mo 01.04.2024		**Start Date**		3	10	Fr 20.12.2024	So 29.12.2024	8	Do 19.12.2024	Do 26.12.2024
6	Mi 01.05.2024		01.01.2024		4	11	Do 19.12.2024	So 29.12.2024	10	Fr 29.03.2024	So 07.04.2024
7	Do 09.05.2024				5	12	Mi 18.12.2024	So 29.12.2024	11	Fr 29.03.2024	Mo 08.04.2024
8	Mo 20.05.2024		**End Date**		6	13	Di 17.12.2024	So 29.12.2024	13	Sa 14.12.2024	Do 26.12.2024
9	Do 30.05.2024		31.12.2024		7	16	Sa 14.12.2024	So 29.12.2024	14	Fr 13.12.2024	Do 26.12.2024
10	Do 15.08.2024				8	17	Fr 13.12.2024	So 29.12.2024	16	Sa 18.05.2024	So 02.06.2024
11	Do 03.10.2024				9	18	Do 12.12.2024	So 29.12.2024	17	Sa 18.05.2024	Mo 03.06.2024
12	Fr 01.11.2024				10	19	Mi 11.12.2024	So 29.12.2024	18	Sa 18.05.2024	Di 04.06.2024
13	Di 24.12.2024				11	20	Di 10.12.2024	So 29.12.2024	20	Sa 07.12.2024	Do 26.12.2024
14	Mi 25.12.2024				12	23	Sa 07.12.2024	So 29.12.2024	21	Fr 06.12.2024	Do 26.12.2024
15	Do 26.12.2024				13	24	Fr 06.12.2024	So 29.12.2024	24	Sa 27.04.2024	Mo 20.05.2024
16	Di 31.12.2024				14	25	Do 05.12.2024	So 29.12.2024	25	Do 09.05.2024	So 02.06.2024
17	Mi 01.01.2025				15	26	Mi 04.12.2024	So 29.12.2024	26	Do 09.05.2024	Mo 03.06.2024
18	Mo 06.01.2025				16	27	Di 03.12.2024	So 29.12.2024	27	Sa 30.11.2024	Do 26.12.2024
19	Fr 18.04.2025				17	30	Sa 30.11.2024	So 29.12.2024	30	Sa 04.05.2024	So 02.06.2024
20	Mo 21.04.2025				18	31	Fr 29.11.2024	So 29.12.2024	31	Sa 04.05.2024	Mo 03.06.2024
21	Do 01.05.2025				19	33	Mi 01.05.2024	So 02.06.2024	32	Sa 27.04.2024	Di 28.05.2024
22	Do 29.05.2025				20	34	Mi 01.05.2024	Mo 03.06.2024	34	Di 30.04.2024	So 02.06.2024
23	Mo 09.06.2025				21	37	Sa 27.04.2024	So 02.06.2024	35	Fr 26.04.2024	Do 30.05.2024
24	Do 19.06.2025				22	38	Sa 27.04.2024	Mo 03.06.2024	38	Fr 26.04.2024	So 02.06.2024
25	Fr 15.08.2025				23	39	Sa 27.04.2024	Di 04.06.2024	39	Fr 26.04.2024	Mo 03.06.2024
26	Fr 03.10.2025				24	40	Mi 01.05.2024	So 09.06.2024	40	Sa 27.04.2024	Mi 05.06.2024
27	Sa 01.11.2025				25	41	Mi 01.05.2024	Mo 10.06.2024	41	Di 30.04.2024	So 09.06.2024
28	Mi 24.12.2025				26	44	Sa 27.04.2024	So 09.06.2024	44	Sa 20.04.2024	So 02.06.2024
29	Do 25.12.2025				27	45	Sa 27.04.2024	Mo 10.06.2024	45	Fr 26.04.2024	So 09.06.2024
30	Fr 26.12.2025				28	46	Sa 27.04.2024	Di 11.06.2024	46	Fr 26.04.2024	Mo 10.06.2024
31	Mi 31.12.2025				29	47	Mi 01.05.2024	So 16.06.2024	47	Sa 27.04.2024	Mi 12.06.2024
32					30	48	Mi 01.05.2024	Mo 17.06.2024	48	Di 30.04.2024	So 16.06.2024

Example: With the given bank holidays in 2024 and just 7 days of vacation you can get 16 free days from 14-Dec-2024 until 29-Dec-2024. You cannot achieve more in 2024, as the second best solution proves (that will result in 14 free days only).

sbOptimalVacationDays Program Code

```
Function sbOptimalVacationDays(lNumberOfVacationDays As Long, _
    dtStartDate As Date, _
    dtEndDate As Date, _
    vBankHolidays As Variant) As Variant
'This function informs you for all days from 1..lNumberOfVacationDays which exact days would result in the
'longest (as well as second longest) series of subsequent free days with given vBankHolidays and assuming
'that weekends (Saturdays and Sundays) are also free.
'(C) (P) Bernd Plumhoff v0.2 07-Jan-2024
Dim dt      As Date
Dim dtMax   As Date
Dim i       As Long
Dim j       As Long
ReDim v(1 To lNumberOfVacationDays, 1 To 7) As Variant
With Application.WorksheetFunction
If lNumberOfVacationDays < 1 Then
  sbOptimalVacationDays = CVErr(xlErrValue)
  Exit Function
End If
If dtStartDate > dtEndDate Then
  sbOptimalVacationDays = CVErr(xlErrNum)
  Exit Function
End If
For i = 1 To lNumberOfVacationDays: v(i, 1) = i: Next i
For dt = dtStartDate To dtEndDate
  For i = 1 To lNumberOfVacationDays
    dtMax = .WorkDay(.WorkDay(dt - 1, 1, vBankHolidays), i, vBankHolidays) - 1
    If dtMax > dtEndDate Then Exit For
      j = dtMax - dt + 1
      If j >= v(i, 2) Then
        v(i, 7) = v(i, 4): v(i, 6) = v(i, 3)
        v(i, 5) = v(i, 2): v(i, 2) = j
        v(i, 3) = dt: v(i, 4) = dtMax
      End If
  Next i
Next dt
sbOptimalVacationDays = v
End With
End Function
```

Create a Round Robin Tournament – sbRondRobin

If you need to organize a round robin tournament you can use this subroutine. It implements the circle method:

Just a sample for 8 or for 9 players to illustrate the principle:

	8	7	6	5
9	Black Table 1 White	White 2 Black	Black 3 White	White 4 Black
	Player 1	2	3	4

In case of even number of players player 1 stays at table 1 and switches colours each game, all other players move on one place counter-clock-wise after each game, and 8 goes to 2.

In case of odd number of players all move on one place counter-clock-wise after each game, 8 moves to 9 and pauses one round, 9 moves to 1.

An example for 6 players:

	Table 1	Table 2	Table 3
Round 1	1 - 6	5 - 2	3 - 4
Round 2	5 - 1	4 - 6	2 - 3
Round 3	1 - 4	3 - 5	6 - 2
Round 4	3 - 1	2 - 4	5 - 6
Round 5	1 - 2	6 - 3	4 - 5

The VBA program also generates this kind of pairings table:

	Player 1	Player 2	Player 3	Player 4	Player 5	Player 6
Player 1	X	Round 5, Table 1, white	Round 4, Table 1, black	Round 3, Table 1, white	Round 2, Table 1, black	Round 1, Table 1, white
Player 2	Round 5, Table 1, black	X	Round 2, Table 3, white	Round 4, Table 2, white	Round 1, Table 2, black	Round 3, Table 3, black
Player 3	Round 4, Table 1, white	Round 2, Table 3, black	X	Round 1, Table 3, white	Round 3, Table 2, white	Round 5, Table 2, black
Player 4	Round 3, Table 1, black	Round 4, Table 2, black	Round 1, Table 3, black	X	Round 5, Table 3, white	Round 2, Table 2, white
Player 5	Round 2, Table 1, white	Round 1, Table 2, white	Round 3, Table 2, black	Round 5, Table 3, black	X	Round 4, Table 3, white
Player 6	Round 1, Table 1, black	Round 3, Table 3, white	Round 5, Table 2, white	Round 2, Table 2, black	Round 4, Table 3, black	X

Further Reading

Suksompong, W. (2018, April 11). Scheduling Asynchronous Round-Robin Tournaments. https://arxiv.org/pdf/1804.04504.pdf

Abel, Finizio, Greig, Lewis (2003). Generalized whist tournament designs. https://www.researchgate.net/publication/222140264_Generalized_whist_tournament_designs

Abel, Finizio, Greig, Morales (2008). Existence of (2, 8) GWhD(v) and (4, 8) GWhD(v) with v ≡ 0, 1 (mod 8).

https://www.researchgate.net/profile/Malcolm_Greig2/publication/257554633_Existence_of_2_8_GWhDv_and_4_8_GWhDv_with_v_equiv_01_mod_8/links/56f56a5f08ae7c1fda2ee68f.pdf

Richard A. DeVenezia's homepage: https://www.devenezia.com/downloads/round-robin/index.html

Ready-to-use tournament tables: https://www.printyourbrackets.com/roundrobin.html

sbRoundRobin Program Code

```
Const CFirstOutputRow = 10

Sub sbRoundRobin()
'Creates a round robin tournament.
'(C) (P) by Bernd Plumhoff  19-May-2023 PB V0.4

Dim bPause           As Boolean

Dim c                As Long
Dim c1               As Long 'Colours, 1 = White (Home game), 2 = Black (Away game)
Dim f                As Long 'Player who has to pause
Dim i                As Long
Dim j                As Long
Dim k                As Long
Dim n                As Long 'Number of players
Dim p                As Long 'Number of players who can play
Dim r                As Long 'Number of rounds
Dim t                As Long 'Temporary storage during moves

Dim state            As SystemState

'Initialize
Set state = New SystemState
n = Range("Number_of_Players")
c = Range("Player1_Game1")
wsR.Range(CFirstOutputRow & ":" & 16382 + CFirstOutputRow).EntireRow.Delete

If n < 2 Then
  wsR.Cells(CFirstOutputRow, 1) = "'Number of players needs to be 2 or higher!"
  Exit Sub
End If
If n > 16383 Then
  wsR.Cells(CFirstOutputRow, 1) = "'Number of players needs to be 16383 or less!"
  Exit Sub
End If
If c < 1 Or c > 2 Then
  wsR.Cells(CFirstOutputRow, 1) = "'Colour of player 1 in game 1 needs to be 1 (White) or 2 (Black)!"
  Exit Sub
End If

wsT.Cells.EntireRow.Delete

ReDim vR(1 To n + 1, 1 To n / 2 + 2) As Variant
ReDim vT(1 To n + 1, 1 To n + 1) As Variant

For i = 1 To n
  vT(1 + i, 1) = "Player " & i
  vT(1, 1 + i) = "Player " & i
  vT(1 + i, 1 + i) = "'X"
Next i

c1 = c

If n Mod 2 = 0 Then
  bPause = False
  p = n
  r = n - 1
Else
  bPause = True
  p = n - 1
  r = n
End If
ReDim a(1 To p) As Long
For i = 1 To p
  a(i) = i
Next i
j = 0
If bPause Then
  f = n
  vR(1, 2) = "Free"
  j = 1
```

```
End If
For i = 1 To p / 2
  vR(1, i + j + 1) = "Table " & i
Next i

For i = 1 To r

  'Output of of current game pairings
  vR(1 + i, 1) = "'Round " & i
  j = 2
  If bPause Then
    vR(1 + i, j) = f & " pauses"
    j = j + 1
  End If
  If c1 = 1 Then
    vR(1 + i, j) = "'" & a(1) & " - " & a(UBound(a))
    vT(1 + a(1), 1 + a(UBound(a))) = "Round " & i & ", Table 1, white"
    vT(1 + a(UBound(a)), 1 + a(1)) = "Round " & i & ", Table 1, black"
  Else
    vR(1 + i, j) = "'" & a(UBound(a)) & " - " & a(1)
    vT(1 + a(1), 1 + a(UBound(a))) = "Round " & i & ", Table 1, black"
    vT(1 + a(UBound(a)), 1 + a(1)) = "Round " & i & ", Table 1, white"
  End If
  j = j + 1
  For k = 2 To UBound(a) / 2
    If (c + k) Mod 2 = 0 Then
      vR(1 + i, j) = "'" & a(k) & " - " & a(UBound(a) - k + 1)
      vT(1 + a(k), 1 + a(UBound(a) - k + 1)) = "Round " & i & ", Table " & k & ", white"
      vT(1 + a(UBound(a) - k + 1), 1 + a(k)) = "Round " & i & ", Table " & k & ", black"
    Else
      vR(1 + i, j) = "'" & a(UBound(a) - k + 1) & " - " & a(k)
      vT(1 + a(k), 1 + a(UBound(a) - k + 1)) = "Round " & i & ", Table " & k & ", black"
      vT(1 + a(UBound(a) - k + 1), 1 + a(k)) = "Round " & i & ", Table " & k & ", white"
    End If
    j = j + 1
  Next k

  'Move on to next round
  If bPause Then
    t = f
    f = a(UBound(a))
    j = 2
  Else
    c1 = 3 - c1 'Switch colour for player 1
    t = a(UBound(a))
    j = 3
  End If
  For k = UBound(a) To j Step -1
    a(k) = a(k - 1)
  Next k
  a(j - 1) = t

Next i

wsR.Range(wsR.Cells(CFirstOutputRow, 1), wsR.Cells(CFirstOutputRow + n, 2 + n / 2)) = vR
wsT.Range(wsT.Cells(1, 1), wsT.Cells(n + 1, n + 1)) = vT
wsT.Cells.EntireColumn.AutoFit

End Sub
```

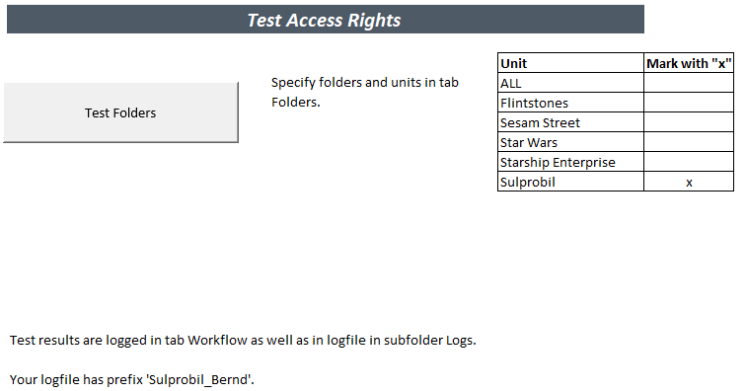
Test Access Rights

You work in a fairly complex environment? Where you need read and write access rights to dozens of folders? You need to request this access from your IT department and and then you want to check whether you already got the requested access?

Then this program will hopefully be of help. First you define all necessary read/write access rights, maybe even for different teams:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Folder [full path without trailing "\", no drive mapping]	Flintstones	Read	Write	Sesam Street	Read	Write	Star Wars	Read	Write	Starship Enterprise	Read	Write	Sulprobil	Read	Write	End
2	C:\Program Files	x	x	x				x	x	x							
3	C:\Windows	x	x	x				x	x	x							
4	C:\Windows\Temp	x	x	x				x	x	x							
5	C:\Program Files				x	x	x	x	x	x							
6	C:\Windows				x	x	x										
7	C:\Windows\Temp				x	x	x										
8	C:\Program Files										x	x	x				
9	C:\Windows										x	x	x				
10	C:\Windows\Temp										x	x	x				
11	C:\Program Files													x	x	x	
12	C:\Windows													x	x	x	
13	C:\Windows\Temp													x	x	x	

Then run this app:



Now you can see which access you have got:

Message
EVER: Sulprobil\Bernd 11.01.2023 10:44:49 [Start_Log] - Logging started with Test_Access_Rights_Version_22, Windows (64-bit) NT 10.00 / Microsoft Windows 11 Home 10.0.22000 and Microsoft Excel [Excel 2021/365] (64-bit) 16.014332 (191029)
INFO: Sulprobil\Bernd 11.01.2023 10:44:49 [Start_Log] - Application ThousandsSeparator '.', DecimalSeparator ',', use system separators
INFO: Sulprobil\Bernd 11.01.2023 10:44:49 [Start_Log] - App.Internl ThousandsSeparator '.', DecimalSeparator ',', ListSeparator ';'
INFO: Sulprobil\Bernd 11.01.2023 10:44:49 [Start_Log] - App.Internl xlCountryCode '49', xlCountrySetting '49'
INFO: Sulprobil\Bernd 11.01.2023 10:44:49 [Start_Log] - VBAProject References: Visual Basic For Applications, Microsoft Excel 16.0 Object Library, OLE Automation, Microsoft Office 16.0 Object Library, Microsoft Forms 2.0 Object Library, Microsoft ActiveX Data Objects 6.1 Library, Microsoft Outlook 16.0 Object Library, Microsoft Scripting Runtime
INFO: Sulprobil\Bernd 11.01.2023 10:44:49 [TestFolders] - Testing access to folders now
INFO: Sulprobil\Bernd 11.01.2023 10:44:49 [TestFolders] - Unit Sulprobil has value 'x'
INFO: Sulprobil\Bernd 11.01.2023 10:44:49 [TestFolders] - Can access (read) folder 'C:\Program Files'
##FATAL: Sulprobil\Bernd 11.01.2023 10:44:49 [TestFolders] - Cannot access (write) folder 'C:\Program Files'. Error number: 75
INFO: Sulprobil\Bernd 11.01.2023 10:44:49 [TestFolders] - Can access (read) folder 'C:\Windows'
##FATAL: Sulprobil\Bernd 11.01.2023 10:44:49 [TestFolders] - Cannot access (write) folder 'C:\Windows'. Error number: 75
INFO: Sulprobil\Bernd 11.01.2023 10:44:49 [TestFolders] - Can access (read) folder 'C:\Windows\Temp'
##FATAL: Sulprobil\Bernd 11.01.2023 10:44:49 [TestFolders] - Cannot access (write) folder 'C:\Windows\Temp'. Error number: 53
INFO: Sulprobil\Bernd 11.01.2023 10:44:49 [TestFolders] - Testing access to folders finished

Testfolders Program Code

```
Public Const AppVersion As String = "Test_Access_Rights_Version_22" 'Each log will show which version it
has been created with

Sub TestFolders()
'Test folder access.
'(C) (P) by Bernd Plumhoff  11-Jan-2023 PB V22

Dim bRead As Boolean, bWrite As Boolean
Dim FileNumber As Integer
```

```
Dim i As Long, j As Long
Dim s As String, sTry As String
Dim state As SystemState
Dim oUnit As Object
Dim v As Variant

Set state = New SystemState
If GLogger Is Nothing Then Call auto_open
GLogger.SubName = "TestFolders"
GLogger.info "Testing access to folders now"
Main.Calculate
Set oUnit = CreateObject("Scripting.Dictionary")
For Each v In Range("Units_Selected")
    s = Main.Range(v.Address).Offset(0, 1).Text
    oUnit(CStr(v)) = s
    If s = "x" Then GLogger.info "Unit " & v & " has value 'x'"
Next v
On Error GoTo ErrHdl
i = 2
s = wsF.Cells(i, 1)
Do While s <> ""
    Application.StatusBar = "Testing " & s
    bRead = False: bWrite = False
    If oUnit("ALL") = "x" Then
        bRead = True
        bWrite = True
    Else
        j = 2
        Do While wsF.Cells(1, j) <> "End"
            If oUnit(wsF.Cells(1, j).Text) = "x" Then
                If wsF.Cells(i, j) = "x" Then
                    If wsF.Cells(i, j + 1) = "x" Then bRead = True
                    If wsF.Cells(i, j + 2) = "x" Then bWrite = True
                End If
            End If
            j = j + 3
        Loop
    End If
    If bRead Then
        'Folder readable? Let us check this by ChDir into it
        sTry = "read"
        ChDir (s)
        GLogger.info "Can access (" & sTry & ") folder '" & s & "'"
    End If
    If bWrite Then
        'Folder writeable? Try to create Remove_me.txt here
        sTry = "write"
        FileNumber = FreeFile
        Open s & "\Remove_me.txt" For Output As #FileNumber
        Write #FileNumber, "This is just a write test. This file should" & _
            "get deleted again automatically. If it does not," & _
            " please do it manually. Thank you."
        Close #FileNumber
        Kill s & "\Remove_me.txt"
        GLogger.info "Can access (" & sTry & ") folder '" & s & "'"
    End If
LabelNext:
    i = i + 1
    s = wsF.Cells(i, 1)
Loop

GLogger.info "Testing access to folders finished"
Exit Sub

ErrHdl:
Select Case Err.Number
Case 52
    'Dir(s, vbDirectory) went wrong
    GLogger.fatal "Cannot access (" & sTry & ") folder '" & s & "'" & _
        IIf(sTry = "read" And bWrite, " - write access expected", "")
    Resume LabelNext 'Back to next row
Case 76
    'ChDir (s) was not possible
    GLogger.fatal "Cannot access (" & sTry & ") folder '" & s & "'" & _
        IIf(sTry = "read" And bWrite, " - write access expected", "")
    Resume LabelNext 'Back to next row
Case Else
    GLogger.fatal "Cannot access (" & sTry & ") folder '" & s & _
        "'. Error number: " & Err.Number & _
        IIf(sTry = "read" And bWrite, " - write access expected", "")
    Resume LabelNext 'Back to next row
End Select

End Sub
```

Advanced VBA Programs

Abstract

In many cases, advanced technical, business, or mathematical knowledge is required for advanced VBA programs. This chapter provides examples of that.

Keeping Track of extreme Cell Values – sbCellWatermarks

If you need to keep track of extreme values of a cell you can use the VBA code as shown below. The code works for numbers as well as for text.

watermark_cell | =-((B1-3)^2)

	A	B	C	D	E
1	x	8			
2	Result f(x)	-25			
3		Result	Date Time	Formula	Input Parameters
4	Max	-4	24/07/2011 10:36:19	=-((B1-3)^2)	5
5	Min	-25	24/07/2011 10:36:24	=-((B1-3)^2)	8

sbCellWatermarks Program Code

```
Sub sbCellWatermarks(rCell As Range, rOutput As Range)
'Keep track of extreme values of a cell calculation.
'Call this sub from a worksheet's calculation event like
'Private Sub Worksheet_Change(ByVal Target As Range)
'  Call sbCellWatermarks(Range("watermark_cell"), _
'       Range("watermark_output"))
'End Sub
'If named range watermark_cell is set to B2 and watermark_output to
'B5:E6 a calculation example could be like:
'    Result  DateTime           Formula       Input Parameters
'Max    0    13/12/2008 12:41   =-((B1-3)^2)  3
'Min   -4    13/12/2008 12:46   =-((B1-3)^2)  5
'(C) (P) by Bernd Plumhoff 24-Jul-2011 PB V0.21

Dim i As Long, k As Long, p As Long, v As Variant

'Check input parameters thoroughly because we will switch off events
If Not TypeOf rCell Is Range Or Not TypeOf rOutput Is Range Then
  Call MsgBox("Input cell or output area are not of type RANGE!", _
          vbOKOnly, "Error")
  Exit Sub
End If
If rCell.Count <> 1 Then
  Call MsgBox("Input range should contain only 1 cell!", _
          vbOKOnly, "Error")
  Exit Sub
End If
If rCell.HasFormula Then p = rCell.DirectPrecedents.Count
If rOutput.Rows.Count < 2 Or rOutput.Columns.Count < 3 + p Then
  Call MsgBox("Output range should contain at least 2 rows and " & _
          3 + p & " columns!", vbOKOnly, "Error")
  Exit Sub
End If

Application.EnableEvents = False

k = Application.Calculation
Application.Calculation = xlCalculationManual
rCell.Calculate

If rCell.FormulaLocal <> rOutput(1, 3) Then
  'If formula changed reset statistics
```

```
  rOutput.ClearContents
  rOutput(1, 1) = rCell
  rOutput(2, 1) = rCell
  rOutput(1, 2) = Now
  rOutput(2, 2) = rOutput(1, 2)
  rOutput(1, 3) = "'" & rCell.FormulaLocal
  rOutput(2, 3) = "'" & rCell.FormulaLocal
  If rCell.HasFormula Then
    i = 4
    For Each v In rCell.DirectPrecedents
      rOutput(1, i) = v
      rOutput(2, i) = v
      i = i + 1
    Next v
  End If
ElseIf rCell > rOutput(1, 1) Then
  rOutput(1, 1) = rCell
  rOutput(1, 2) = Now
  If rCell.HasFormula Then
    i = 4
    For Each v In rCell.DirectPrecedents
      rOutput(1, i) = v
      i = i + 1
    Next v
  End If
ElseIf rCell < rOutput(2, 1) Then
  rOutput(2, 1) = rCell
  rOutput(2, 2) = Now
  If rCell.HasFormula Then
    i = 4
    For Each v In rCell.DirectPrecedents
      rOutput(2, i) = v
      i = i + 1
    Next v
  End If
End If

Application.Calculation = k
Application.EnableEvents = True

End Sub
```

A Task List – *sbTaskList*

If your team runs many different manual tasks, not all of them on each working day but only on specified weekdays or days of a month, then this tasklist creator might be of help to you.

In sheet Param enter your team name or other reference which will be put into the footer of each page, and enter ther valuation day:

	A	B	C
1	**Valuation Date**	12-Sep-2022	**Please enter valuation date here.**
2	**Weekday**	1	
3	**Working Day from Month Start**	8	
4	**Working Day from Month End**	-15	
5	**Footer Text**	www.Sulprobil.com	
6			

Build Tasklist

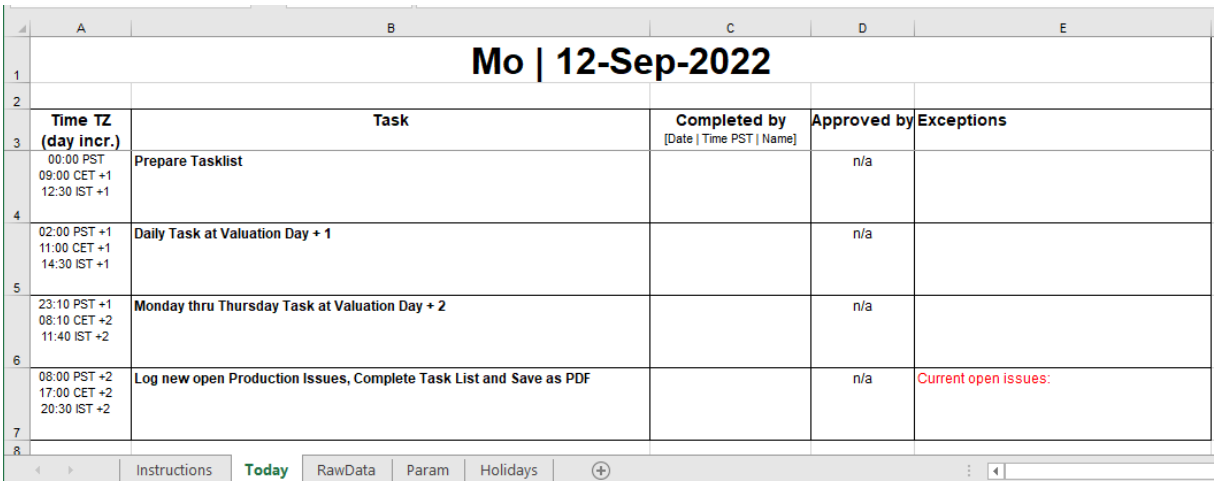
Instructions | Today | RawData | **Param** | Holidays

Define in sheet RawData which tasks needs to be done on which days and set the time of the day this needs to be started. It is not necessary to order all tasks by the time but it might help:

	A	B	C	D	E	F	G	H
1				Leave empty to run on each working day in a month. Set to any sequence of comma-separated integer numbers to run on this day (i.e. -1,1 for last and first working day).				
2				Leave empty to run on each working day in a week. Set to any sequence of comma-separated positive numbers to run on this weekday (i.e. 4,5 for Thursdays and Fridays).				
3	**Month Day**	**Week-day**	**Time [CET]**	**Task**	**Completed by** [Date | Time PST | Name]	**Approved by**	**Exceptions**	**Day Increment**
4			09:00	**Prepare Tasklist**		n/a		1
5			11:00	**Daily Task at Valuation Day + 1**		n/a		1
6		1,2,3,4	08:10	**Monday thru Thursday Task at Valuation Day + 2**		n/a		2
7		5	08:10	**Friday Task at Valuation Day + 2**				2
8	-1		14:00	**Monthly Task at Last Workday**		n/a		2
9	1,2,3		15:00	**Monthly Task at Last Workdays 1, 2, 3**		n/a		2
10			17:00	**Log new open Production Issues, Complete Task List and Save as PDF**		n/a	Current open issues:	2

Instructions | Today | **RawData** | Param | Holidays

Now press the button in sheet Param. You will get in sheet Today:



	A	B	C	D	E
1	Mo | 12-Sep-2022				
2					
3	Time TZ (day incr.)	Task	Completed by [Date | Time PST | Name]	Approved by	Exceptions
4	00:00 PST 09:00 CET +1 12:30 IST +1	Prepare Tasklist		n/a	
5	02:00 PST +1 11:00 CET +1 14:30 IST +1	Daily Task at Valuation Day + 1		n/a	
6	23:10 PST +1 08:10 CET +2 11:40 IST +2	Monday thru Thursday Task at Valuation Day + 2		n/a	
7	08:00 PST +2 17:00 CET +2 20:30 IST +2	Log new open Production Issues, Complete Task List and Save as PDF		n/a	Current open issues:
8					

Instructions | Today | RawData | Param | Holidays

rint sheet Today. Let your team sign off the tasks (when they are done!) and let them note any exceptions (problems, errors, etc.) encountered. I used to scan the signed tasklist at the end of each day to stay paperless and to provide an audit trail.

Note: You can run the whole process paperless if you enter all data electronically and store the final file as pdf.

sbTaskList Program Code

Note: This program requires (uses) the class *SystemState* (see page 17) and the function *ConvertTime* (see page 84).

```
Enum rawdata_columns
  rw_day = 1
  rw_weekday
  rw_time
  rw_task
  rw_completed_by
  rw_approved_by
  rw_exceptions
  rw_day_increment '+1 means time is given for day after valuation day, for example
  rw_comment
End Enum 'rawdata columns

Enum today_columns
  td_time = 1
  td_task
  td_completed_by
  td_approved_by
  td_exceptions
End Enum 'today columns

Sub Build_Tasklist()
'(C) (P) by Bernd Plumhoff 12-Sep-2022 PB V1.07

Dim bTBD                As Boolean 'To be done?
Dim dt                  As Date
Dim lrw                 As Long
Dim ltd                 As Long
Dim s                   As String
Dim v                   As Variant
Dim state               As SystemState

Set state = New SystemState
Application.Calculate
wsToday.Activate
wsToday.Rows("4:1048576").Delete

'Set destination column widths to source's
wsToday.Columns(Chr(64 + td_time) & ":" & Chr(64 + td_time)).ColumnWidth = _
  wsRawData.Columns(Chr(64 + rw_time) & ":" & Chr(64 + rw_time)).ColumnWidth
wsToday.Columns(Chr(64 + td_task) & ":" & Chr(64 + td_task)).ColumnWidth = _
  wsRawData.Columns(Chr(64 + rw_task) & ":" & Chr(64 + rw_task)).ColumnWidth
```

```
wsToday.Columns(Chr(64 + td_completed_by) & ":" & Chr(64 + td_completed_by)).ColumnWidth = _
  wsRawData.Columns(Chr(64 + rw_completed_by) & ":" & Chr(64 + rw_completed_by)).ColumnWidth
wsToday.Columns(Chr(64 + td_approved_by) & ":" & Chr(64 + td_approved_by)).ColumnWidth = _
  wsRawData.Columns(Chr(64 + rw_approved_by) & ":" & Chr(64 + rw_approved_by)).ColumnWidth
wsToday.Columns(Chr(64 + td_exceptions) & ":" & Chr(64 + td_exceptions)).ColumnWidth = _
  wsRawData.Columns(Chr(64 + rw_exceptions) & ":" & Chr(64 + rw_exceptions)).ColumnWidth

lrw = 4: ltd = 4
Do While Not (IsEmpty(wsRawData.Cells(lrw, rw_time))) 'As long as we have tasks timed ...

  Application.StatusBar = "Processing RawData row " & lrw & " ..."
  'Determine whether source row needs to be copied
  bTBD = False
  If IsEmpty(wsRawData.Cells(lrw, rw_day)) And IsEmpty(wsRawData.Cells(lrw, rw_weekday)) Then
    bTBD = True 'Empty rows will be copied
  Else
    'Check Month Day
    If Not (IsEmpty(wsRawData.Cells(lrw, rw_day))) Then
      For Each v In Split(wsRawData.Cells(lrw, rw_day).Text, ",")
        If CLng(v) = wsParam.Range("Evaldate_WDMS") Or CLng(v) = wsParam.Range("Evaldate_WDME") Then
          bTBD = True 'Right day from month start or month end: copy!
          Exit For
        End If
      Next v
    End If
    'Check Weekday
    If Not (IsEmpty(wsRawData.Cells(lrw, rw_weekday))) Then
      For Each v In Split(wsRawData.Cells(lrw, rw_weekday).Text, ",")
        If CLng(v) = wsParam.Range("Evaldate_Weekday") Then
          bTBD = True 'Right weekday: copy!
          Exit For
        End If
      Next v
    End If
  End If

  If bTBD Then
    'Task needs to be done - copy into sheet Today
    wsRawData.Range(wsRawData.Cells(lrw, rw_time), wsRawData.Cells(lrw, rw_exceptions)).Copy
    wsToday.Range(Cells(ltd, td_time), Cells(ltd, td_exceptions)).PasteSpecial Paste:=xlPasteValues
    wsToday.Range(Cells(ltd, td_time), Cells(ltd, td_exceptions)).PasteSpecial _
      Paste:=xlPasteFormats
    wsToday.Range(Cells(ltd, td_time), Cells(ltd, td_exceptions)).PasteSpecial _
      Paste:=xlPasteAllUsingSourceTheme

    dt = Range("Evaldate") + wsRawData.Cells(lrw, rw_day_increment) + wsRawData.Cells(lrw, rw_time)
    dt = ConvertTime(dt, "Central European Standard Time", "Pacific Standard Time")
    s = Format(dt, "hh:nn") & " PST" & IIf(dt - Range("Evaldate") > 1, _
        " +" & Format(Int(dt - Range("Evaldate")), "0"), "") & vbCrLf
    dt = Range("Evaldate") + wsRawData.Cells(lrw, rw_day_increment) + wsRawData.Cells(lrw, rw_time)
    s = s & Format(dt, "hh:nn") & " CET" & IIf(dt - Range("Evaldate") > 1, _
        " +" & Format(Int(dt - Range("Evaldate")), "0"), "") & vbCrLf
    dt = ConvertTime(dt, "Central European Standard Time", "India Standard Time")
    s = s & Format(dt, "hh:nn") & " IST" & IIf(dt - Range("Evaldate") > 1, _
        " +" & Format(Int(dt - Range("Evaldate")), "0"), "") & vbCrLf
    wsToday.Cells(ltd, td_time) = s

    wsToday.Rows(ltd & ":" & ltd).EntireRow.AutoFit
    If wsToday.Rows(ltd & ":" & ltd).RowHeight < wsRawData.Rows(lrw & ":" & lrw).RowHeight Then
      wsToday.Rows(ltd & ":" & ltd).RowHeight = wsRawData.Rows(lrw & ":" & lrw).RowHeight
    End If
    ltd = ltd + 1
  End If
  lrw = lrw + 1
Loop

With wsToday.PageSetup
  .PrintTitleRows = "$1:$3"
  .PrintArea = "$A$1:$" & Chr(64 + td_exceptions) & "$" & ltd - 1
  On Error Resume Next 'Quick and dirty because next command rows will fail in case no printer is defined
  .Orientation = xlPortrait
  .FitToPagesWide = 1
  .FitToPagesTall = 1 + Int(ltd / 5) 'Just to ensure that we have enough pages
  .LeftFooter = wsParam.Range("Footer_Text")
  .CenterFooter = ""
  .RightFooter = "Page &P/&N"
  On Error GoTo 0
End With

End Sub
```

Accounts Receivable Problem

"It's not that I'm so smart, it's just that I stay with problems longer." [Albert Einstein]

Mr Excel's challenge of the month of August 2002 stated:

"An accounts receivable department receives a check from a customer for $4,556.92. Upon looking in the accounting system, there are 54 unpaid invoices, ranging from $77.74 to $5,465.45. The payment must be for some exact combination of entire invoices, but we don't know which invoices are being paid."

The winning solution was published at:
https://www.mrexcel.com/challenges/accounts-receivable-challenge/
Michael Schwimmer presented a nice and elegant solution on his (now retired) website - in German. I translated his version with his permission into English because I found it really beautiful. Any errors are mine, I am sure.

	A	B	C	D	E	F
1	Values	Amount	Tolerance	Result		Calculate
2	$895,39	$4.556,92	$0,00	$4.556,92		
3	$83,60			$77,74		
4	$280,71			$83,60		Instructions
5	$1.021,70			$89,40		1 - Enter your invoice amounts into column A starting cell A2 (up to 100 values).
6	$219,10			$116,14		2 - Enter your desired result amount into cell B2.
7	$1.587,52			$126,69		3 - [Optional] If you enter a tolerance amount into cell C2, any result within Amount - Tolerance up to Amount + Tolerance will be accepted.
8	$507,80			$144,77		4 - Press Calculate button.
9	$628,89			$160,62		5 - When the calculation is finished column D will show the values which sum up to the desired result amount (if there is a solution).
10	$222,52			$185,58		
11	$192,65			$192,65		
12	$194,58			$194,58		
13	$764,18			$219,10		
14	$680,23			$222,52		
15	$244,22			$228,31		
16	$89,40			$230,72		
17	$862,12			$244,22		
18	$1.842,59			$280,71		
19	$329,97			$346,35		
20	$444,98			$441,43		
21	$630,92			$456,68		
22	$440,93			$515,11		
23	$324,84					
24	$978,53					
25	$144,77					

Note: I changed the program to indicate that there is no solution in case the program cannot find any solution with the given tolerance.

Calculate and Combinations Program Code

Please notice that this code requires (calls) the user-defined function ArrayDim which you can find on page 34 and that you need to

- either set the worksheet's codename (its internal VBA name) name of your sheet to 'ws'
- or at program start execute commands `Dim ws As Worksheet: Set ws = Sheets("Your_Sheetname")`

```
Sub Calculate()

Dim dGoal      As Double
Dim dTolerance As Double
Dim dAmounts() As Double
Dim vResult    As Variant
Dim m          As Long
Dim n          As Long

With ws

  dGoal = .Range("B2")
  dTolerance = .Range("C2")
  ReDim dAmounts(1 To 100)
  For m = 2 To 101
    If (.Cells(m, 1) <> "") And (IsNumeric(.Cells(m, 1))) Then
      dAmounts(m - 1) = .Cells(m, 1)
    Else
      ReDim Preserve dAmounts(1 To m - 1)
      Exit For
    End If
  Next
  ReDim Preserve dAmounts(1 To UBound(dAmounts) - 1)

  vResult = Combinations(dAmounts, dGoal, dTolerance)
  Application.ScreenUpdating = False
  .Range("D3:D65536").ClearContents
  If ArrayDim(vResult) < 1 Then
    .Cells(3, 4) = "No solution within this tolerance"
  Else
    .Range(.Cells(3, 4), .Cells(UBound(vResult) + 3, 4)) = vResult
  End If
  .Calculate
  Application.ScreenUpdating = True

End With

End Sub

Function Combinations( _
   Elements As Variant, _
   Goal As Double, _
   Optional Tolerance As Double, _
   Optional SoFar As Variant, _
   Optional Position As Long) As Variant

Dim i           As Long
Dim k           As Long
Dim dCompare    As Double
Dim dDummy      As Double
Dim vDummy      As Variant
Dim vResult     As Variant

If Not IsMissing(SoFar) Then

  'Sum of elements so far
  For Each vDummy In SoFar
    dCompare = dCompare + vDummy
  Next

Else

  'Start elements sorted by amount
  For i = 1 To UBound(Elements)
    For k = i + 1 To UBound(Elements)
      If Elements(k) < Elements(i) Then
        dDummy = Elements(i)
        Elements(i) = Elements(k)
        Elements(k) = dDummy
      End If
    Next
  Next

  Set SoFar = New Collection

End If
```

```
If Position = 0 Then Position = LBound(Elements)
For i = Position To UBound(Elements)

  'Add current element
  SoFar.Add Elements(i)
  dCompare = dCompare + Elements(i)

  If Abs(Goal - dCompare) < (0.001 + Tolerance) Then

    'Goal achieved
    k = 0
    ReDim vResult(0 To SoFar.Count - 1, 0)
    For Each vDummy In SoFar
      vResult(k, 0) = vDummy
      k = k + 1
    Next
    Combinations = vResult
    Exit For

  ElseIf dCompare < (Goal + 0.001 + Tolerance) Then
    'Enough room for another element
    'Call recursively starting with next higher amount
    vResult = Combinations(Elements, Goal, Tolerance, SoFar, i + 1)
    If IsArray(vResult) Then
      Combinations = vResult
      Exit For
    Else
      SoFar.Remove SoFar.Count
      dCompare = dCompare - Elements(i)
    End If

  Else

    'Amount too high
    SoFar.Remove SoFar.Count
    Exit For

  End If

Next 'Try next higher amount

End Function
```

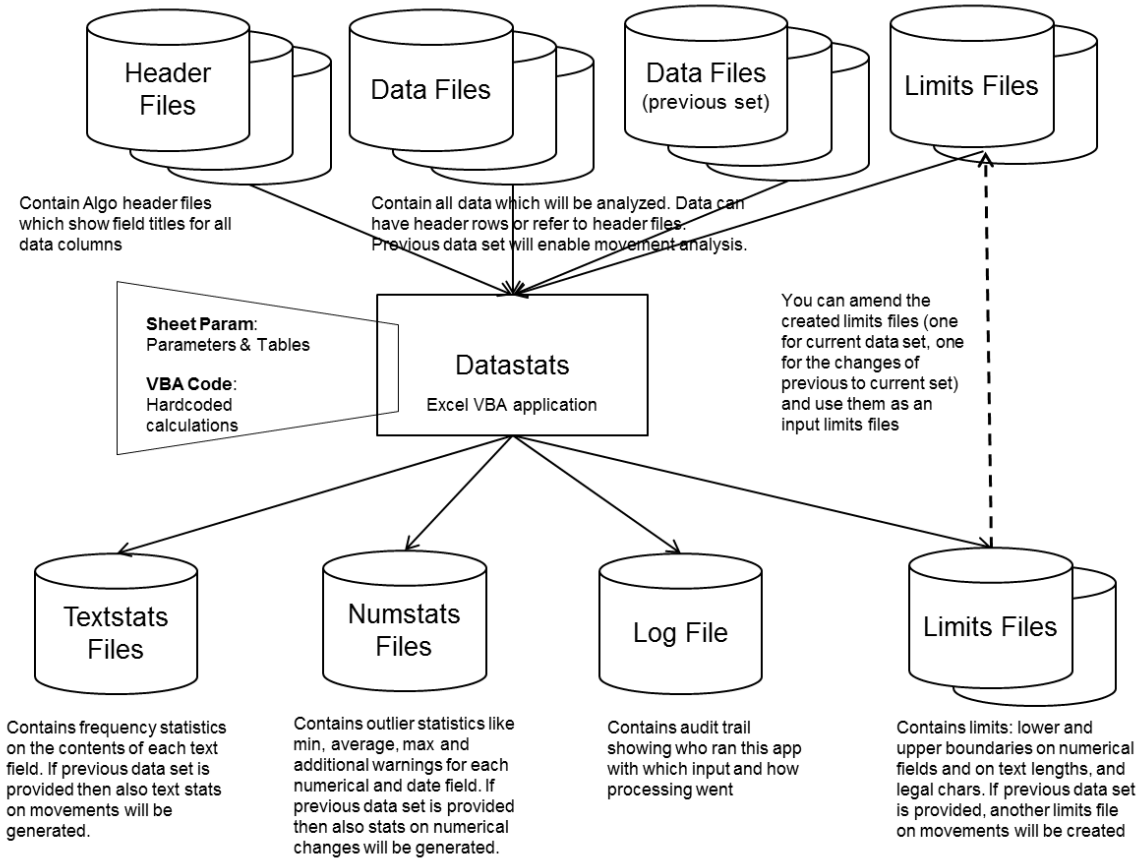
Data Analysis – sbDatastats

Wouldn't it be nice to have a data analyser which shows you numerical and string outliers without much effort? Well, this has been the design idea and the implementation approach of the Excel VBA sbDatastats application which this document describes.
You copy all your data files into one folder, all header files into another one, and you define a log folder. Then you just press a button in the Workflow tab, and the program will deliver outlier statistics for all numerical and date fields (showing min, average, max plus some additional warnings) as well as for all text fields (showing a frequency statistic for all contents).
You, that can be any team member in Data Service, in IT or Operations, in any business unit or even internal or external audit (for a thorough IT data audit).

System Handbook

This part describes how the application needs to be installed and maintained. It requires basic MS Windows and MS Excel knowledge only.

Overview



The installation is pretty straight forward. First create a new folder and copy the Excel VBA sbDatastats application into it, then generate subfolders for data (Data_Files and Data_Files_Prev), Algo One headers (Header_Files), logs (Logs), configuration (Config), and output (Datastats) files, finally initialize application parameters in sheet Param of Excel VBA application.

Application design is that the user should only need to copy new data files into the data subfolder(s) and to amend, rename, and move (to folder Config) the limits file(s) in order to direct the application on which border values to act. If you also provide a previous data set folder with corresponding data then this application will produce a movement analysis as well.

Parameters in Sheet Param

Log_Level – Usually 1 to get all log outputs including errors, warnings and simple information. If set to 2 only errors and warnings will be logged but then you will lose your audit trail because filenames in use and checking information will no longer be printed.

Warning Threshold (# of Stdevs) – Contains the number of standard deviations from which on an extreme value will be warned about in the Warning column of Numstats output. A standard value of 3 is suggested to be used.

Keep stats in tabs – The sbDatastats spreadsheet contains four tabs (sheets) called Numstats, Textstats, LimitsIn, and Limitsout which normally show the run results which will also be stored in output files Numstats_yyyymmdd.csv, Textstats_yyyymmdd.csv, Input_Limits_File, and Output_Limits_File in subfolder Datastats. For frequent runs it is very convenient to have the output in the application spreadsheet for the ease of lookup. In this case set this value to True. If you strictly need to separate the data from the program, then set this to False, and these tabs get cleared at program end (usually necessary if run by audit).

Max number of string attributes per field – This parameter drives how many different fields are maximally accepted in the Attributes field of the limits input file.

No single character check – f this parameter is set to ‘True’ we do not check whether single characters are valid.

Input File Delimiter – his parameter is defining the field separation character for input files. This would in general be ‘,’ for comma separated files (CSV), but you can define any character, for example ‘|’ for pipe separated files ‘|’ (PSV). Config files and limit files for the Datastats application are all CSV.

User Handbook

This is a description of how to run this application.

Summary

Short instruction is: Just copy your data files into subfolder Data_Files (and previous period’s ones into subfolder Data_Files_Prev). For Algo data files you need to copy corresponding header files into subfolder Header_Files, for all other data files you need to specify one or more sort columns in files FileSpecs.csv in subfolder Config. Then press button ‘1. Read Input’ in sheet Workflow.
Now wait for the final message ‘INFO: …. 16/08/2015 17:06:14 [Read_Input] - Reading Input finished’.

Errors, warnings and other useful information will pop up during execution in sheet Workflow.

Example:

	Message
1. Read Input	INFO: Bernd 12/01/2019 20:09:57 [Read_Input] - Reading Input started with Datastats Version 36 and with data folder D:\Bernd\Dropbox\CD30_Excel\Datastats_app\Data_Files
	INFO: Bernd 12/01/2019 20:09:58 [Read_Input] - Reading D:\Bernd\Dropbox\CD30_Excel\Datastats_app\Data_Files\aaa_ABSInstr_20190113.csv finished: 64 rows read
	WARN: Bernd 12/01/2019 20:09:58 [Read_Input] - File aaa_ABSInstr_20190113.csv, Field CurrencyUNIT, Row 33, Value 'XXX' not in Ok values list of limits file
	INFO: Bernd 12/01/2019 20:09:58 [Read_Input] - Reading D:\Bernd\Dropbox\CD30_Excel\Datastats_app\Data_Files_Prev\aaa_ABSInstr_20190106.csv finished: 64 rows read
	INFO: Bernd 12/01/2019 20:09:58 [Read_Input] - Reading D:\Bernd\Dropbox\CD30_Excel\Datastats_app\Data_Files\aaa_BondInstr_20190113.csv finished: 66 rows read
	WARN: Bernd 12/01/2019 20:09:58 [Read_Input] - File aaa_BondInstr_20190113.csv, Field SpotPriceVAL, Row 21, Value 1023.7702 > 150 [Max Limit]
	INFO: Bernd 12/01/2019 20:09:59 [Read_Input] - Reading D:\Bernd\Dropbox\CD30_Excel\Datastats_app\Data_Files_Prev\aaa_BondInstr_20190106.csv finished: 66 rows read
	WARN: Bernd 12/01/2019 20:09:59 [Read_Input] - Change on File aaa_BondInstr_20190113.csv, Field MaturityDATE, Row 19, Value 366 > 0 [Max Limit]
	WARN: Bernd 12/01/2019 20:09:59 [Read_Input] - Change on File aaa_BondInstr_20190113.csv, Field SpotPriceVAL, Row 7, Value -0.057250908 < -0.05 [Min Limit]
	WARN: Bernd 12/01/2019 20:09:59 [Read_Input] - Change on File aaa_BondInstr_20190113.csv, Field SpotPriceVAL, Row 9, Value -0.054794057 < -0.05 [Min Limit]
	WARN: Bernd 12/01/2019 20:09:59 [Read_Input] - Change on File aaa_BondInstr_20190113.csv, Field SpotPriceVAL, Row 11, Value -0.051592279 < -0.05 [Min Limit]
	WARN: Bernd 12/01/2019 20:09:59 [Read_Input] - Change on File aaa_BondInstr_20190113.csv, Field SpotPriceVAL, Row 15, Value 1022.692554 > 0.05 [Max Limit]
	WARN: Bernd 12/01/2019 20:09:59 [Read_Input] - Change on File aaa_BondInstr_20190113.csv, Field SpotPriceVAL, Row 49, Value -0.05163524 < -0.05 [Min Limit]
	WARN: Bernd 12/01/2019 20:09:59 [Read_Input] - Change on File aaa_BondInstr_20190113.csv, Field SpotPriceVAL, Row 67, Value -0.055971756 < -0.05 [Min Limit]
	INFO: Bernd 12/01/2019 20:10:00 [Read_Input] - Reading Input finished with Datastats Version 36
	INFO: Bernd 12/01/2019 20:10:00 [Read_Input] - ----------

But they will also be written into the logfile (file sbDatastats_Logfile_YYYYMMDD.csv in subfolder Logs) for audit purposes.

The output files of this application are in subfolder Datastats.

Config File FileSpecs.csv

This file resides in subfolder Config and contains necessary information for the sbDatastats application to perform movement checks on normal (i. e. non-Algo One) input files if a previous data set is provided:

	A	B	C	D
1	Filename	SortColumn1	SortColumn2	SortColumn3
2	Input	7		
3	More_Input	7		

Up to three sort columns tell the application on which column(s) to sort the data sets so that the program can identify deleted, new or changed (can even be identical) data records. Please note that column A just needs to contain the 'core' filename. The program will automatically detect whether this core FILENAME is a prefix or a suffix or anywhere in the middle of the actual filenames which can read 'FILENAME_20171004.csv' or '20171004_FILENAME.psv', for example.

Numstats Output

With the outlier statistic on each numerical and date field you can easily spot potential errors:

	A	B	C	D	E	F	G	H	I	J	K
1	Filename	RowType	Fieldname	Warning	Min 1	Min 2	Min 3	Average	Max 3	Max 2	Max 1
2	aaa_BondInstr_20190113.csv	BAS:DM Bond FutureSPEC	MaturityDATE		26-Nov-2030	11-Jan-2031	30-May-2031	27-Feb-2035	19-Jun-2039	19-Feb-2040	04-Sep-2040
3	aaa_BondInstr_20190113.csv	BAS:DM Bond FutureSPEC	SpotPriceVAL	Max is off by more than 3 stdev.	0.97	0.99	0.99	32.13	1.25	1.29	1,023.77

The red circle highlights a coupon rate which is off by a factor of 1000.

Numstats Move Output

With the outlier statistic on ***changes*** of numerical and date fields you also spot potential errors:

	A	B	C	D	E	F	G	H	I	J	K
1	Filename	RowType	Fieldname	Warning	Min 1	Min 2	Min 3	Average	Max 3	Max 2	Max 1
2	aaa_BondInstr_20190113.csv	BAS:DM Bond FutureSPEC	MaturityDATE	Max is off by more than 3 stdev.	-	-	-	11.09	-	-	366.00
3	aaa_BondInstr_20190113.csv	BAS:DM Bond FutureSPEC	SpotPriceVAL	Max is off by more than 3 stdev.	(0.06)	(0.06)	(0.05)	30.98	0.04	0.04	1,022.69

In the case above you might want to check the change of the maturity date (which normally should not change) and obviously the price change is in error.

Textstats Output

With the frequency statistic on each text field you can easily verify whether there are suspicious string values:

	A	B	C	D	E	F	G	H	I	J
1	Filename	RowType	Fieldname	Warning	Char	Char Frequency	Text Length	Frequency	Text	Count
2	aaa_ABSInstr_20190113.csv	BAS:DM Bond FutureSPEC	ContractSizeVAL		CHAR(067) = 'C'	32	15	32	'ContractSizeVAL'	32
3	aaa_ABSInstr_20190113.csv	BAS:DM Bond FutureSPEC	CurrencyUNIT		CHAR(088) = 'X'	3	3	32	'GBP'	14
4									'EUR'	11
5									'USD'	6
6									'XXX'	1

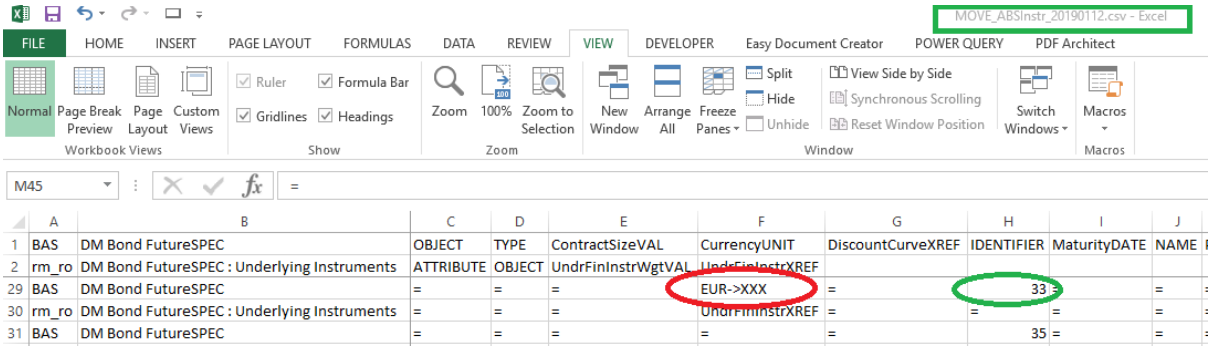
This example show the erroneous currency ‘XXX’. Infrequent string values should generally be reviewed.

Textstats Move Output

With the frequency statistic on the ***change*** of each text field in case you provided a previous data set you can also verify whether there are strange changes:

	A	B	C	D	E	F	G	H	I	J
1	Filename	RowType	Fieldname	Warning	Char	Char Frequency	Text Length	Frequency	Text	Count
2	aaa_ABSInstr_20190113.csv	BAS:DM Bond FutureSPEC	ContractSizeVAL				1	32	'='	32
3	aaa_ABSInstr_20190113.csv	BAS:DM Bond FutureSPEC	CurrencyUNIT				1	31	'='	31
4							8		'EUR->XXX'	1
5	aaa_ABSInstr_20190113.csv	BAS:DM Bond FutureSPEC	DiscountCurveXREF				1	32	'='	32

This example shows the erroneous change in the currency field from ‘EUR’ to ‘XXX’.
If you want to identify the offending data record you need to look into the corresponding MOVE file for the file given in column A:



	A	B	C	D	E	F	G	H	I	J
1	BAS	DM Bond FutureSPEC	OBJECT	TYPE	ContractSizeVAL	CurrencyUNIT	DiscountCurveXREF	IDENTIFIER	MaturityDATE	NAME
2	rm_ro	DM Bond FutureSPEC : Underlying Instruments	ATTRIBUTE	OBJECT	UndrFinInstrWgtVAL	UndrFinInstrXREF				
29	BAS	DM Bond FutureSPEC	=	=	=	EUR->XXX	=	33	=	=
30	rm_ro	DM Bond FutureSPEC : Underlying Instruments	=	=	=	UndrFinInstrXREF	=	=	=	=
31	BAS	DM Bond FutureSPEC	=	=	=	=	=	35	=	=

In the above case the instrument with the identifier 33 was changed in error.

Output Limits File

A sample extract of an output limits file in subfolder Datastats after you have run the application:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Limit File Version 1.00												
2	Filename	RowType	Fieldname	Allow Empty	Min	Max	Min Date	Max Date	Length Min	Length Max	Formula	OkFormat	OkValues
3	ABSInstr	BAS:DM Bond FutureSPEC	ContractSizeVAL						15	15			ContractSizeVAL|
4	ABSInstr	BAS:DM Bond FutureSPEC	CurrencyUNIT						3	3			GBP|EUR|USD|XXX|
5	ABSInstr	BAS:DM Bond FutureSPEC	DiscountCurveXREF						17	17			DiscountCurveXREF|
6	ABSInstr	BAS:DM Bond FutureSPEC	IDENTIFIER		1	63							
41	BondInstr	BAS:DM Bond FutureSPEC	ProductTypeENUM						15	15			ProductTypeENUM|
42	BondInstr	BAS:DM Bond FutureSPEC	SettlementTYPE						14	14			SettlementTYPE|
43	BondInstr	BAS:DM Bond FutureSPEC	SpotPriceVAL		0.97169473	1023.7702							
44	BondInstr	BAS:DM Bond FutureSPEC	StlmntDayRuleBUSD						17	17			StlmntDayRuleBUSD|
45	BondInstr	BAS:DM Bond FutureSPEC	StlmntDayRuleCONV						17	17			StlmntDayRuleCONV|

As you can see, this file lists min and max numerical and date values as well as text lengths and (if field NoSingleCharCheck in tab Param is set to 'False') the characters used. A 'True' in column Allow Empty indicates that this field is empty for at least one input record.

The output limits file is intended to help you defining an input limits file. The output limits file shows limits and values which the data analyser has encountered when reading previous input.
An example of a limits input file (which you need to specify as file Limits_Input.csv in subfolder Config):

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Limit File Version 1.00												
2	Filename	RowType	Fieldname	Allow Empty	Min	Max	Min Date	Max Date	Length Min	Length Max	Formula	OkFormat	OkValues
3	ABSInstr	BAS:DM Bond FutureSPEC	CurrencyUNIT						3	3			GBP|EUR|USD|
4	BondInstr	BAS:DM Bond FutureSPEC	SpotPriceVAL		0	150							

The columns of limits files contain:

Filename, RowType, and Fieldname – These columns are identifying the fields in the input files for which the limits apply. RowType is only used or necessary for Algo(rithmics) input files.

AllowEmpty – The only valid values in here are 'True' or 'False' (or empty which means 'False'). If set to 'True', empty cells are allowed for the corresponding field in the input file. If set to 'False', the data analyser will warn on empty input data cells.

Min, Max – These numerical values define the minimal resp. maximal allowed input values. Leave these empty if you do not want to apply them.

Min Date, Max Date – These date values define the minimal resp. maximal allowed date input values. Leave these empty if you do not want to apply them.

Length Min, Length Max – These numerical values define the minimal resp. maximal allowed input values. Leave these empty if you do not want to apply them.

Formula – You can define a worksheet formula here which is required to result either in 'True' or 'False'. The data analyser will ignore 'True' results warn on 'False' ones. You can use worksheet functions here, or to field names of the current input record: [*field*] refers to field *field*, [ThisCell] will refer to current input cell.

Beware: Excel / VBA's internal date representation is mm/dd/yyyy (US format). You will need to perform a string conversion to this format before you can apply date functions like DATEVALUE.
OkFormat – You can define a regular expression here. If the input cell adheres to this expression, the data analyser will not complain.

OkValues – Define your valid input values separated by '|', and the data analyser will only accept those.

CHAR(000) .. CHAR(255) – You need to set these columns to 'True' for string fields.

Output Limits Move File

A sample extract of an output limits move(ment) file (which you will find with name Limits_Move_Output.csv in subfolder Datastats if you have run the application with previous period's input files in subfolder Data_Files_Prev so that the program could analyse the moves / changes):

	A	B	C	D	E	F	G	H	I	J
1	Limit Moves File Version 1.00									
2	Filename	RowType	Fieldname	Allow Empty	Min	Max	Min Date	Max Date	Length Min	Length Max
33	BondInstr	BAS:DM Bond FutureSPEC	DiscountCurveXREF						1	1
34	BondInstr	BAS:DM Bond FutureSPEC	MaturityDATE		0	366				
35	BondInstr	BAS:DM Bond FutureSPEC	NAME						1	1
36	BondInstr	BAS:DM Bond FutureSPEC	PositionUnitsVAL						1	1
37	BondInstr	BAS:DM Bond FutureSPEC	PortfolioXREF						1	1
38	BondInstr	BAS:DM Bond FutureSPEC	ProductTypeENUM						1	1
39	BondInstr	BAS:DM Bond FutureSPEC	SettlementTYPE						1	1
40	BondInstr	BAS:DM Bond FutureSPEC	SpotPriceVAL		-0.057250908	1022.692554				

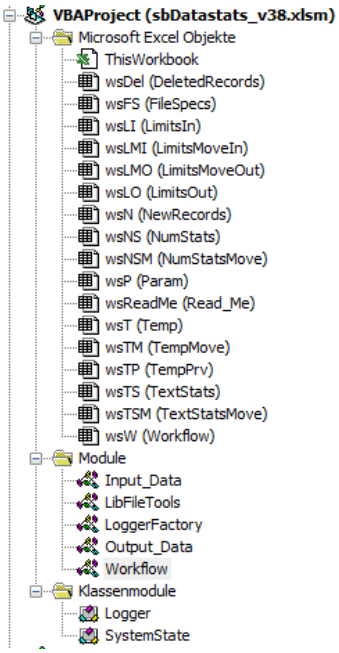
As you can see, this file lists min and max ***changes*** of numerical and date values as well as text lengths of changes (a '1' clearly indicates no change at all because it's '=').
For the example given above you might like to define a Limits_Move_Input.csv like:

	A	B	C	D	E	F	G	H	I	J
1	Limit Moves File Version 1.00									
2	Filename	RowType	Fieldname	Allow Empty	Min	Max	Min Date	Max Date	Length Min	Length Max
3	BondInstr	BAS:DM Bond FutureSPEC	MaturityDATE		0	0				
4	BondInstr	BAS:DM Bond FutureSPEC	SpotPriceVAL		-0.05	0.05				

sbDatastats Program Code

This application uses classes Logger and SystemState as well as module LibFileTools.

The VBA project:



Module Input_Data

```
Option Explicit

'Source: http://www.zerrtech.com/content/excel-vba-open-csv-file-and-import [14-Jun-2011]

Function doFileQuery(fileName As String, outSheet As String, Optional sDelimiter As String = ",") As
Boolean
'Reads in csv files.
' filename = CSV filename (complete pathname!)
' outSheet = name of the worksheet in the current workbook
'            where the data should go, will start in A1
'Change History:
'Version Date        Programmer Change
'1.00    14/06/2011 Bernd      Copied from the web
'1.01    15/06/2011 Bernd      Drop query before adding a new one, error handling, logging
'2       24/09/2017 Bernd      Also process psv files "|"
Dim connectionName           As String
Dim qt                       As QueryTable
On Error GoTo ErrHdl
doFileQuery = True
If Worksheets(outSheet).QueryTables.Count > 0 Then Worksheets(outSheet).QueryTables.Item(1).Delete
'GLogger.info "Create Querytable"
connectionName = "TEXT;" & fileName
With Worksheets(outSheet).QueryTables.Add(Connection:=connectionName, _
                                          Destination:=Worksheets(outSheet).Range("A1"))
  .Name = fileName
  .FieldNames = True
  .RowNumbers = False
  .FillAdjacentFormulas = False
  .PreserveFormatting = True
  .RefreshOnFileOpen = False
  .RefreshStyle = xlOverwriteCells
  .SavePassword = False
  .SaveData = True
  .AdjustColumnWidth = True
  .RefreshPeriod = 0
  .TextFileColumnDataTypes = Array(xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, _
                                   xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, _
                                   xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, _
                                   xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, _
                                   xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, _
                                   xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, _
                                   xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, _
                                   xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, _
                                   xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, _
                                   xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, _
```

```
                                    xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, _
                                    xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, _
                                    xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, _
                                    xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, xlTextFormat, _
                                    xlTextFormat, xlTextFormat, xlTextFormat)
  .TextFilePromptOnRefresh = False
  .TextFilePlatform = 437
  .TextFileStartRow = 1
  .TextFileParseType = xlDelimited
  .TextFileTextQualifier = xlTextQualifierDoubleQuote
  .TextFileConsecutiveDelimiter = False
  .TextFileTabDelimiter = False
  .TextFileSemicolonDelimiter = False
  .TextFileCommaDelimiter = False
  .TextFileSpaceDelimiter = False
  Select Case sDelimiter
  Case ","
    .TextFileCommaDelimiter = True
  Case ";"
    .TextFileSemicolonDelimiter = True
  Case " "
    .TextFileSpaceDelimiter = True
  Case Else
    .TextFileOtherDelimiter = sDelimiter
  End Select
  .Refresh BackgroundQuery:=False
End With
Exit Function
ErrHdl:
GLogger.SubName = "doFileQuery"
GLogger.fatal "Could not establish Querytable successfully"
doFileQuery = False
Resume errhdlexit
errhdlexit:
End Function

'If you have a file other than a CSV file, just change the
'Excel VBA options (.TextFileTabDelimiter, '.TextFileSemicolonDelimiter, etc..)
'to True and the (.TextFileCommaDelimiter) to False.
```

Module Output_Data

```
Option Explicit

Dim GsCurrentWorkbook As String
Dim GlCurrentFormat As Long

'Source: http://www.erlandsendata.no/english/index.php?d=envbatextexportcsv
Sub ExportRangeAsDelimitedText(SourceWS As String, SourceAddress As String, _
                               TargetFile As String, SepChar As String, SaveValues As Boolean, _
                               ExportLocalFormulas As Boolean, AppendToFile As Boolean)
' Exports the data in Workbooks(SourceWB).Worksheets(SourceWS).Range(SourceAddress) to
' the textfile TargetFile in CSV format, uses SepChar as column delimiter
' Example:     ExportRangeAsDelimitedText ThisWorkbook.Name, _
  "ExportSheet", "A3:E23", "C:\FolderName\DelimitedText.txt", _
  ";", True, True, False

  Dim SourceRange As Range, sC As String * 1
  Dim A As Integer, r As Long, c As Integer, totr As Long, pror As Long
  Dim fn As Integer, LineString As String, tLine As String
  ' validate the input data if necessary
  Worksheets(SourceWS).Activate
  If Application.WorksheetFunction.CountA(Range(SourceAddress)) = 0 Then Exit Sub
  If Not AppendToFile Then
    On Error Resume Next
    Kill TargetFile
    On Error GoTo 0
  End If
  If UCase(SepChar) = "TAB" Or UCase(SepChar) = "T" Then
    sC = Chr(9)
  Else
    sC = Left(SepChar, 1)
  End If

  ' perform export
  Set SourceRange = Range(SourceAddress)
  On Error GoTo NotAbleToExport
  fn = FreeFile
  Open TargetFile For Append As #fn  ' open textfile for new input
  On Error GoTo 0
  ' determine the total number of rows to process
  totr = 0
  For A = 1 To SourceRange.Areas.Count
    totr = totr + SourceRange.Areas(A).Rows.Count
  Next A
  ' start writing the character-separated textfile
  pror = 0
  For A = 1 To SourceRange.Areas.Count
    For r = 1 To SourceRange.Areas(A).Rows.Count
      LineString = ""
      For c = 1 To SourceRange.Areas(A).Columns.Count
        tLine = ""
        On Error Resume Next
        If SaveValues Then
          tLine = SourceRange.Areas(A).Cells(r, c).Value
        Else
          If ExportLocalFormulas Then
            tLine = SourceRange.Areas(A).Cells(r, c).FormulaLocal
          Else
            tLine = SourceRange.Areas(A).Cells(r, c).Formula
          End If
        End If
        On Error GoTo 0
        LineString = LineString & tLine & sC
      Next c
      pror = pror + 1
      If pror Mod 50 = 0 Then
        Application.StatusBar = "Writing delimited textfile " & _
                                Format(pror / totr, "0 %") & "..."
      End If
      If Len(LineString) > 1 Then LineString = Left(LineString, Len(LineString) - 1)
      If LineString = "" Then
        Print #fn,
      Else
        Print #fn, LineString
      End If
    Next r
  Next A
  Close #fn  ' close the textfile
NotAbleToExport:
  Set SourceRange = Nothing
  Application.StatusBar = False
End Sub
```

Module Workflow

```
Option Explicit

'Change History:
'Version Date       Programmer Change
'1       23/09/2015 Bernd      Initial Version
'37      19/01/2019 Bernd      lRowTypes + 2 to correctly skip header rows and use 1 instead of True
'                              for limAllowEmpty because of internationalization, and check for
'                              Datastats subfolder
'38      22/01/2025 Bernd      New modules LibFileTools, SystemState, and Logger / LoggerFactory

Public Const AppVersion As String = "Datastats_Version_38"
Public Const CLimFileVersion As String = "Limit File Version 1.00"
Public Const CLimMovFileVersion As String = "Limit Moves File Version 1.00"
Public Const CMinDouble As Double = -1.79769313486231E+308
Public Const CMaxDouble As Double = 1.79769313486231E+308
Public Const CMinDate As Double = -1.79769313486231E+308
Public Const CMaxDate As Double = 1.79769313486231E+308
Public Const CMinLength As Long = -1
Public Const CMaxLength As Long = 32767
Public Const CInitialUBounddStdDevInput As Long = 10000

Dim GdMax(1 To 3) As Double
Dim GlMaxIdx As Long
Dim GdMin(1 To 3) As Double
Dim GlMinIdx As Long

'We enumerate columns of some worksheets now which means we define
'names for the columns of some sheets we use.
'This is to stay flexible in case we need to delete/move/add a column.
'We can just delete, move or the/a name for that column and our code
'would get adjusted automatically.
'If you hardcode all columns (by referring to them with Range("H:H") for example)
'you would need to look for all locations and to change all these references
'AND other dependent columns.

Enum out_columns  'Structure of sheet NumStats and NumStatsMove
  outLbound = 0  'To be able to iterate from outLBound + 1
  outFile
  outHeader
  outField
  outWarn
  outMin1
  outMin2
  outMin3
  outAverage
  outMax3
  outMax2
  outMax1
  outUbound  'To be able to iterate until outUBound - 1
End Enum  'out_columns

Enum out2_columns  'Structure of sheet TextStats and TestStatsMove
  out2Lbound = 0  'To be able to iterate from out2LBound + 1
  out2File
  out2Header
  out2Field
  out2Warn
  out2Char
  out2CharCount
  out2Len
  out2LenCount
  out2Text
  out2TextCount
  out2Ubound  'To be able to iterate until out2UBound - 1
End Enum  'out2_columns

Enum lim_columns  'Structure of sheets LimitsIn, LimitsOut, LimitsMoveIn, and LimitsMoveOut
  limLbound = 0  'To be able to iterate from limLBound + 1
  limFilePrefix
  limHeader
  limField
  limAllowEmpty  'But we don't set this to True anymore - in Germany "WAHR"
  'won't be treated correctly. So let us use CBool(1)
  limNumMin
  limNumMax
  limDateMin
  limDateMax
  limLenMin
  limLenMax
  limFormula
  limOkFormat
  limOkValues
  limCharSet  '256 fields indicating whether this char may/does appear
  'limUbound 'To be able to iterate until limUBound - 1
End Enum  'lim_columns
```

```
Sub Read_Input()
'This code reads any csv file data and also Algo(rithmics) One input data files and performs
'numerical stats like min, avg, max on numerical fields and on date fields, and frequency
'stats on text fields.
'
'Change History:
'Version Date       Programmer Change
'1       09/10/2016 Bernd      Show Limit File version in tabs, right title row for limits file, and
'                              hint for Formula field
'8       19/01/2019 Bernd      lRowTypes + 2 to correctly skip header rows and use 1 instead of True
'                              for limAllowEmpty because of internationalization, and check for
'                              Datastats subfolders

  Const CsAttrDelim = "|"
  Dim b             As Boolean
  Dim bClearDataAfterRun As Boolean
  Dim bDeletedRecords As Boolean
  Dim bFatal        As Boolean
  Dim bFileSpecs    As Boolean
  Dim bHasEmpty     As Boolean
  Dim bIsIDField    As Boolean
  Dim bLimitsIn     As Boolean
  Dim bLimitsMoveIn As Boolean
  Dim bIsDate       As Boolean
  Dim bHeaderFile   As Boolean
  Dim bMoveCheck    As Boolean
  Dim bNewRecords   As Boolean
  Dim bNoSingleCharCheck As Boolean
  Dim bProcessThisRow As Boolean
  Dim d             As Double
  Dim dMax          As Double
  Dim dMaxDate      As Double
  Dim dMin          As Double
  Dim dMinDate      As Double
  Dim dStdDevThreshold As Double
  Dim dSum          As Double
  Dim dVal          As Double
  Dim dtDate        As Date
  Dim i As Long, j As Long, k As Long, m As Long
  Dim lColumns      As Long
  Dim lColumnStart  As Long
  Dim lCount        As Long
  Dim lCountAll     As Long
  Dim lDel          As Long
  Dim lFileFormat   As Long
  Dim lLen          As Long
  Dim lLenMax       As Long
  Dim lLenMin       As Long
  Dim lLim          As Long
  Dim lLimRow       As Long
  Dim lLMO          As Long
  Dim lMainSortCol  As Long
  Dim lMax          As Long
  Dim lMaxColumns   As Long
  Dim lMaxOkValueCountPerField As Long
  Dim lMin          As Long
  Dim lNew          As Long
  Dim lNSM          As Long
  Dim lOut          As Long
  Dim lOut2         As Long
  Dim lRowTypes     As Long
  Dim lTM           As Long
  Dim lTSM          As Long
  Dim lUBounddStdDevInput As Long
  Dim oOkValues     As Object
  Dim oBASRow       As Object
  Dim oData         As Object
  Dim oFileSpecs    As Object
  Dim oHeader       As Object
  Dim oInputFiles   As Object
  Dim oLimitRow     As Object
  Dim oLimitMoveRow As Object
  Dim oRegEx        As Object
  Dim oRowTypes     As Object
  Dim oSortCol      As Object
  Dim oStringLen    As Object
  Dim oChar         As Object
  Dim s             As String
  Dim sCell         As String
  Dim sCellPrev     As String
  Dim sDelimiter    As String
  Dim sDirData      As String
  Dim sDirDataPrev  As String
  Dim sField        As String
  Dim sFile         As String
  Dim sCoreFile     As String
  Dim sFilePrev     As String
  Dim sFileSpecs    As String
  Dim sFormula      As String
  Dim sFullName     As String
```

```
  Dim sHeader       As String
  Dim Sid           As String
  Dim sID2          As String
  Dim sID3          As String
  Dim sLimitsIn     As String
  Dim sLimitsMoveIn As String
  Dim sMsg          As String
  Dim sReverse      As String
  Dim state         As SystemState
  Dim v             As Variant
  Dim vFile         As Variant
  Dim vID           As Variant
  Dim vPresuffix    As Variant
  Dim vPresuffixPrev As Variant

  sFullName = ActiveWorkbook.FullName
  lFileFormat = ActiveWorkbook.FileFormat
  Set oRegEx = New regexp             'Add reference to Microsoft VBScript Regular Expressions 5.5 or more
  If GLogger Is Nothing Then Start_Log
  GLogger.LogLevel = wsP.Range("Log_Level")
  GLogger.SubName = "Read_Input"
  GLogger.info String(160, ">")
  GLogger.info "Reading Input started with data folder " & ThisWorkbook.Path & "\Data_Files"
  wsDel.Cells.NumberFormat = "@"                      'Treat whole sheet as text
  wsN.Cells.NumberFormat = "@"
  wsT.Cells.NumberFormat = "@"
  wsTP.Cells.NumberFormat = "@"
  sDelimiter = wsP.Range("Input_File_Delimiter")
  If sDelimiter = "" Then sDelimiter = ","

  'Now execute sub Class_Initialize() of class SystemState which stores those system variables
  Set state = New SystemState

  Set oHeader = CreateObject("Scripting.Dictionary")
  Set oInputFiles = CreateObject("Scripting.Dictionary")
  Set oSortCol = CreateObject("Scripting.Dictionary")
  bClearDataAfterRun = Not wsP.Range("Keep_Stats_In_Tabs")
  bNoSingleCharCheck = wsP.Range("NoSingleCharCheck")
  Call Delete_QueryTables

  wsN.Range("1:1048576").Delete
  wsDel.Range("1:1048576").Delete
  On Error GoTo ErrHdl

  'Check for Datastats folder
  If Dir(ThisWorkbook.Path & "\Datastats\", vbDirectory) = "" Then
    GLogger.fatal "Folder " & ThisWorkbook.Path & "\Datastats\" & " does not exist"
    wsW.Select
    Exit Sub
  End If

  'Read in Header info
  'Header files are supposed to be given in this format (header type in first column, subtype in second):
  'BAS DM  Fixed SwapSPEC                              OBJECT      TYPE            IDENTIFIER          NAME
BusDayRuleBUSD  BusDayRuleCONV  BusDayRuleRULE  CouponRatePERD  CouponGenENUM   CurrencyUNIT
DiscountCurveXREF   EffectiveDATE   AccrualDCBasisDAYC  LastRegCouponDATE   MaturityDATE
NotionalAtEndFLAG   OddFirstCouponENUM  PortfolioXREF   PositionUnitsVAL    ProductTypeENUM PayTermNB
PayTermUNIT NotionlAtStartFLAG  Company_SF_ClassificationNAME   Company_SF_Sub_ClassificationNAME
Company_SF_CounterpartyNAME Company_SF_RatingNAME   Company_SF_Attachment_PointNAME
Company_SF_Detachment_PointNAME Company_SF_TenureNAME   Company_Country_CodeNAME
Company_Counterparty_NameNAME   Company_Issuer_NameNAME Company_Level2_ClassificationNAME
Company_RatingNAME  Company_Industry_SectorNAME
  'rm_ro   DM Fixed SwapSPEC : Coupon List ATTRIBUTE   OBJECT      CouponListDATE  CouponListVAL
  'rm_ro   DM Fixed SwapSPEC : Variable Notional       ATTRIBUTE   OBJECT          VariabNotionalDATE
VariabNotionalVAL

  sFile = Dir(ThisWorkbook.Path & "\Header_Files\")
  Do While Len(sFile) > 0
    If FileLen(ThisWorkbook.Path & "\Header_Files\" & sFile) > 0 Then
      Application.StatusBar = "Read_Input: Reading header file " & sFile
      wsT.Range("1:1048576").Delete
      If Not doFileQuery(ThisWorkbook.Path & "\Header_Files\" & sFile, "Temp") Then
        GLogger.fatal "Reading " & sFile & " failed"
        wsW.Select
        Exit Sub
      End If
      wsT.Cells.EntireColumn.AutoFit
      i = 1
      Do While Not IsEmpty(wsT.Cells(i, 1))
        sHeader = wsT.Cells(i, 1).Text & ":" & wsT.Cells(i, 2).Text
        j = 1
        Do While Not IsEmpty(wsT.Cells(i, j))
          oHeader.Item(sHeader & ":" & j) = wsT.Cells(i, j).Text
          j = j + 1
        Loop
        oHeader.Item(sHeader & ":COLUMNS") = j - 1

        If wsT.Cells(i, 1).Text = "BAS" Then
          j = 5
          Do While Not IsEmpty(wsT.Cells(i, j))
```

```
            If wsT.Cells(i, j).Text = "IDENTIFIER" Then
              oSortCol.Item(wsT.Cells(i, 2).Text) = j
              Exit Do
            End If
            j = j + 1
          Loop
          If wsT.Cells(i, j).Text <> "IDENTIFIER" Then
            GLogger.fatal "BAS row " & i & " in header file " & sFile & " has no IDENTIFIER"
            wsW.Select
            Exit Sub
          End If
        ElseIf wsT.Cells(i, 1).Text = "rm_ro" Then
          j = 2
          Do While Not IsEmpty(wsT.Cells(i, j))
            If wsT.Cells(i, j).Text Like "*XREF" Or _
               wsT.Cells(i, j).Text Like "*Coupon List" Or _
               wsT.Cells(i, j).Text Like "*DATE" Or _
               wsT.Cells(i, j).Text Like "*0AXS" Or _
               wsT.Cells(i, j).Text Like "*NODE" Or _
               wsT.Cells(i, j).Text Like "*SEQ" Or _
               wsT.Cells(i, j).Text Like "*IDENTIFIER" Or _
               wsT.Cells(i, j).Text Like "*NB" Then
              oSortCol.Item(wsT.Cells(i, 2).Text) = j
              Exit Do
            End If
            j = j + 1
          Loop
          If CLng(oSortCol.Item(wsT.Cells(i, 2).Text)) = 0 Then
            If j = 6 And wsT.Cells(i, 5).Text Like "*VAL" Then
              oSortCol.Item(wsT.Cells(i, 2).Text) = 5
              GoTo Label_Next_Row_in_Headerfile
            Else
              oSortCol.Item(wsT.Cells(i, 2).Text) = 5
              GLogger.warn "rm_ro row " & i & " in header file " & sFile & _
                           " has no sort column id, column 5 taken"
            End If
          End If
        End If
Label_Next_Row_in_Headerfile:
        i = i + 1
      Loop
      wsT.QueryTables.Item(1).Delete
    Else
      GLogger.warn "Skipping zero file " & sFile
    End If
    sFile = Dir
  Loop

  'Check whether there is a FileSpecs file (a Sort Columns File for "non-Algo One" files!)
  'Expected file format (example, first row is constant title row):
  'Filename Suffix/Prefix   SortColumn_1 SortColumn_2 SortColumn_3
  'OB1_CASHFLOW             2            1            7
  'OB2_TRANS                1            4            7
  'OB7_LPIFWDS              2            5           16
  'SOL2_CREDIT_PROCESS_DATA 5

  sFileSpecs = ThisWorkbook.Path & "\Config\FileSpecs.csv"
  wsFS.Range("1:1048576").Delete
  If sFileSpecs = "" Then
    bFileSpecs = False
  ElseIf Len(Dir(sFileSpecs)) = 0 Then
    GLogger.info "File Specs file cannot be read. No individual file checks nor " & _
                 "previous period comparison"
    bFileSpecs = False
  ElseIf doFileQuery(sFileSpecs, "FileSpecs") Then
    bFileSpecs = True
    Set oFileSpecs = CreateObject("Scripting.Dictionary")
    i = 2
    Do While Not IsEmpty(wsFS.Cells(i, 1))
      j = 2
      Do While Not IsEmpty(wsFS.Cells(i, j))
        oFileSpecs.Item(wsFS.Cells(i, 1).Text & ":" & j) = CLng(wsFS.Cells(i, j))
        j = j + 1
      Loop
      i = i + 1
    Loop
  Else
    GLogger.fatal "Reading " & sFileSpecs & " failed"
    wsW.Select
    Exit Sub
  End If

  'Check whether there is a limits input file
  sLimitsIn = ThisWorkbook.Path & "\Config\Limits_Input.csv"
  lMaxOkValueCountPerField = wsP.Range("Max_Attr_Count_per_Field").Text
  wsLI.Range("1:1048576").Delete
  If sLimitsIn = "" Then
    bLimitsIn = False
  ElseIf Len(Dir(sLimitsIn)) = 0 Then
    GLogger.info "Input Limits file cannot be read. No input limits check"
```

```
    bLimitsIn = False
  ElseIf doFileQuery(sLimitsIn, "LimitsIn") Then
    If wsLI.Cells(1, 1).Text <> CLimFileVersion Then
      GLogger.fatal "Limits Input File version '" & CLimFileVersion & "' not compatible with " & _
                    "program version '" & wsLI.Cells(1, 1).Text & "'"
      wsW.Select
      Exit Sub
    End If
    bLimitsIn = True
    Set oLimitRow = CreateObject("Scripting.Dictionary")
    i = 3
    Do While Not IsEmpty(wsLI.Cells(i, limFilePrefix))
      oLimitRow.Item(wsLI.Cells(i, limFilePrefix).Text & ":" & _
                     wsLI.Cells(i, limField).Text) = i
      i = i + 1
    Loop
  Else
    GLogger.fatal "Reading " & sLimitsIn & " failed"
    wsW.Select
    Exit Sub
  End If

  'Check whether there is a limits move input file (containing movement limits over periods)
  sLimitsMoveIn = ThisWorkbook.Path & "\Config\Limits_Move_Input.csv"
  wsLMI.Range("1:1048576").Delete
  If sLimitsMoveIn = "" Then
    bLimitsMoveIn = False
  ElseIf Len(Dir(sLimitsMoveIn)) = 0 Then
    GLogger.info "Input Limits Move file cannot be read. No input limits move check"
    bLimitsMoveIn = False
  ElseIf doFileQuery(sLimitsMoveIn, "LimitsMoveIn") Then
    If wsLMI.Cells(1, 1).Text <> CLimMovFileVersion Then
      GLogger.fatal "Limits Move Input File version '" & _
                    CLimMovFileVersion & "' not compatible with " & _
                    "program version '" & wsLMI.Cells(1, 1).Text & "'"
      wsW.Select
      Exit Sub
    End If
    bLimitsMoveIn = True
    Set oLimitMoveRow = CreateObject("Scripting.Dictionary")
    i = 3
    Do While Not IsEmpty(wsLMI.Cells(i, limFilePrefix))
      oLimitMoveRow.Item(wsLMI.Cells(i, limFilePrefix).Text & ":" & _
                         wsLMI.Cells(i, limField).Text) = i
      i = i + 1
    Loop
  Else
    GLogger.fatal "Reading " & sLimitsMoveIn & " failed"
    wsW.Select
    Exit Sub
  End If

  'Read in data files
  sDirData = ThisWorkbook.Path & "\Data_Files\"
  sDirDataPrev = ThisWorkbook.Path & "\Data_Files_Prev\"

  dStdDevThreshold = wsP.Range("StdDevThreshold")

  wsNS.Range("1:1048576").Delete
  wsNS.Range(wsNS.Cells(1, outLbound + 1), _
             wsNS.Cells(1, outUbound - 1)).FormulaArray = _
                                                          Array("Filename", "RowType", _
                                                                "Fieldname", "Warning", "Min 1", _
                                                                "Min 2", "Min 3", "Average", _
                                                                "Max 3", "Max 2", "Max 1")
  lOut = 2

  wsNSM.Range("1:1048576").Delete
  wsNSM.Range(wsNSM.Cells(1, outLbound + 1), _
              wsNSM.Cells(1, outUbound - 1)).FormulaArray = _
                                                            Array("Filename", "RowType", _
                                                                  "Fieldname", "Warning", "Min 1", _
                                                                  "Min 2", "Min 3", "Average", _
                                                                  "Max 3", "Max 2", "Max 1")
  lNSM = 2

  wsLO.Range("1:1048576").Delete
  wsLO.Cells(1, 1) = CLimFileVersion
  wsLO.Range(wsLO.Cells(2, limLbound + 1), _
             wsLO.Cells(2, limCharSet - 1)).FormulaArray = _
                                                           Array("Filename", "RowType", _
                                                                 "Fieldname", "Allow Empty", _
                                                                 "Min", "Max", "Min Date", "Max Date", _
                                                                 "Length Min", "Length Max", _
                                                                 "Formula", "OkFormat", "OkValues")
  For i = 0 To 255
    wsLO.Cells(2, limCharSet + i) = "'CHAR(" & Format(i, "000") & ") = '" & Chr(i) & "'"
  Next i
  wsLO.Cells(2, limCharSet + 10) = "'CHAR(010)"       'Otherwise you will face a surprise if you read in
the file
```

```
wsLO.Cells(2, limCharSet + 13) = "'CHAR(013)"
lLim = 3

wsLMO.Range("1:1048576").Delete
wsLMO.Cells(1, 1) = CLimMovFileVersion
wsLMO.Range(wsLMO.Cells(2, limLbound + 1), _
            wsLMO.Cells(2, limLenMax)).FormulaArray = _
                                                      Array("Filename", "RowType", "Fieldname", _
                                                            "Allow Empty", "Min", "Max", "Min Date", _
                                                            "Max Date", "Length Min", "Length Max")
lLMO = 3

wsTS.Range("1:1048576").Delete
wsTS.Range(wsTS.Cells(1, out2Lbound + 1), _
           wsTS.Cells(1, out2Ubound - 1)).FormulaArray = _
                                                         Array("Filename", "RowType", _
                                                               "Fieldname", "Warning", "Char", _
                                                               "Char Frequency", "Text Length", _
                                                               "Frequency", "Text", "Count")
lOut2 = 2

wsTSM.Range("1:1048576").Delete
wsTSM.Range(wsTSM.Cells(1, out2Lbound + 1), _
            wsTSM.Cells(1, out2Ubound - 1)).FormulaArray = _
                                                           Array("Filename", "RowType", _
                                                                 "Fieldname", "Warning", "Char", _
                                                                 "Char Frequency", "Text Length", _
                                                                 "Frequency", "Text", "Count")
lTSM = 2

lNew = 1
lDel = 1

sFile = Dir(sDirData)
Do While Len(sFile) > 0
  oInputFiles.Item(sFile) = 1
  sFile = Dir()
Loop

'Find out common prefixes and suffixes of all (previous) data files
vPresuffix = Files_PrefixStart_SuffixEnd(sDirData)
vPresuffixPrev = Files_PrefixStart_SuffixEnd(sDirDataPrev)
'Kludge for "_YYYYMMDD.csv" resp. ".???" suffix
If Right(vPresuffix(3), 13) Like "_[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9].???" Then
  vPresuffix(3) = Right(vPresuffix(3), 13)
  vPresuffix(1) = -13
ElseIf Right(vPresuffix(3), 4) Like ".???" Then
  vPresuffix(3) = Right(vPresuffix(3), 4)
  vPresuffix(1) = -4
End If
If Right(vPresuffixPrev(3), 13) Like "_[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9].???" Then
  vPresuffixPrev(3) = Right(vPresuffixPrev(3), 13)
  vPresuffixPrev(1) = -13
ElseIf Right(vPresuffixPrev(3), 4) Like ".???" Then
  vPresuffixPrev(3) = Right(vPresuffixPrev(3), 4)
  vPresuffixPrev(1) = -4
End If
'Kludge for "YYYYMMDD_" prefix
If Right(vPresuffix(2), 9) Like "[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]_" Then
  vPresuffix(2) = Left(vPresuffix(2), 9)
  vPresuffix(0) = 9
End If
If Right(vPresuffixPrev(2), 9) Like "[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]_" Then
  vPresuffixPrev(2) = Left(vPresuffixPrev(2), 9)
  vPresuffixPrev(0) = 9
End If

For Each vFile In oInputFiles.keys()
  If FileLen(sDirData & vFile) > 0 Then
    Application.StatusBar = "Read_Input: Reading data file " & vFile
    'GLogger.info "Reading " & vFile
    wsT.Range("1:1048576").Delete
    If doFileQuery(sDirData & vFile, "Temp", sDelimiter) Then
      GLogger.info "Reading " & sDirData & vFile & " finished: " & _
                   wsT.Cells(1000000, 1).End(xlUp).Row & " rows read"
    Else
      GLogger.fatal "Reading " & sDirData & vFile & " failed"
      wsW.Select
      Exit Sub
    End If
    wsT.Cells.EntireColumn.AutoFit

    sCoreFile = Mid(vFile, vPresuffix(0) + 1, Len(vFile) + vPresuffix(1) - vPresuffix(0))

    'First collect all different kind of row types (BAS, RO, ...)
    i = 1
    sHeader = wsT.Cells(i, 1).Text & ":" & wsT.Cells(i, 2).Text & ":5"
    If oHeader.Item(sHeader) = "" Then
      'No header file - header info is in first row
      bHeaderFile = False
```

```
    j = 1
    Do While Not IsEmpty(wsT.Cells(1, j))
      oHeader.Item(CStr(j)) = wsT.Cells(1, j).Text
      j = j + 1
    Loop
    oHeader.Item(CStr(j)) = ""
    lColumns = j - 1
    lColumnStart = 1
    lRowTypes = 0
    bProcessThisRow = True
  Else
    'We do have a header file
    bHeaderFile = True
    lColumnStart = 5
    Set oRowTypes = CreateObject("Scripting.Dictionary")
    Do While Not IsEmpty(wsT.Cells(i, 1))
      oRowTypes.Item(wsT.Cells(i, 1).Text & ":" & wsT.Cells(i, 2).Text) = 1
      i = i + 1
    Loop
    lRowTypes = oRowTypes.Count - 1
  End If

  'Now iterate over a) row types, b) columns, and c) rows
  lMaxColumns = 0
  For k = 0 To lRowTypes
    If bHeaderFile Then
      lColumns = oHeader.Item(oRowTypes.keys()(k) & ":COLUMNS")
      If lColumns = 0 Then
        GLogger.fatal "File " & vFile & ", row type '" & _
                      oRowTypes.keys()(k) & "' not found in header"
        wsW.Select
        Exit Sub
      End If
    End If
    If lMaxColumns < lColumns Then lMaxColumns = lColumns
    For j = lColumnStart To lColumns
      Set oOkValues = CreateObject("Scripting.Dictionary")
      Set oData = CreateObject("Scripting.Dictionary")
      Set oStringLen = CreateObject("Scripting.Dictionary")
      Set oChar = CreateObject("Scripting.Dictionary")
      dSum = 0#
      sMsg = ""
      lCount = 0
      lUBounddStdDevInput = CInitialUBounddStdDevInput
      ReDim dStdDevInput(1 To lUBounddStdDevInput) As Double
      lCountAll = 0
      bIsDate = False
      For i = 1 To 3
        GdMax(i) = -1E+308
        GdMin(i) = 1E+308
      Next i
      GlMaxIdx = 0
      GlMinIdx = 0
      If bHeaderFile Then
        i = 1
      Else
        i = 2
      End If

      dMin = CMaxDouble
      dMax = CMinDouble
      dMinDate = CMaxDate
      dMaxDate = CMinDate
      lLenMin = CMaxLength
      lLenMax = CMinLength
      bHasEmpty = False

      If Not bNoSingleCharCheck Then
        ReDim bCharSet(0 To 255) As Boolean
      End If

      Do While Not (IsEmpty(wsT.Cells(i, 1)) And IsEmpty(wsT.Cells(i, 1).End(xlToRight)))
        bIsIDField = False
        If bHeaderFile Then
          sHeader = wsT.Cells(i, 1).Text & ":" & _
                    wsT.Cells(i, 2).Text
          bProcessThisRow = (sHeader = oRowTypes.keys()(k))
          sField = oHeader.Item(oRowTypes.keys()(k) & ":" & j)
        Else
          sField = oHeader.Item(CStr(j))
        End If
        If bProcessThisRow Then
          lCountAll = lCountAll + 1
          sCell = wsT.Cells(i, j).Text
          If sCell = "" Then
            bHasEmpty = True
            If bLimitsIn Then
              lLimRow = oLimitRow.Item(sCoreFile & ":" & sField)
              If lLimRow > 0 Then
                If Not CBool(wsLI.Cells(lLimRow, limAllowEmpty)) Then
```

```
        GLogger.warn "File " & vFile & ", Field " & _
                     sField & ", Row " & i & ", should not be empty"
      End If
    End If
  End If
End If
If Not bIsIDField Then
  If IsNumeric(sCell) Then
    dVal = CDbl(sCell)
    If dVal > dMax Then dMax = dVal
    If dVal < dMin Then dMin = dVal
    dSum = dSum + dVal
    lCount = lCount + 1
    dStdDevInput(lCount) = dVal
    Call Extreme(dVal)
    If bLimitsIn Then
      lLimRow = oLimitRow.Item(sCoreFile & ":" & sField)
      If lLimRow > 0 Then
        'Format check
        If Not IsEmpty(wsLI.Cells(lLimRow, limOkFormat)) Then
          oRegEx.Pattern = wsLI.Cells(lLimRow, limOkFormat).Text
          If Not oRegEx.Test(sCell) Then
            GLogger.warn "File " & vFile & ", Field " & sField & _
                         ", Row " & i & _
                         ", Value '" & sCell & "' not of format " & _
                         wsLI.Cells(lLimRow, limOkFormat).Text
          End If
        End If
        If Not IsEmpty(wsLI.Cells(lLimRow, limNumMin)) Then
          If dVal < wsLI.Cells(lLimRow, limNumMin) Then
            GLogger.warn "File " & vFile & ", Field " & _
                         sField & ", Row " & i & ", Value " & dVal & _
                         " < " & wsLI.Cells(lLimRow, limNumMin) & _
                         " [Min Limit]"
          End If
        End If
        If Not IsEmpty(wsLI.Cells(lLimRow, limNumMax)) Then
          If dVal > wsLI.Cells(lLimRow, limNumMax) Then
            GLogger.warn "File " & vFile & ", Field " & sField & _
                         ", Row " & i & ", Value " & dVal & " > " & _
                         wsLI.Cells(lLimRow, limNumMax) & " [Max Limit]"
          End If
        End If
      End If
    End If
  ElseIf IsDate(sCell) Then
    bIsDate = True
    dtDate = CDate(sCell)
    dVal = CDbl(dtDate)
    If dVal > dMaxDate Then dMaxDate = dVal
    If dVal < dMinDate Then dMinDate = dVal
    dSum = dSum + dVal
    lCount = lCount + 1
    dStdDevInput(lCount) = dVal
    Call Extreme(dVal)
    If bLimitsIn Then
      lLimRow = oLimitRow.Item(sCoreFile & ":" & sField)
      If lLimRow > 0 Then
        If Not IsEmpty(wsLI.Cells(lLimRow, limDateMin)) Then
          'Format check
          If Not IsEmpty(wsLI.Cells(lLimRow, limOkFormat)) Then
            oRegEx.Pattern = wsLI.Cells(lLimRow, limOkFormat).Text
            If Not oRegEx.Test(sCell) Then
              GLogger.warn "File " & vFile & ", Field " & sField & _
                           ", Row " & i & _
                           ", Value '" & sCell & "' not of format " & _
                           wsLI.Cells(lLimRow, limOkFormat).Text
            End If
          End If
          If dVal < CDbl(CDate(wsLI.Cells(lLimRow, limDateMin).Text)) Then
            GLogger.warn "File " & vFile & ", Field " & sField & _
                         ", Row " & i & ", Date " & Format(dVal, "dd-mmm-yyyy") & _
                         " < " & Format(wsLI.Cells(lLimRow, limDateMin), _
                                        "dd-mmm-yyyy") & " [Min Limit]"
          End If
        End If
        If Not IsEmpty(wsLI.Cells(lLimRow, limDateMax)) Then
          If dVal > CDbl(CDate(wsLI.Cells(lLimRow, limDateMax).Text)) Then
            GLogger.warn "File " & vFile & ", Field " & sField & _
                         ", Row " & i & ", Date " & Format(dVal, "dd-mmm-yyyy") & _
                         " > " & Format(wsLI.Cells(lLimRow, limDateMax), _
                                        "dd-mmm-yyyy") & " [Max Limit]"
          End If
        End If
      End If
    End If
  Else
    'String
    If sCell <> "" Then
      If oOkValues.Count <= lMaxOkValueCountPerField Then
```

```
            oOkValues.Item(sCell) = oOkValues.Item(sCell) + 1
          End If
        End If
        oData.Item("'" & sCell & "'") = _
                                        oData.Item("'" & sCell & "'") + 1
        lLen = Len(sCell)
        oStringLen.Item(CStr(lLen)) = oStringLen.Item(CStr(lLen)) + 1
        If lLen > 0 Then
          If lLen > lLenMax Then lLenMax = lLen
          If lLen < lLenMin Then lLenMin = lLen
          For m = 1 To lLen
            s = Mid(sCell, m, 1)
            If Not bNoSingleCharCheck Then bCharSet(Asc(s)) = True
            Sid = "CHAR(" & Format(Asc(s), "000") & ") = '" & s & "'"
            oChar.Item(Sid) = oChar.Item(Sid) + 1
          Next m
        End If
        If bLimitsIn Then
          lLimRow = oLimitRow.Item(sCoreFile & ":" & sField)
          If lLimRow > 0 Then
            If Not CBool(wsLI.Cells(lLimRow, limAllowEmpty)) And _
               Not IsEmpty(wsLI.Cells(lLimRow, limLenMin)) Then
              If lLen < wsLI.Cells(lLimRow, limLenMin).Value Then
                GLogger.warn "File " & vFile & ", Field " & sField & _
                             ", Row " & i & _
                             ", Length('" & sCell & "') = " & lLen & " < " & _
                             wsLI.Cells(lLimRow, limLenMin) & _
                             " [Min Length]"
              End If
            End If
            If Not IsEmpty(wsLI.Cells(lLimRow, limLenMax)) Then
              If lLen > wsLI.Cells(lLimRow, limLenMax).Value Then
                GLogger.warn "File " & vFile & ", Field " & sField & _
                             ", Row " & i & _
                             ", Length('" & sCell & "') = " & lLen & " > " & _
                             wsLI.Cells(lLimRow, limLenMax) & _
                             " [Max Length]"
              End If
            End If
            If Not bNoSingleCharCheck Then
              For m = 1 To lLen
                s = Mid(sCell, m, 1)
                If Not CBool(wsLI.Cells(lLimRow, limCharSet + Asc(s))) Then
                  GLogger.warn "File " & vFile & ", Field " & sField & _
                               ", Row " & i & _
                               ", Character " & m & " = '" & s & "' not legal"
                End If
              Next m
            End If
            If sCell <> "" Then
              'Format check
              If Not IsEmpty(wsLI.Cells(lLimRow, limOkFormat)) Then
                oRegEx.Pattern = wsLI.Cells(lLimRow, limOkFormat).Text
                If Not oRegEx.Test(sCell) Then
                  GLogger.warn "File " & vFile & ", Field " & sField & _
                               ", Row " & i & _
                               ", Value '" & sCell & "' not of format " & _
                               wsLI.Cells(lLimRow, limOkFormat).Text
                End If
              End If
              'Values check
              If Not IsEmpty(wsLI.Cells(lLimRow, limOkValues)) Then
                If InStr(CsAttrDelim & wsLI.Cells(lLimRow, limOkValues) & _
                         CsAttrDelim, sCell) = 0 Then
                  GLogger.warn "File " & vFile & ", Field " & sField & _
                               ", Row " & i & ", Value '" & sCell & _
                               "' not in Ok values list of limits file"
                End If
              End If
            End If                          'If sCell <> ""
          End If                            'If lLimRow > 0
        End If                              'If bLimitsIn
      End If                                'If IsNumeric(sCell) ... ElseIf ... Else ...
    End If                                  'If Not bIsIDField
  End If                                    'If bProcessThisRow
  i = i + 1
Loop

If bIsIDField Then
  wsTS.Cells(lOut2, out2File) = vFile
  If bHeaderFile Then
    wsTS.Cells(lOut2, out2Header) = oRowTypes.keys()(k)
    wsTS.Cells(lOut2, out2Field) = oHeader.Item(oRowTypes.keys()(k) & ":" & j)
  Else
    wsTS.Cells(lOut2, out2Field) = oHeader.Item(CStr(j))
  End If
  wsTS.Cells(lOut2, out2Warn) = "No stats - this field is an identifier field"
  lOut2 = lOut2 + 1
Else
  'Deal with numeric values
```

```
If lCount > 0 Then
  wsNS.Cells(lOut, outFile) = vFile
  If bHeaderFile Then
    wsNS.Cells(lOut, outHeader) = oRowTypes.keys()(k)
  End If
  wsNS.Cells(lOut, outField) = sField
  wsNS.Cells(lOut, outAverage) = dSum / lCount
  ReDim Preserve dStdDevInput(1 To lCount) As Double  'Reduce dStdDevInput size to used values
  If lCountAll > lCount Then
    sMsg = sMsg & lCountAll - lCount & " values of " & lCountAll & " are non-numeric. "
  End If
  d = 0#
  On Error Resume Next
  d = Application.WorksheetFunction.StDevP(dStdDevInput)
  On Error GoTo ErrHdl
  If d > 0# Then
    If Abs(dSum / lCount - GdMin(1)) > dStdDevThreshold * d Then
      sMsg = sMsg & "Min is off by more than " & dStdDevThreshold & " stdev. "
    End If
    If Abs(dSum / lCount - GdMax(1)) > dStdDevThreshold * d Then
      sMsg = sMsg & "Max is off by more than " & dStdDevThreshold & " stdev. "
    End If
  End If
  wsNS.Cells(lOut, outWarn) = sMsg
  sMsg = ""
  If GlMinIdx > 2 Then wsNS.Cells(lOut, outMin1) = GdMin(1)
  If GlMinIdx > 1 Then wsNS.Cells(lOut, outMin2) = GdMin(2 + (GlMinIdx < 3))
  If GlMinIdx > 0 Then wsNS.Cells(lOut, outMin3) = GdMin(3 + (GlMinIdx < 3) + (GlMinIdx < 2))
  If GlMaxIdx > 2 Then wsNS.Cells(lOut, outMax1) = GdMax(1)
  If GlMaxIdx > 1 Then wsNS.Cells(lOut, outMax2) = GdMax(2 + (GlMaxIdx < 3))
  If GlMaxIdx > 0 Then wsNS.Cells(lOut, outMax3) = GdMax(3 + (GlMaxIdx < 3) + (GlMaxIdx < 2))
  If bIsDate Then
    Range(wsNS.Cells(lOut, outMin1), wsNS.Cells(lOut, outMax1)).NumberFormat = "dd-mmm-yyyy"
  End If
  lOut = lOut + 1
End If

'Deal with string values
If lCount < lCountAll Then
  wsTS.Cells(lOut2, out2File) = vFile
  If bHeaderFile Then
    wsTS.Cells(lOut2, out2Header) = oRowTypes.keys()(k)
    wsTS.Cells(lOut2, out2Field) = oHeader.Item(oRowTypes.keys()(k) & ":" & j)
  Else
    wsTS.Cells(lOut2, out2Field) = oHeader.Item(CStr(j))
  End If
  If lCount > 0 Then
    sMsg = sMsg & lCount & " values of " & lCountAll & " are numeric. "
  End If
  If oData.Count > lCountAll - 5 Or (oData.Count / lCountAll > 0.5 And _
                                     lCountAll > 10) Then
    For m = oData.Count - 1 To 0 Step -1
      If oData.keys()(m) <> "'''" And oData.keys()(m) <> "'' '" _
         And oData.Item(oData.keys()(m)) = 1 Then
        oData.Remove oData.keys()(m)
      End If
    Next m
    sMsg = sMsg & lCountAll - oData.Count & " strings occurring once not shown. "
  End If
  wsTS.Cells(lOut2, out2Warn) = sMsg
  sMsg = ""
  If oData.Count > 0 Then
    On Error Resume Next
    wsTS.Cells(lOut2, _
               out2Text).Resize(oData.Count, 2) = _
                                                  Application.WorksheetFunction.Transpose( _
                                                    Array(oData.keys, oData.items))
    On Error GoTo ErrHdl
  End If
  If oStringLen.Count > 0 Then
    On Error Resume Next
    wsTS.Cells(lOut2, _
               out2Len).Resize(oStringLen.Count, 2) = _
                                                 Application.WorksheetFunction.Transpose( _
                                                 Array(oStringLen.keys, oStringLen.items))
    On Error GoTo ErrHdl
  End If

  'Now print rarest occurring character
  lMin = 1000000
  Sid = ""
  For m = 0 To oChar.Count - 1
    If oChar.items()(m) < lMin Then
      lMin = oChar.items()(m)
      Sid = oChar.keys()(m)
    End If
  Next m
  wsTS.Cells(lOut2, out2Char) = Sid
  If lMin < 1000000 Then wsTS.Cells(lOut2, out2CharCount) = lMin
```

```
        'Increase row counter
        lMax = 1
        If lMax < oData.Count Then lMax = oData.Count
        If lMax < oStringLen.Count Then lMax = oStringLen.Count
        lOut2 = lOut2 + lMax
      End If

      'Update Limits Out sheet
      wsLO.Cells(lLim, limFilePrefix) = sCoreFile
      If bHeaderFile Then
        wsLO.Cells(lLim, limHeader) = oRowTypes.keys()(k)
        wsLO.Cells(lLim, limField) = oHeader.Item(oRowTypes.keys()(k) & ":" & j)
      Else
        wsLO.Cells(lLim, limField) = oHeader.Item(CStr(j))
      End If
      If bHasEmpty Then wsLO.Cells(lLim, limAllowEmpty) = 1  'True
      If dMin < CMaxDouble Then wsLO.Cells(lLim, limNumMin) = dMin
      If dMax > CMinDouble Then wsLO.Cells(lLim, limNumMax) = dMax
      If dMinDate < CMaxDate Then
        wsLO.Cells(lLim, limDateMin) = dMinDate
        wsLO.Cells(lLim, limDateMin).NumberFormat = "dd-mmm-yyyy"
      End If
      If dMaxDate > CMinDate Then
        wsLO.Cells(lLim, limDateMax) = dMaxDate
        wsLO.Cells(lLim, limDateMax).NumberFormat = "dd-mmm-yyyy"
      End If
      If lLenMin < CMaxLength Then wsLO.Cells(lLim, limLenMin) = lLenMin
      If lLenMax > CMinLength Then wsLO.Cells(lLim, limLenMax) = lLenMax
      If oOkValues.Count > 0 And oOkValues.Count <= lMaxOkValueCountPerField Then
        wsLO.Cells(lLim, limOkValues) = sbCat(Array(oOkValues.keys)(0), CsAttrDelim)
      End If
      'Now we protect "True" from becoming "TRUE" etc. if reloaded from csv later
      If Not IsEmpty(wsLO.Cells(lLim, limOkValues)) And _
         Right(wsLO.Cells(lLim, limOkValues), 1) <> CsAttrDelim Then
        wsLO.Cells(lLim, limOkValues) = wsLO.Cells(lLim, limOkValues) & CsAttrDelim
      End If
      If Not bNoSingleCharCheck Then
        For m = 0 To 255
          If bCharSet(m) Then wsLO.Cells(lLim, limCharSet + m) = True
        Next m
      End If
      lLim = lLim + 1

    End If                                        'If bIsIDField

    Set oChar = Nothing
    Set oData = Nothing
    Set oOkValues = Nothing
    Set oStringLen = Nothing

  Next j

Next k

'Now formula checks - we need to iterate through rows first, then columns
If bHeaderFile Then
  i = 1
  lColumnStart = 5
Else
  i = 2
  lColumnStart = 1
End If
Do While Not (IsEmpty(wsT.Cells(i, 1)) And IsEmpty(wsT.Cells(i, 1).End(xlToRight)))
  If bHeaderFile Then
    lColumns = oHeader.Item(wsT.Cells(i, 1).Text & ":" & wsT.Cells(i, 2).Text & ":COLUMNS")
    Debug.Assert lColumns > 0
    If wsT.Cells(i, 1).Text = "BAS" Then
      Set oBASRow = Nothing
      Set oBASRow = CreateObject("Scripting.Dictionary")
      For j = lColumnStart To lColumns
        oBASRow.Item(oHeader.Item(sHeader & ":" & j)) = wsT.Cells(i, j).Text
      Next j
    End If
  Else
    Set oBASRow = Nothing
    Set oBASRow = CreateObject("Scripting.Dictionary")
    For j = lColumnStart To lColumns
      oBASRow.Item(CStr(oHeader.Item(CStr(j)))) = wsT.Cells(i, j).Text
    Next j
  End If
  For j = lColumnStart To lColumns
    'Formula check
    If bHeaderFile Then
      sHeader = wsT.Cells(i, 1).Text & ":" & wsT.Cells(i, 2).Text
      sField = oHeader.Item(sHeader & ":" & j)
    Else
      sField = oHeader.Item(CStr(j))
    End If
    If bLimitsIn Then
      lLimRow = oLimitRow.Item(sCoreFile & ":" & sField)
```

```
      If lLimRow > 0 Then
        If Not IsEmpty(wsLI.Cells(lLimRow, limFormula)) Then
          oBASRow.Item("ThisCell") = wsT.Cells(i, j).Text
          'Parameter substitution in formula
          sFormula = wsLI.Cells(lLimRow, limFormula).Text
          m = InStr(sFormula, "[")
          Do While m > 0
            k = InStr(sFormula, "]")
            If k <= m Then
              GLogger.fatal "File " & vFile & ", Field " & sField & _
                            ", Row " & i & ", Formula '" & _
                            wsLI.Cells(lLimRow, limFormula).Text & _
                            "' has nonmatching parameter brackets []"
              wsW.Select
              Exit Sub
            End If
            s = Mid(sFormula, m + 1, k - m - 1)
            If oBASRow.exists(s) Then
              sFormula = Replace(sFormula, "[" & s & "]", """" & oBASRow.Item(s) & """")
            Else
              GLogger.fatal "File " & vFile & ", Field " & sField & _
                            ", Row " & i & ", Formula '" & _
                            wsLI.Cells(lLimRow, limFormula).Text & _
                            "', Param [" & s & "] is not a valid field name"
              wsW.Select
              Exit Sub
            End If
            m = InStr(sFormula, "[")
          Loop
          'Formula evaluation
          On Error Resume Next
          If Not Application.Evaluate(sFormula) Then
            If Err.Number <> 0 Then
              GLogger.fatal "File " & vFile & ", Field " & sField & _
                            ", Row " & i & _
                            ", syntax error " & Err.Number & " in Limits formula '" & _
                            wsLI.Cells(lLimRow, limFormula).Text & "' = '" & sFormula & "'"
              GLogger.fatal "Please note: Application.International(xlMDY) = " & _
                            Application.International(xlMDY)
              On Error GoTo 0
              wsW.Select
              Exit Sub
            Else
              On Error GoTo ErrHdl
              GLogger.warn "File " & vFile & ", Field " & sField & _
                           ", Row " & i & _
                           ", Value '" & wsT.Cells(i, j).Text & "' does not pass formula '" & _
                           wsLI.Cells(lLimRow, limFormula).Text & "' = '" & sFormula & "'"
            End If
          End If
          On Error GoTo ErrHdl
        End If
      End If
    End If
  Next j
  i = i + 1
Loop

wsTM.Range("1:1048576").Delete

'Movement Checks
bMoveCheck = True
If bMoveCheck Then
  sFilePrev = Dir(sDirDataPrev & vPresuffixPrev(2) & sCoreFile & vPresuffixPrev(3))
  If sFilePrev = "" Then
    bMoveCheck = False
    GLogger.warn "Filename " & sDirDataPrev & vPresuffixPrev(2) & sCoreFile & _
                 vPresuffixPrev(3) & " not found in Data_Files_Prev. No movement check."
  End If
End If
If bMoveCheck Then
  If FileLen(sDirDataPrev & sFilePrev) > 0 Then
    Application.StatusBar = "Read_Input: Reading previous data file " & sFilePrev
    wsTP.Range("1:1048576").Delete
    If doFileQuery(sDirDataPrev & sFilePrev, wsTP.Name, sDelimiter) Then
      GLogger.info "Reading " & sDirDataPrev & sFilePrev & " finished: " & _
                   wsTP.Cells(1000000, 1).End(xlUp).Row & " rows read"
    Else
      GLogger.fatal "Reading " & sDirDataPrev & sFilePrev & " failed"
      wsW.Select
      Exit Sub
    End If
    wsTP.Cells.EntireColumn.AutoFit
    bDeletedRecords = False
    bNewRecords = False
    'Prepare both temp sheets for comparison
    'Temp sheet first
    lMainSortCol = lMaxColumns + 1
    If bHeaderFile Then
      Sid = ""
```

```
    sCellPrev = ":"
    i = 1
    Do While Not (IsEmpty(wsT.Cells(i, 2)) And IsEmpty(wsT.Cells(i, 2).End(xlToRight)))
      sCell = wsT.Cells(i, 2)
      If InStr(sCell, ":") = 0 Then
        Sid = wsT.Cells(i, CLng(oSortCol.Item(wsT.Cells(i, 2).Text))).Text
        sID2 = "_BAS_"
        sID3 = "_BAS_"
      ElseIf sCell <> sCellPrev Then
        sID2 = wsT.Cells(i, 3)
        sID3 = wsT.Cells(i, CLng(oSortCol.Item(wsT.Cells(i, 2).Text))).Text
      Else
        sID3 = wsT.Cells(i, CLng(oSortCol.Item(wsT.Cells(i, 2).Text))).Text
      End If
      wsT.Cells(i, lMainSortCol) = Sid & "|" & sID2 & "|" & sID3
      sCellPrev = sCell
      i = i + 1
    Loop

    wsT.Sort.SortFields.Clear
    wsT.Sort.SortFields.Add _
        Key:=Range(wsT.Cells(1, lMainSortCol), _
                   wsT.Cells(i, lMainSortCol)), _
        SortOn:=xlSortOnValues, _
        Order:=xlAscending, DataOption:=xlSortNormal
    With wsT.Sort
      .SetRange Range(wsT.Cells(1, 1), wsT.Cells(i, lMainSortCol))
      .Header = xlNo
      .Apply
    End With

    'Now TempPrev sheet
    Sid = ""
    sCellPrev = ":"
    i = 1
    Do While Not (IsEmpty(wsTP.Cells(i, 2)) And IsEmpty(wsTP.Cells(i, 2).End(xlToRight)))
      sCell = wsTP.Cells(i, 2)
      If InStr(sCell, ":") = 0 Then
        Sid = wsTP.Cells(i, CLng(oSortCol.Item(wsTP.Cells(i, 2).Text))).Text
        sID2 = "_BAS_"
        sID3 = "_BAS_"
      ElseIf sCell <> sCellPrev Then
        sID2 = wsTP.Cells(i, 3)
        sID3 = wsTP.Cells(i, CLng(oSortCol.Item(wsTP.Cells(i, 2).Text))).Text
      Else
        sID2 = wsTP.Cells(i, 3)
        sID3 = wsTP.Cells(i, CLng(oSortCol.Item(wsTP.Cells(i, 2).Text))).Text
      End If
      wsTP.Cells(i, lMainSortCol) = Sid & "|" & sID2 & "|" & sID3
      sCellPrev = sCell
      i = i + 1
    Loop
    wsT.Cells.EntireColumn.AutoFit

    wsTP.Sort.SortFields.Clear
    wsTP.Sort.SortFields.Add Key:=Range(wsTP.Cells(1, lMainSortCol), _
                                        wsTP.Cells(i, lMainSortCol)), _
                             SortOn:=xlSortOnValues, Order:=xlAscending, _
                             DataOption:=xlSortNormal
    With wsTP.Sort
      .SetRange Range(wsTP.Cells(1, 1), wsTP.Cells(i, lMainSortCol))
      .Header = xlNo
      .Apply
    End With
    wsTP.Cells.EntireColumn.AutoFit
    For i = 0 To lRowTypes
      For j = 1 To oHeader.Item(oRowTypes.keys()(i) & ":COLUMNS")
        wsTM.Cells(i + 1, j) = oHeader.Item(oRowTypes.keys()(i) & ":" & j)
      Next j
    Next i
    lTM = i + 1
  Else
    'Check whether field names of "non-Algo One" files have changed - yes: error!
    j = 1
    bFatal = False
    Do While Not IsEmpty(wsTP.Cells(1, j))
      If oHeader.Item(CStr(j)) <> wsTP.Cells(1, j).Text Then
        bFatal = True
        GLogger.fatal "Name of field " & j & " has changed: " & vFile & _
                      "="""" & oHeader.Item(CStr(j)) & """, " & sFilePrev & "="""" & _
                      wsTP.Cells(1, j).Text & """"
      End If
      j = j + 1
    Loop
    If bFatal Then
      wsW.Select
      Exit Sub
    End If

    i = 2
```

```
            Do While Not (IsEmpty(wsT.Cells(i, 1)) And IsEmpty(wsT.Cells(i, 1).End(xlToRight)))
              If Not bFileSpecs Then
                GLogger.fatal "Please provide FileSpecs.csv file and specify main sort column " & _
                              "for file " & vFile
                wsW.Select
                Exit Sub
              End If
              If Not oFileSpecs.exists(sCoreFile & ":2") Then
                GLogger.fatal "Please specify main sort column for file " & vFile & _
                              " in Config file " & sFileSpecs
                wsW.Select
                End
              End If
              Sid = wsT.Cells(i, CLng(oFileSpecs.Item(sCoreFile & ":2")))
              sID2 = ""
              If CLng(oFileSpecs.Item(sCoreFile & ":3")) > 0 Then
                sID2 = wsT.Cells(i, CLng(oFileSpecs.Item(sCoreFile & ":3")))
              End If
              sID3 = ""
              If CLng(oFileSpecs.Item(sCoreFile & ":4")) > 0 Then
                sID3 = wsT.Cells(i, CLng(oFileSpecs.Item(sCoreFile & ":4")))
              End If
              wsT.Cells(i, lMainSortCol) = Sid & "|" & sID2 & "|" & sID3
              i = i + 1
            Loop

            wsT.Sort.SortFields.Clear
            wsT.Sort.SortFields.Add Key:=wsT.Range(wsT.Cells(2, lMainSortCol), _
                                                   wsT.Cells(i - 1, lMainSortCol)), _
                                    SortOn:=xlSortOnValues, Order:=xlAscending, DataOption:=xlSortNormal
            With wsT.Sort
              .SetRange Range(wsT.Cells(2, 1), wsT.Cells(i - 1, lMainSortCol))
              .Header = xlNo
              .Apply
            End With

            i = 2
            Do While Not (IsEmpty(wsTP.Cells(i, 1)) And IsEmpty(wsTP.Cells(i, 1).End(xlToRight)))
              Sid = wsTP.Cells(i, CLng(oFileSpecs.Item(sCoreFile & ":2")))
              sID2 = ""
              If CLng(oFileSpecs.Item(sCoreFile & ":3")) > 0 Then
                sID2 = wsTP.Cells(i, CLng(oFileSpecs.Item(sCoreFile & ":3")))
              End If
              sID3 = ""
              If CLng(oFileSpecs.Item(sCoreFile & ":4")) > 0 Then
                sID3 = wsTP.Cells(i, CLng(oFileSpecs.Item(sCoreFile & ":4")))
              End If
              wsTP.Cells(i, lMainSortCol) = Sid & "|" & sID2 & "|" & sID3
              i = i + 1
            Loop
            wsTP.Sort.SortFields.Clear
            wsTP.Sort.SortFields.Add Key:=Range(wsTP.Cells(2, lMainSortCol), _
                                                wsTP.Cells(i - 1, lMainSortCol)), _
                                     SortOn:=xlSortOnValues, Order:=xlAscending, DataOption:=xlSortNormal
            With wsTP.Sort
              .SetRange Range(wsTP.Cells(2, 1), wsTP.Cells(i - 1, lMainSortCol))
              .Header = xlNo
              .Apply
            End With
            wsTM.Range(wsTM.Cells(1, 1), _
                       wsTM.Cells(1, lColumns)).FormulaArray = _
                                                   Array(wsT.Range(wsT.Cells(1, 1), _
                                                                   wsT.Cells(1, lColumns)))
            lTM = 2
          End If

          'Now walk through sorted input files

          If bHeaderFile Then
            i = 1
            j = 1
          Else
            i = 2
            j = 2
          End If
Label_Start:
          If bHeaderFile Then
            'See that both Temp and TempPrev are on BAS

            'First Temp
            k = i
            Do While Not (IsEmpty(wsT.Cells(i, 1)) And _
                          IsEmpty(wsT.Cells(i, 1).End(xlToRight))) And wsT.Cells(i, 1) <> "BAS"
              i = i + 1
            Loop

            'Now TempPrev
            k = j
            Do While Not (IsEmpty(wsTP.Cells(j, 1)) And _
                          IsEmpty(wsTP.Cells(j, 1).End(xlToRight))) And wsTP.Cells(j, 1) <> "BAS"
```

```
    j = j + 1
  Loop
End If

Do While Not (IsEmpty(wsT.Cells(i, 1)) And IsEmpty(wsT.Cells(i, 1).End(xlToRight)) And _
              IsEmpty(wsTP.Cells(j, 1)) And IsEmpty(wsTP.Cells(j, 1).End(xlToRight)))
  k = lMainSortCol
  Debug.Assert k > 0
  If Not IsEmpty(wsTP.Cells(j, 1)) And _
     (IsEmpty(wsT.Cells(i, 1)) Or _
      StrComp(wsT.Cells(i, k).Text, wsTP.Cells(j, k).Text, vbTextCompare) = 1) Then
    'Copy old record from previous file into Deleted tab
    If bHeaderFile Then
      s = wsTP.Cells(j, 1)
      Do
        If wsTP.Cells(j, 1) = "BAS" Then
          lColumns = oHeader.Item(wsTP.Cells(j, 1).Text & ":" & _
                                  wsTP.Cells(j, 2).Text & ":COLUMNS")
          If Not bDeletedRecords Then
            wsDel.Cells(lDel, 1) = "FileName"
            For m = 1 To lColumns
              wsDel.Cells(lDel, m + 1) = oHeader.Item(wsTP.Cells(j, _
                                         1).Text & ":" & wsTP.Cells(j, 2).Text & ":" & m)
            Next m
            bDeletedRecords = True
            lDel = lDel + 1
          End If
          'Copy only BAS row
          wsDel.Cells(lDel, 1) = vFile
          On Error Resume Next
          wsDel.Range(wsDel.Cells(lDel, 2), _
                      wsDel.Cells(lDel, lColumns + 1)).FormulaArray = _
                                  Array(wsTP.Range(wsTP.Cells(j, 1), wsTP.Cells(j, lColumns)))
          On Error GoTo ErrHdl
          lDel = lDel + 1
        End If
        j = j + 1
      Loop Until wsTP.Cells(j, 1) = "" Or s = wsTP.Cells(j, 1)
    Else
      If Not bDeletedRecords Then
        wsDel.Cells(lDel, 1) = "FileName"
        wsDel.Range(wsDel.Cells(lDel, 2), _
                    wsDel.Cells(lDel, lColumns + 1)).FormulaArray = _
                                  Array(wsTP.Range(wsTP.Cells(1, 1), wsTP.Cells(1, lColumns)))
        bDeletedRecords = True
        lDel = lDel + 1
      End If
      wsDel.Cells(lDel, 1) = vFile
      wsDel.Range(wsDel.Cells(lDel, 2), _
                  wsDel.Cells(lDel, lColumns + 1)).FormulaArray = _
                                  Array(wsTP.Range(wsTP.Cells(j, 1), wsTP.Cells(j, lColumns)))
      lDel = lDel + 1
      j = j + 1
    End If
  ElseIf Not IsEmpty(wsT.Cells(i, 1)) And _
         (IsEmpty(wsTP.Cells(j, 1)) Or _
          StrComp(wsT.Cells(i, k).Text, wsTP.Cells(j, k).Text, vbTextCompare) = -1) Then
    'Copy new record from current file into New tab
    If bHeaderFile Then
      s = wsT.Cells(i, 1)
      Do
        If wsT.Cells(i, 1) = "BAS" Then
          lColumns = oHeader.Item(wsT.Cells(i, 1).Text & ":" & _
                                  wsT.Cells(i, 2).Text & ":COLUMNS")
          If Not bNewRecords Then
            wsN.Cells(lNew, 1) = "FileName"
            For m = 1 To lColumns
              wsN.Cells(lNew, m + 1) = oHeader.Item(wsT.Cells(i, 1).Text & _
                                                    ":" & wsT.Cells(i, 2).Text & ":" & m)
            Next m
            bNewRecords = True
            lNew = lNew + 1
          End If
          'Copy only BAS row
          wsN.Cells(lNew, 1) = vFile
          On Error Resume Next
          wsN.Range(wsN.Cells(lNew, 2), _
                    wsN.Cells(lNew, _
                              lColumns + 1)).FormulaArray = _
                              Array(wsT.Range(wsT.Cells(i, 1), wsT.Cells(i, lColumns)))
          On Error GoTo ErrHdl
          lNew = lNew + 1
        End If
        i = i + 1
      Loop Until wsT.Cells(i, 1) = "" Or s = wsT.Cells(i, 1)
    Else
      If Not bNewRecords Then
        wsN.Cells(lNew, 1) = "FileName"
        wsN.Range(wsN.Cells(lNew, 2), _
                  wsN.Cells(lNew, _
```

```
                                    lColumns + 1)).FormulaArray = _
                                    Array(wsT.Range(wsT.Cells(1, 1), wsT.Cells(1, lColumns)))
                  bNewRecords = True
                  lNew = lNew + 1
                End If
                wsN.Cells(lNew, 1) = vFile
                On Error Resume Next
                wsN.Range(wsN.Cells(lNew, 2), _
                          wsN.Cells(lNew, _
                                    lColumns + 1)).FormulaArray = _
                                    Array(wsT.Range(wsT.Cells(i, 1), wsT.Cells(i, lColumns)))
                On Error GoTo ErrHdl
                i = i + 1
                lNew = lNew + 1
              End If
            Else
              'Copy same or changed record into TempMove tab
              If bHeaderFile Then
                If wsT.Cells(i, 1) <> "BAS" Or wsTP.Cells(j, 1) <> "BAS" Then
                  GLogger.fatal "Internal error. BAS rows expected for both " & vFile & _
                                ", row " & i & " and " & sFilePrev & ", row " & j
                  Stop
                  Exit Sub
                End If
                'We implement a finite automaton = state engine, _
                 starting with two BAS rows with same Id
                Do
Label_State_BAS_BAS:
                  lColumns = oHeader.Item(wsT.Cells(i, 1).Text & ":" & _
                                          wsT.Cells(i, 2).Text & ":COLUMNS")
                  For k = 1 To lColumns
                    sCell = wsT.Cells(i, k).Text
                    If k = 1 Or k = 2 Or _
                       k = CLng(oSortCol.Item(wsT.Cells(i, 2).Text)) Then
                      wsTM.Cells(lTM, k) = sCell
                    Else
                      sCellPrev = wsTP.Cells(j, k).Text
                      If IsNumeric(sCell) And IsNumeric(sCellPrev) Then
                        wsTM.Cells(lTM, k) = CDbl(sCell) - CDbl(sCellPrev)
                      ElseIf IsDate(sCell) And IsDate(sCellPrev) Then
                        wsTM.Cells(lTM, k) = CDate(sCell) - CDate(sCellPrev)
                      Else
                        If sCell = "" Then sCell = "''"
                        If sCellPrev = "" Then sCellPrev = "''"
                        If sCell = " " Then sCell = "' '"
                        If sCellPrev = " " Then sCellPrev = "' '"
                        If sCell = sCellPrev Then
                          wsTM.Cells(lTM, k) = "="
                        Else
                          wsTM.Cells(lTM, k) = sCellPrev & "->" & sCell
                        End If
                      End If
                    End If
                  Next k
                  lTM = lTM + 1
                  i = i + 1
                  j = j + 1

                  'Now state transition: decide to which state to switch to
                  If wsT.Cells(i, 1) = "BAS" And wsTP.Cells(j, 1) = "BAS" Then
                    GoTo Label_Start
                  ElseIf wsT.Cells(i, 1) = "BAS" Then
                    k = j
                    Do While Not (IsEmpty(wsTP.Cells(j, 1)) And _
                                  IsEmpty(wsTP.Cells(j, 1).End(xlToRight))) And wsTP.Cells(j, 1) <> "BAS"
                      j = j + 1
                    Loop
                    GoTo Label_Start
                  ElseIf wsTP.Cells(j, 1) = "BAS" Then
                    k = i
                    Do While Not (IsEmpty(wsT.Cells(i, 1)) And _
                                  IsEmpty(wsT.Cells(i, 1).End(xlToRight))) And wsT.Cells(i, 1) <> "BAS"
                      i = i + 1
                    Loop
                    GoTo Label_Start
                  Else
                    sID2 = wsT.Cells(i, 3)
                    sID3 = wsTP.Cells(j, 3)
Label_Sublevel_Same_BAS:
                    If sID2 = sID3 Then
                      If sID2 = "" Then GoTo Label_Start
                      If wsT.Cells(i, _
                                   CLng(oSortCol.Item(wsT.Cells(i, _
                                   2).Text))).Text = wsTP.Cells(j, _
                                   CLng(oSortCol.Item(wsT.Cells(i, 2).Text))).Text Then
                        For k = 1 To lColumns
                          sCell = wsT.Cells(i, k).Text
                          If k = 1 Or k = 2 Or k = _
                             CLng(oSortCol.Item(wsT.Cells(i, 2).Text)) Then
                            wsTM.Cells(lTM, k) = sCell
```

```
      Else
        sCellPrev = wsTP.Cells(j, k).Text
        If IsNumeric(sCell) And IsNumeric(sCellPrev) Then
          wsTM.Cells(lTM, k) = CDbl(sCell) - CDbl(sCellPrev)
        ElseIf IsDate(sCell) And IsDate(sCellPrev) Then
          wsTM.Cells(lTM, k) = CDate(sCell) - CDate(sCellPrev)
        Else
          If sCell = "" Then sCell = "''"
          If sCellPrev = "" Then sCellPrev = "''"
          If sCell = " " Then sCell = "' '"
          If sCellPrev = " " Then sCellPrev = "' '"
          If sCell = sCellPrev Then
            wsTM.Cells(lTM, k) = "="
          Else
            wsTM.Cells(lTM, k) = sCellPrev & "->" & sCell
          End If
        End If
      End If
    Next k
  ElseIf wsT.Cells(i, _
                  CLng(oSortCol.Item(wsT.Cells(i, _
                  2).Text))).Text > wsTP.Cells(j, _
                  CLng(oSortCol.Item(wsT.Cells(i, 2).Text))).Text Then
    k = j
    'Only evaluate conditions until we face FALSE
    b = Not IsEmpty(wsTP.Cells(j, 1))
    If b Then
      b = b And (wsTP.Cells(j, 1) <> "BAS")
      If b Then
        b = b And (sID2 = sID3)
        If b Then
          b = b And wsT.Cells(i, _
                  CLng(oSortCol.Item(wsT.Cells(i, _
                  2).Text))).Text > wsTP.Cells(j, _
                  CLng(oSortCol.Item(wsT.Cells(i, _
                  2).Text))).Text
        End If
      End If
    End If
    Do While b
      j = j + 1
      sID3 = wsTP.Cells(j, 3)
      'Only evaluate conditions until we face FALSE
      b = Not IsEmpty(wsTP.Cells(j, 1))
      If b Then
        b = b And (wsTP.Cells(j, 1) <> "BAS")
        If b Then
          b = b And (sID2 = sID3)
          If b Then
            b = b And wsT.Cells(i, _
                              CLng(oSortCol.Item(wsT.Cells(i, _
                              2).Text))).Text > wsTP.Cells(j, _
                              CLng(oSortCol.Item(wsT.Cells(i, _
                              2).Text))).Text
          End If
        End If
      End If
    Loop
    If IsEmpty(wsTP.Cells(j, 1)) Or _
       wsTP.Cells(j, 1) = "BAS" Then
      GoTo Label_Start
    Else
      GoTo Label_Sublevel_Same_BAS
    End If
  Else
    k = i
    'Only evaluate conditions until we face FALSE
    b = Not IsEmpty(wsT.Cells(i, 1))
    If b Then
      b = b And (wsT.Cells(i, 1) <> "BAS")
      If b Then
        b = b And (sID2 = sID3)
        If b Then
          b = b And wsT.Cells(i, CLng(oSortCol.Item(wsT.Cells(i, _
                  2).Text))).Text < wsTP.Cells(j, CLng(oSortCol.Item(wsT.Cells(i, _
                  2).Text))).Text
        End If
      End If
    End If
    Do While b
      i = i + 1
      sID2 = wsT.Cells(i, 3)
      'Only evaluate conditions until we face FALSE
      b = Not IsEmpty(wsT.Cells(i, 1))
      If b Then
        b = b And (wsT.Cells(i, 1) <> "BAS")
        If b Then
          b = b And (sID2 = sID3)
          If b Then
            b = b And wsT.Cells(i, _
```

```
                                        CLng(oSortCol.Item(wsT.Cells(i, _
                                        2).Text))).Text < wsTP.Cells(j, _
                                        CLng(oSortCol.Item(wsT.Cells(i, _
                                        2).Text))).Text
                    End If
                  End If
                End If
              Loop
              If IsEmpty(wsT.Cells(i, 1)) Or wsT.Cells(i, 1) = "BAS" Then
                GoTo Label_Start
              Else
                GoTo Label_Sublevel_Same_BAS
              End If
            End If
          ElseIf sID2 > sID3 Then
            k = j
            Do While Not (IsEmpty(wsTP.Cells(j, 1)) And _
                          IsEmpty(wsTP.Cells(j, 1).End(xlToRight))) And _
             wsTP.Cells(j, 1) <> "BAS" And sID2 > sID3
              j = j + 1
            Loop
            If IsEmpty(wsTP.Cells(j, 1)) Or wsTP.Cells(j, 1) = "BAS" Then
              GoTo Label_Start
            Else
              GoTo Label_Sublevel_Same_BAS
            End If
          Else
            k = i
            Do While Not (IsEmpty(wsT.Cells(i, 1)) And _
                          IsEmpty(wsT.Cells(i, 1).End(xlToRight))) And _
             wsT.Cells(i, 1) <> "BAS" And sID2 < sID3
              i = i + 1
            Loop
            If IsEmpty(wsT.Cells(i, 1)) Or wsT.Cells(i, 1) = "BAS" Then
              GoTo Label_Start
            Else
              GoTo Label_Sublevel_Same_BAS
            End If
          End If
        End If
        lTM = lTM + 1
        i = i + 1
        j = j + 1

        'Now state transition: decide to which state to switch to
        If wsT.Cells(i, 1) = "BAS" And wsTP.Cells(j, 1) = "BAS" Then
          GoTo Label_Start
        ElseIf wsT.Cells(i, 1) = "BAS" Then
          k = j
          Do While Not (IsEmpty(wsTP.Cells(j, 1)) And IsEmpty(wsTP.Cells(j, 1).End(xlToRight))) _
             And wsTP.Cells(j, 1) <> "BAS"
            j = j + 1
          Loop
          GoTo Label_Start
        ElseIf wsTP.Cells(j, 1) = "BAS" Then
          k = i
          Do While Not (IsEmpty(wsT.Cells(i, 1)) And IsEmpty(wsT.Cells(i, 1).End(xlToRight))) _
             And wsT.Cells(i, 1) <> "BAS"
            i = i + 1
          Loop
          GoTo Label_Start
        End If
      Loop Until IsEmpty(wsT.Cells(i, 1)) Or IsEmpty(wsTP.Cells(j, 1)) Or _
           (wsT.Cells(i, 1) = "BAS" And wsTP.Cells(j, 1) = "BAS")
    Else
      For k = 1 To lColumns
        sCell = wsT.Cells(i, k).Text
        If k = CLng(oFileSpecs(sCoreFile & ":2")) Or _
           k = CLng(oFileSpecs(sCoreFile & ":3")) Or _
           k = CLng(oFileSpecs(sCoreFile & ":4")) Then
          wsTM.Cells(lTM, k) = sCell
        Else
          sCellPrev = wsTP.Cells(j, k).Text
          If IsNumeric(sCell) And IsNumeric(sCellPrev) Then
            wsTM.Cells(lTM, k) = CDbl(sCell) - CDbl(sCellPrev)
          ElseIf IsDate(sCell) And IsDate(sCellPrev) Then
            wsTM.Cells(lTM, k) = CDate(sCell) - CDate(sCellPrev)
          Else
            If sCell = "" Then sCell = "''"
            If sCellPrev = "" Then sCellPrev = "''"
            If sCell = " " Then sCell = "' '"
            If sCellPrev = " " Then sCellPrev = "' '"
            If sCell = sCellPrev Then
              wsTM.Cells(lTM, k) = "="
            Else
              wsTM.Cells(lTM, k) = sCellPrev & "->" & sCell
            End If
          End If
        End If
      Next k
```

```
        i = i + 1
        j = j + 1
        lTM = lTM + 1

        'Now state transition: decide to which state to switch to
        If wsT.Cells(i, 1) = "BAS" And wsTP.Cells(j, 1) = "BAS" Then
          GoTo Label_Start
        ElseIf wsT.Cells(i, 1) = "BAS" Then
          k = j
          Do While Not (IsEmpty(wsTP.Cells(j, 1)) And IsEmpty(wsTP.Cells(j, 1).End(xlToRight))) _
             And wsTP.Cells(j, 1) <> "BAS"
            j = j + 1
          Loop
          GoTo Label_Start
        ElseIf wsTP.Cells(j, 1) = "BAS" Then
          k = i
          Do While Not (IsEmpty(wsT.Cells(i, 1)) And IsEmpty(wsT.Cells(i, 1).End(xlToRight))) _
             And wsT.Cells(i, 1) <> "BAS"
            i = i + 1
          Loop
          GoTo Label_Start
        End If
      End If
    End If

    If IsEmpty(wsT.Cells(i, 1)) Or IsEmpty(wsT.Cells(j, 1)) Then GoTo Label_Start

  Loop  'While Not (IsEmpty(wsT.Cells(i, 1)) And IsEmpty(wsT.Cells(j, 1)))

  '############################## Now process TempMove tab ##############################
  wsTM.Cells.EntireColumn.AutoFit

  'Now iterate over a) row types, b) columns, and c) rows
  For k = 0 To lRowTypes
    'GLogger.info oRowTypes.Keys()(k) & ":COLUMNS = " & _
     oHeader.Item(oRowTypes.Keys()(k) & ":COLUMNS")
    If bHeaderFile Then
      lColumns = oHeader.Item(oRowTypes.keys()(k) & ":COLUMNS")
      If lColumns = 0 Then
        GLogger.warn "File " & vFile & ", row type '" & oRowTypes.keys()(k) & _
                     "' not found in header"
      End If
    End If
    For j = lColumnStart To lColumns
      Set oData = CreateObject("Scripting.Dictionary")
      Set oStringLen = CreateObject("Scripting.Dictionary")
      dSum = 0#
      sMsg = ""
      lCount = 0
      lUBounddStdDevInput = CInitialUBounddStdDevInput
      ReDim dStdDevInput(1 To lUBounddStdDevInput) As Double
      lCountAll = 0
      bIsDate = False
      For i = 1 To 3
        GdMax(i) = -1E+308
        GdMin(i) = 1E+308
      Next i
      GlMaxIdx = 0
      GlMinIdx = 0
      If bHeaderFile Then
        i = lRowTypes + 2
      Else
        i = 2
      End If

      dMin = CMaxDouble
      dMax = CMinDouble
      dMinDate = CMaxDate
      dMaxDate = CMinDate
      lLenMin = CMaxLength
      lLenMax = CMinLength
      bHasEmpty = False

      Do While Not (IsEmpty(wsTM.Cells(i, 1)) And IsEmpty(wsTM.Cells(i, 1).End(xlToRight)))
        If bHeaderFile Then
          sHeader = wsTM.Cells(i, 1).Text & ":" & _
                    wsTM.Cells(i, 2).Text
          bProcessThisRow = (sHeader = oRowTypes.keys()(k))
          sField = oHeader.Item(oRowTypes.keys()(k) & ":" & j)
        Else
          sField = oHeader.Item(CStr(j))
        End If
        If bProcessThisRow Then
          If bHeaderFile Then
            bIsIDField = (oSortCol.Item(wsTM.Cells(i, 2).Text) = j)
          Else
            bIsIDField = (oFileSpecs.Item(sCoreFile & ":2") = j Or _
                          oFileSpecs.Item(sCoreFile & ":3") = j Or _
                          oFileSpecs.Item(sCoreFile & ":4") = j)
          End If
```

```
lCountAll = lCountAll + 1
sCell = wsTM.Cells(i, j).Text
If sCell = "" Then
  bHasEmpty = True
  If bLimitsMoveIn Then
    lLimRow = oLimitMoveRow.Item(sCoreFile & ":" & sField)
    If lLimRow > 0 Then
      If Not CBool(wsLMI.Cells(lLimRow, limAllowEmpty)) Then
        GLogger.warn "Change on File " & vFile & _
                     ", Field " & sField & ", Row " & i & _
                     ", should not be empty"
      End If
    End If
  End If
End If
If IsNumeric(sCell) Then
  dVal = CDbl(sCell)
  If dVal > dMax Then dMax = dVal
  If dVal < dMin Then dMin = dVal
  dSum = dSum + dVal
  lCount = lCount + 1
  dStdDevInput(lCount) = dVal
  Call Extreme(dVal)
  If bLimitsMoveIn Then
    lLimRow = oLimitMoveRow.Item(sCoreFile & ":" & sField)
    If lLimRow > 0 Then
      If Not IsEmpty(wsLMI.Cells(lLimRow, limNumMin)) Then
        If dVal < wsLMI.Cells(lLimRow, limNumMin) Then
          GLogger.warn "Change on File " & vFile & _
                       ", Field " & sField & ", Row " & i & _
                       ", Value " & dVal & " < " & _
                       wsLMI.Cells(lLimRow, limNumMin) & _
                       " [Min Limit]"
        End If
      End If
      If Not IsEmpty(wsLMI.Cells(lLimRow, limNumMax)) Then
        If dVal > wsLMI.Cells(lLimRow, limNumMax) Then
          GLogger.warn "Change on File " & vFile & _
                       ", Field " & sField & ", Row " & i & _
                       ", Value " & dVal & " > " & _
                       wsLMI.Cells(lLimRow, limNumMax) & _
                       " [Max Limit]"
        End If
      End If
    End If
  End If
ElseIf IsDate(sCell) Then
  bIsDate = True
  dtDate = CDate(sCell)
  dVal = CDbl(dtDate)
  If dVal > dMaxDate Then dMaxDate = dVal
  If dVal < dMinDate Then dMinDate = dVal
  dSum = dSum + dVal
  lCount = lCount + 1
  dStdDevInput(lCount) = dVal
  Call Extreme(dVal)
  If bLimitsMoveIn Then
    lLimRow = oLimitMoveRow.Item(sCoreFile & ":" & sField)
    If lLimRow > 0 Then
      If Not IsEmpty(wsLMI.Cells(lLimRow, limDateMin)) Then
        If dVal < CDbl(CDate(wsLMI.Cells(lLimRow, _
                                         limDateMin).Text)) Then
          GLogger.warn "Change on File " & vFile & _
                       ", Field " & sField & ", Row " & i & _
                       ", Date " & Format(dVal, "dd-mmm-yyyy") & _
                       " < " & Format(wsLMI.Cells(lLimRow, _
                                                  limDateMin), "dd-mmm-yyyy") & _
                       " [Min Limit]"
        End If
      End If
      If Not IsEmpty(wsLMI.Cells(lLimRow, limDateMax)) Then
        If dVal > CDbl(CDate(wsLMI.Cells(lLimRow, _
                                         limDateMax).Text)) Then
          GLogger.warn "Change on File " & vFile & _
                       ", Field " & sField & ", Row " & i & _
                       ", Date " & Format(dVal, "dd-mmm-yyyy") & _
                       " > " & Format(wsLMI.Cells(lLimRow, _
                                                  limDateMax), "dd-mmm-yyyy") & _
                       " [Max Limit]"
        End If
      End If
    End If
  End If
Else
  'String
  oData.Item("'" & sCell & "") = _
                                 oData.Item("'" & sCell & "") + 1
  lLen = Len(sCell)
  oStringLen.Item(CStr(lLen)) = oStringLen.Item(CStr(lLen)) + 1
  If lLen > 0 Then
```

```
        If lLen > lLenMax Then lLenMax = lLen
        If lLen < lLenMin Then lLenMin = lLen
      End If
      If bLimitsMoveIn Then
        lLimRow = oLimitMoveRow.Item(sCoreFile & ":" & sField)
        If lLimRow > 0 Then
          If Not CBool(wsLI.Cells(lLimRow, limAllowEmpty)) And _
             Not IsEmpty(wsLMI.Cells(lLimRow, limLenMin)) Then
            If lLen < wsLMI.Cells(lLimRow, limLenMin).Value Then
              GLogger.warn "Change on File " & vFile & _
                           ", Field " & sField & ", Row " & i & _
                           ", Length('" & sCell & "') = " & lLen & " < " & _
                           wsLMI.Cells(lLimRow, limLenMin) & _
                           " [Min Length]"
            End If
          End If
          If Not IsEmpty(wsLMI.Cells(lLimRow, limLenMax)) Then
            If lLen > wsLMI.Cells(lLimRow, limLenMax).Value Then
              GLogger.warn "Change on File " & vFile & _
                           ", Field " & sField & ", Row " & i & _
                           ", Length('" & sCell & "') = " & lLen & " > " & _
                           wsLMI.Cells(lLimRow, limLenMax) & _
                           " [Max Length]"
            End If
          End If
        End If
      End If
    End If
  End If  'If bProcessThisRow
  i = i + 1
Loop

If bIsIDField Then
  wsTSM.Cells(lTSM, out2File) = vFile
  If bHeaderFile Then
    wsTSM.Cells(lTSM, out2Header) = oRowTypes.keys()(k)
    wsTSM.Cells(lTSM, out2Field) = oHeader.Item(oRowTypes.keys()(k) & ":" & j)
  Else
    wsTSM.Cells(lTSM, out2Field) = oHeader.Item(CStr(j))
  End If
  wsTSM.Cells(lTSM, out2Warn) = "No stats - this field is an identifier field"
  lTSM = lTSM + 1
Else
  'Deal with numeric values
  If lCount > 0 Then
    wsNSM.Cells(lNSM, outFile) = vFile
    If bHeaderFile Then
      wsNSM.Cells(lNSM, outHeader) = oRowTypes.keys()(k)
    End If
    wsNSM.Cells(lNSM, outField) = sField
    wsNSM.Cells(lNSM, outAverage) = dSum / lCount
    ReDim Preserve dStdDevInput(1 To lCount) As Double
    If lCountAll > lCount Then
      sMsg = sMsg & lCountAll - lCount & " values of " & lCountAll & _
             " are non-numeric. "
    End If
    d = 0#
    On Error Resume Next
    d = Application.WorksheetFunction.StDevP(dStdDevInput)
    On Error GoTo ErrHdl
    If d > 0# Then
      If Abs(dSum / lCount - GdMin(1)) > dStdDevThreshold * d Then
        sMsg = sMsg & "Min is off by more than " & dStdDevThreshold & _
               " stdev. "
      End If
      If Abs(dSum / lCount - GdMax(1)) > dStdDevThreshold * d Then
        sMsg = sMsg & "Max is off by more than " & dStdDevThreshold & _
               " stdev. "
      End If
      '                                Else
      'GLogger.warn "No standard deviation possible for file " & _
       vFile & ", field " & wsNSM.Cells(lNSM, outField)
    End If
    wsNSM.Cells(lNSM, outWarn) = sMsg
    sMsg = ""
    If GlMinIdx > 2 Then wsNSM.Cells(lNSM, outMin1) = GdMin(1)
    If GlMinIdx > 1 Then wsNSM.Cells(lNSM, outMin2) = GdMin(2 + _
                                                        (GlMinIdx < 3))
    If GlMinIdx > 0 Then wsNSM.Cells(lNSM, outMin3) = GdMin(3 + _
                                                        (GlMinIdx < 3) + (GlMinIdx < 2))
    If GlMaxIdx > 2 Then wsNSM.Cells(lNSM, outMax1) = GdMax(1)
    If GlMaxIdx > 1 Then wsNSM.Cells(lNSM, outMax2) = GdMax(2 + _
                                                        (GlMaxIdx < 3))
    If GlMaxIdx > 0 Then wsNSM.Cells(lNSM, outMax3) = GdMax(3 + _
                                                        (GlMaxIdx < 3) + (GlMaxIdx < 2))
    If bIsDate Then
      Range(wsNSM.Cells(lNSM, outMin1), wsNSM.Cells(lNSM, _
                                                  outMax1)).NumberFormat = "dd-mmm-yyyy"
    End If
    lNSM = lNSM + 1
```

```
      End If

      'Deal with string values
      If lCount < lCountAll Then
        wsTSM.Cells(lTSM, out2File) = vFile
        If bHeaderFile Then
          wsTSM.Cells(lTSM, out2Header) = oRowTypes.keys()(k)
          wsTSM.Cells(lTSM, out2Field) = oHeader.Item(oRowTypes.keys()(k) & ":" & j)
        Else
          wsTSM.Cells(lTSM, out2Field) = oHeader.Item(CStr(j))
        End If
        If lCount > 0 Then
          sMsg = sMsg & lCount & " values of " & lCountAll & " are numeric. "
        End If
        If oData.Count > lCountAll - 5 Or (oData.Count / lCountAll > 0.5 And _
                                           lCountAll > 10) Then
          For i = oData.Count - 1 To 0 Step -1
            If oData.keys()(i) <> "'''" And oData.keys()(i) <> "'' '" _
               And oData.Item(oData.keys()(i)) = 1 _
               And InStr(oData.keys()(i), "->") = 0 Then
              oData.Remove oData.keys()(i)
            End If
          Next i
          If lCountAll - oData.Count > 0 Then
            sMsg = sMsg & lCountAll - oData.Count & _
                   " strings occurring once not shown. "
          End If
        End If
        wsTSM.Cells(lTSM, out2Warn) = sMsg
        sMsg = ""
        If oData.Count > 0 Then
          On Error Resume Next
          wsTSM.Cells(lTSM, out2Text).Resize(oData.Count, 2) = _
               Application.WorksheetFunction.Transpose(Array(oData.keys, oData.items))
          On Error GoTo ErrHdl
        End If
        If oStringLen.Count > 0 Then
          On Error Resume Next
          wsTSM.Cells(lTSM, out2Len).Resize(oStringLen.Count, 2) = _
               Application.WorksheetFunction.Transpose(Array(oStringLen.keys, oStringLen.items))
          On Error GoTo ErrHdl
        End If

        'Increase row counter
        lMax = 1
        If lMax < oData.Count Then lMax = oData.Count
        If lMax < oStringLen.Count Then lMax = oStringLen.Count
        lTSM = lTSM + lMax
      End If

      'Update Limits Move Out sheet
      wsLMO.Cells(lLMO, limFilePrefix) = sCoreFile
      If bHeaderFile Then
        wsLMO.Cells(lLMO, limHeader) = oRowTypes.keys()(k)
        wsLMO.Cells(lLMO, limField) = oHeader.Item(oRowTypes.keys()(k) & ":" & j)
      Else
        wsLMO.Cells(lLMO, limField) = oHeader.Item(CStr(j))
      End If
      If bHasEmpty Then wsLMO.Cells(lLMO, limAllowEmpty) = 1  'True
      If dMin < CMaxDouble Then wsLMO.Cells(lLMO, limNumMin) = dMin
      If dMax > CMinDouble Then wsLMO.Cells(lLMO, limNumMax) = dMax
      If dMinDate < CMaxDate Then
        wsLMO.Cells(lLMO, limDateMin) = dMinDate
        wsLMO.Cells(lLMO, limDateMin).NumberFormat = "dd-mmm-yyyy"
      End If
      If dMaxDate > CMinDate Then
        wsLMO.Cells(lLMO, limDateMax) = dMaxDate
        wsLMO.Cells(lLMO, limDateMax).NumberFormat = "dd-mmm-yyyy"
      End If
      If lLenMin < CMaxLength Then wsLMO.Cells(lLMO, limLenMin) = lLenMin
      If lLenMax > CMinLength Then wsLMO.Cells(lLMO, limLenMax) = lLenMax
      lLMO = lLMO + 1
    End If  'If bIsIDField

    Set oData = Nothing
    Set oStringLen = Nothing

  Next j
Next k

If lTM > 1 Then
  Call ExportRangeAsDelimitedText("TempMove", "A1:" & _
       wsTM.Cells(1, 1).SpecialCells(xlCellTypeLastCell).Address, _
       ThisWorkbook.Path & "\Datastats\" & "MOVE_" & sCoreFile & _
       "_" & Format(Now, "yyyymmdd") & ".csv", ",", True, _
       True, False)
End If

'######################## TempMove tab processing finished #########################
```

```
          wsTP.QueryTables.Item(1).Delete
        Else
          GLogger.warn "Previous period's file " & sFilePrev & " does not exist"
        End If
      End If  'bMoveCheck

      wsT.QueryTables.Item(1).Delete
      Set oRowTypes = Nothing
    Else
      GLogger.warn "Skipping zero file " & vFile
    End If
    If bClearDataAfterRun Then
      wsT.Range("1:1048576").Delete
      wsTP.Range("1:1048576").Delete
    End If
  Next vFile
  Call ExportRangeAsDelimitedText("NumStats", "A1:" & wsNS.Cells(lOut - 1, outUbound - 1).Address, _
                                  ThisWorkbook.Path & "\Datastats\" & "NumStats_" & _
                                  Format(Now, "yyyymmdd") & ".csv", ",", True, _
                                  True, False)
  Call ExportRangeAsDelimitedText("NumStatsMove", "A1:" & wsNSM.Cells(lNSM - 1, outUbound - 1).Address, _
                                  ThisWorkbook.Path & "\Datastats\" & "NumStatsMove_" & _
                                  Format(Now, "yyyymmdd") & ".csv", ",", True, _
                                  True, False)
  Call ExportRangeAsDelimitedText("TextStats", "A1:" & wsTS.Cells(lOut2 - 1, out2Ubound - 1).Address, _
                                  ThisWorkbook.Path & "\Datastats\" & "TextStats_" & _
                                  Format(Now, "yyyymmdd") & ".csv", ",", True, _
                                  True, False)
  Call ExportRangeAsDelimitedText("TextStatsMove", "A1:" & wsTSM.Cells(lTSM - 1, out2Ubound - 1).Address, _
                                  ThisWorkbook.Path & "\Datastats\" & "TextStatsMove_" & _
                                  Format(Now, "yyyymmdd") & ".csv", ",", True, _
                                  True, False)
  Call ExportRangeAsDelimitedText("LimitsOut", "A1:" & wsLO.Cells(lLim - 1, limCharSet + 255).Address, _
                                  ThisWorkbook.Path & "\Datastats\Limits_Output_" & _
                                  Format(Now, "yyyymmdd") & ".csv", ",", True, _
                                  True, False)
  Call ExportRangeAsDelimitedText("LimitsMoveOut", _
                                  "A1:" & wsLMO.Cells(lLMO - 1, limCharSet + 255).Address, _
                                  ThisWorkbook.Path & "\Datastats\Limits_Move_Output_" & _
                                  Format(Now, "yyyymmdd") & ".csv", ",", True, _
                                  True, False)
  Call ExportRangeAsDelimitedText("DeletedRecords", _
                                  "A1:" & wsDel.Range("A1").SpecialCells(xlCellTypeLastCell).Address, _
                                  ThisWorkbook.Path & "\Datastats\" & "DeletedRecords_" & _
                                  Format(Now, "yyyymmdd") & ".csv", ",", True, _
                                  True, False)
  Call ExportRangeAsDelimitedText("NewRecords", _
                                  "A1:" & wsN.Range("A1").SpecialCells(xlCellTypeLastCell).Address, _
                                  ThisWorkbook.Path & "\Datastats\" & "NewRecords_" & _
                                  Format(Now, "yyyymmdd") & ".csv", ",", True, _
                                  True, False)
  wsW.Select
  If bClearDataAfterRun Then
    wsNS.Range("1:1048576").Delete
    wsDel.Range("1:1048576").Delete
    wsFS.Range("1:1048576").Delete
    wsLI.Range("1:1048576").Delete
    wsLO.Range("1:1048576").Delete
    wsLMI.Range("1:1048576").Delete
    wsLMO.Range("1:1048576").Delete
    wsN.Range("1:1048576").Delete
    wsNSM.Range("1:1048576").Delete
    wsTS.Range("1:1048576").Delete
    wsTM.Range("1:1048576").Delete
    wsTSM.Range("1:1048576").Delete
  End If
  ThisWorkbook.SaveAs sFullName, lFileFormat
  GLogger.info "Reading Input finished"
  GLogger.info String(160, "<")
  Call Delete_QueryTables

  'Function doFileQuery provided unwanted names which now show #REF! errors. Delete those.
  For Each v In ActiveWorkbook.Names
    If InStr(v.Value, "#REF!") Then v.Delete
  Next

  'At the end of this sub the variable state gets destroyed, i.e. previous
  'system state variable values will be written back

  Exit Sub

ErrHdl:
  If Err.Number = 9 Then
    If lCount > lUBounddStdDevInput Then
      'Here we normally get if we breach dStdDevInput(lCount)
      'So we need to increase array size of dStdDevInput
      lUBounddStdDevInput = 2 * lUBounddStdDevInput
      ReDim Preserve dStdDevInput(1 To lUBounddStdDevInput) As Double
      Err.Number = 0
      Resume  'Back to statement which caused error
```

```
    End If
  End If
  'Other error - terminate
  On Error GoTo 0
  Resume

End Sub

Sub Extreme(d)
'Keep track of extreme values
'
'Change History:
'Version Date       Programmer Change
'1.00    21/12/2015 Bernd      Create

'Initialise with, for example:
'Dim GdMax(1 To 3) As Double
'Dim GlMaxIdx As Long
'Dim GdMin(1 To 3) As Double
'Dim GlMinIdx As Long
'For i = 1 To 3
'    GdMax(i) = -1E+308
'    GdMin(i) = 1E+308
'Next i
'GlMaxIdx = 0
'GlMinIdx = 0

  Dim i As Long, j As Long
  i = 1
  Do While i <= GlMaxIdx
    If d > GdMax(i) Then Exit Do
    i = i + 1
  Loop
  If i <= UBound(GdMax) Then
    If UBound(GdMax) > GlMaxIdx Then
      GlMaxIdx = GlMaxIdx + 1
    End If
    For j = GlMaxIdx To i + 1 Step -1
      GdMax(j) = GdMax(j - 1)
    Next j
    GdMax(i) = d
  End If
  i = 1
  Do While i <= GlMinIdx
    If d < GdMin(i) Then Exit Do
    i = i + 1
  Loop
  If i <= UBound(GdMin) Then
    If UBound(GdMin) > GlMinIdx Then
      GlMinIdx = GlMinIdx + 1
    End If
    For j = GlMinIdx To i + 1 Step -1
      GdMin(j) = GdMin(j - 1)
    Next j
    GdMin(i) = d
  End If

End Sub

Sub Delete_QueryTables()
'Delete query tables - hard code ignoring errors
'
'Change History:
'Version Date       Programmer Change
'1.00    21/12/2015 Bernd      Create

  On Error Resume Next  'If the QueryTable does not exist the next row will encounter an error but not
terminate
  wsNS.QueryTables.Item(1).Delete
  wsFS.QueryTables.Item(1).Delete
  wsLI.QueryTables.Item(1).Delete
  wsLMI.QueryTables.Item(1).Delete
  wsT.QueryTables.Item(1).Delete
  wsTP.QueryTables.Item(1).Delete
  wsTS.QueryTables.Item(1).Delete
  On Error GoTo 0  'Any errors will let the program terminate

End Sub

Function Files_PrefixStart_SuffixEnd(ParamArray vPath() As Variant) As Variant
'Determines common prefix and suffix strings of all files in one or more folders
'and returns prefix end (positive number) and suffix start (negative number) as
'well as prefix string and suffix string in a variant array with four values.
'Example: Files abc_file1_20181231.csv and abc_file2_20181231.csv would
'         result in return variant array [8,-13,"abc_file","_20181231.csv"}
'
'Change History:
'Version Date       Programmer Change
'0.1     05/01/2019 Bernd      Create
Dim i As Long, k As Long, lPrefix As Long, lSuffix As Long
```

```
Dim sFile1 As String, sFile As String, sPrefix As String, sSuffix As String

If Right(vPath(0), 1) <> "\" And Right(vPath(0), 1) <> "/" Then
  Files_PrefixStart_SuffixEnd = CVErr(xlErrValue)
  Exit Function
End If
sFile = Dir(vPath(0)): sPrefix = sFile: sSuffix = sFile: sFile1 = sFile
lPrefix = 0: lSuffix = 0    'Just one file so far
For k = LBound(vPath) To UBound(vPath)
  sFile = Dir(vPath(k))
  Do While Len(sFile) > 0
    If Len(sPrefix) > 0 Then
      If Left(sFile, Len(sPrefix)) <> sPrefix Then
        i = 0
        Do While Mid(sFile, i + 1, 1) = Mid(sPrefix, i + 1, 1)
          If i + 1 > Len(sFile) Or i + 1 > Len(sPrefix) Then Exit Do
          i = i + 1
        Loop
        If i < Len(sPrefix) Then sPrefix = Left(sPrefix, i): lPrefix = i
      End If
    End If
    If Len(sSuffix) > 0 Then
      If Right(sFile, Len(sSuffix)) <> sSuffix Then
        i = 0
        Do While Left(Right(sFile, i + 1), 1) = Left(Right(sSuffix, i + 1), 1)
          If i + 1 > Len(sFile) Or i + 1 > Len(sSuffix) Then Exit Do
          i = i + 1
        Loop
        If i < Len(sSuffix) Then sSuffix = Right(sSuffix, i): lSuffix = -i
      End If
    End If
    sFile = Dir()
  Loop
Next k
If lPrefix = 0 And lSuffix = 0 And Left(Right(sFile1, 4), 1) = "." Then sPrefix = "": sSuffix =
Right(sFile1, 4)
lPrefix = Len(sPrefix): lSuffix = Len(sSuffix)
Files_PrefixStart_SuffixEnd = Array(lPrefix, lSuffix, sPrefix, sSuffix)
End Function

Function sbCat(vP As Variant, _
               Optional sDel As String = ",", _
               Optional bNonEmpty As Boolean = True) As String
'Concatenate all cells in a range or array, delimited
'by sDel. If bNonEmpty is TRUE then only non-empty
'cells will be concatenated.
Dim v, s                     As String

For Each v In vP
  If Not (bNonEmpty And v = "") Then
    sbCat = sbCat & s & v
    s = sDel
  End If
Next v

End Function
```

Mini Calculator

This is a small (mini) calculator with two registers: a program counter *pc* and an accumulator *acc*. Two VBA programs interprete different modi: a direct mode, also called command line mode, and a program mode:

	A	B
1	**Minirechner**	
2	==========	
3		
4	**Direkter Modus:**	
5	bgn <adr>	pc := <adr>
6	dbg on|off	Debug Modus ein oder aus
7	srt	Programmstart
8		
9	**Programmmodus:**	
10	add <adr>	acc := acc + <adr>
11	beq <adr>	Springe nach <adr> wenn acc = 0
12	bgr <adr>	Springe nach <adr> wenn acc > 0
13	ble <adr>	Springe nach <adr> wenn acc < 0
14	bsa <adr>	Sprung zur Subroutine ab <adr>
15	bun <adr>	Sprung zu <adr>
16	cla	acc := 0
17	dac	acc := acc - 1
18	div <adr>	acc := acc / <adr> (ganzzahlige Division!)
19	hlt	Programmende
20	iac	acc := acc + 1
21	lda <adr>	acc := <adr>
22	mul <adr>	acc := acc * <adr>
23	out <adr>	Print <adr>
24	ret	Return von Subroutine
25	sta <adr>	<adr> := (acc)
26	sub <adr>	acc := acc - <adr>

This example program calculates the greatest common divisor and the least common multiple for two integers:

	D	E	F	G	H	I	J
1	**Direkte Eingabe = Kommandozeile:**						**Meldung / Fehlermeldung:**
2	srt						Programm wird gestartet bei pc = start
3							
4	**Zeile #**	**Label**	**OpCode**	**Argument**	**Kommentar**		**Ausgabebereich:**
5	1	start	bsa	ggt			Der größte gemeinsame Teiler ist:
6	2		out	out_ggt			250
7	3		out	result_ggt			Das kleinste gemeinsame Vielfache ist:
8	4		bsa	kgv			3750
9	5		out	out_kgv			Programmende erreicht in Zeile 7.
10	6		out	result_kgv			
11	7		hlt				
12	8	ggt	lda	arg1			
13	9		sta	temp1			
14	10		lda	arg2			
15	11		sta	temp2			
16	12	ggt_intern	lda	temp1			
17	13		sub	temp2			
18	14		beq	end_ggt			
19	15		bgr	store			
20	16		lda	temp2			
21	17		sub	temp1			
22	18		sta	temp2			
23	19		bun	ggt_intern			
24	20	store	sta	temp1			
25	21		bun	ggt_intern			
26	22	end_ggt	lda	temp1			
27	23		sta	result_ggt			
28	24		ret				
29	25	kgv	lda	arg1			
30	26		div	result_ggt			
31	27		mul	arg2			
32	28		sta	result_kgv			
33	29		ret				
34	30	temp1		250			
35	31	temp2		250			
36	32	out_ggt		Der größte gemeinsame Teiler ist:			
37	33	out_kgv		Das kleinste gemeinsame Vielfache ist:			
38	34	arg1		750			
39	35	arg2		1250			
40	36	result_ggt		250			
41	37	result_kgv		3750			

To run the example program, start the mini-calculator as follows:

- Set the starting point of your program in the command line with *bgn 1* or *bgn start* (if you have set the label *start*).
- Optionally, you can turn the debug mode on or off in the command line with *dbg on* or *dbg off*.
- Start your program with *srt* in the command line.

If you enter *dbg* on before starting the example program, you will get this output.:

```
Ausgabebereich:
Label 'start' := 1
Label 'ggt' := 8
Label 'ggt_intern' := 12
Label 'store' := 20
Label 'end_ggt' := 22
Label 'kgv' := 25
Label 'temp1' := 30
Label 'temp2' := 31
Label 'out_ggt' := 32
Label 'out_kgv' := 33
Label 'arg1' := 34
Label 'arg2' := 35
Label 'result_ggt' := 36
Label 'result_kgv' := 37
Unterprogrammaufruf 'ggt'. Rücksprung wurde auf Zeile 2 gesetzt. Stackindex 1.
Programmzähler wurde auf Zeile 8 gesetzt.
acc := 750
Argument in Zeile 30 wurde auf acc = 750 gesetzt.
acc := 1250
Argument in Zeile 31 wurde auf acc = 1250 gesetzt.
acc := 750
acc := acc - 1250
acc != 0 -> keine Verzweigung.
acc <= 0 -> keine Verzweigung.
acc := 1250
acc := acc - 750
Argument in Zeile 31 wurde auf acc = 500 gesetzt.
Verzweige nach ggt_intern.
Programmzähler wurde auf Zeile 12 gesetzt.
acc := 750
acc := acc - 500
acc != 0 -> keine Verzweigung.
acc > 0 -> verzweige nach store.
Programmzähler wurde auf Zeile 20 gesetzt.
Argument in Zeile 30 wurde auf acc = 250 gesetzt.
Verzweige nach ggt_intern.
Programmzähler wurde auf Zeile 12 gesetzt.
acc := 250
acc := acc - 500
acc != 0 -> keine Verzweigung.
acc <= 0 -> keine Verzweigung.
acc := 500
acc := acc - 250
Argument in Zeile 31 wurde auf acc = 250 gesetzt.
Verzweige nach ggt_intern.
Programmzähler wurde auf Zeile 12 gesetzt.
acc := 250
acc := acc - 250
acc = 0 -> verzweige nach end_ggt.
Programmzähler wurde auf Zeile 22 gesetzt.
acc := 250
Argument in Zeile 36 wurde auf acc = 250 gesetzt.
Unterprogrammrücksprung nach '2'. Stackindex 0.
Der größte gemeinsame Teiler ist:
250
Unterprogrammaufruf 'kgv'. Rücksprung wurde auf Zeile 5 gesetzt. Stackindex 1.
Programmzähler wurde auf Zeile 25 gesetzt.
acc := 750
acc := acc / 250
acc := acc * 1250
Argument in Zeile 37 wurde auf acc = 3750 gesetzt.
Unterprogrammrücksprung nach '5'. Stackindex 0.
Das kleinste gemeinsame Vielfache ist:
3750
Programmende erreicht in Zeile 7.
```

The Command Line Interpreter - Worksheet_Change Program Code

This code resides in sheet wsMain:

```
Private Sub Worksheet_Change(ByVal Target As Range)
'This implements the command line interpreter of the mini-calculator.
'(C) (P) by Bernd Plumhoff  26-Dec-2023 PB V0.1
Dim s           As String
'Application.EnableEvents = False
If Target.Address = Range("Kommandozeile").Address Then
  s = Range("Kommandozeile")
  Select Case Left(s, 3)
  Case "srt"
    If pc = 0 Or pc = "" Then pc = 1
    Range("Meldung") = "Programm wird gestartet bei pc = " & pc
    Range("Meldung").Font.ColorIndex = xlCIGreen
    Call interpreter
  Case "bgn"
    s = Right(s, Len(s) - 4)
    pc = s
    Range("Meldung") = "pc := " & s
    Range("Meldung").Font.ColorIndex = xlCIGreen
  Case "dbg"
    s = Right(s, Len(s) - 4)
    Select Case s
    Case "on"
      dbg = True
      Range("Meldung") = "dbg := on"
      Range("Meldung").Font.ColorIndex = xlCIGreen
    Case "off"
      dbg = False
      Range("Meldung") = "dbg := off"
      Range("Meldung").Font.ColorIndex = xlCIGreen
    Case Else
      Range("Meldung") = "Ungültiger Debug Modus '" & s & "'"
      Range("Meldung").Font.ColorIndex = xlCIRed
    End Select
  Case Else
    Range("Meldung") = "Unbekannter Befehl '" & s & "'"
    Range("Meldung").Font.ColorIndex = xlCIRed
  End Select
End If
'Application.EnableEvents = True
End Sub
```

The Program Interpreter - Interpreter Program Code

This code resides in module General:

```
'This implements the main program interpreter of the mini-calculator.
'(C) (P) by Bernd Plumhoff  26-Dec-2023 PB V0.1

Public Enum xlCI 'Excel Color Index
: xlCIBlack = 1: xlCIWhite: xlCIRed: xlCIBrightGreen: xlCIBlue '1 - 5
: xlCIYellow: xlCIPink: xlCITurquoise: xlCIDarkRed: xlCIGreen '6 - 10
: xlCIDarkBlue: xlCIDarkYellow: xlCIViolet: xlCITeal: xlCIGray25 '11 - 15
: xlCIGray50: xlCIPeriwinkle: xlCIPlum: xlCIIvory: xlCILightTurquoise '16 - 20
: xlCIDarkPurple: xlCICoral: xlCIOceanBlue: xlCIIceBlue: xlCILightBrown '21 - 25
: xlCIMagenta2: xlCIYellow2: xlCICyan2: xlCIDarkPink: xlCIDarkBrown '26 - 30
: xlCIDarkTurquoise: xlCISeaBlue: xlCISkyBlue: xlCILightTurquoise2: xlCILightGreen '31 - 35
: xlCILightYellow: xlCIPaleBlue: xlCIRose: xlCILavender: xlCITan '36 - 40
: xlCILightBlue: xlCIAqua: xlCILime: xlCIGold: xlCILightOrange '41 - 45
: xlCIOrange: xlCIBlueGray: xlCIGray40: xlCIDarkTeal: xlCISeaGreen '46 - 50
: xlCIDarkGreen: xlCIGreenBrown: xlCIBrown: xlCIDarkPink2: xlCIIndigo '51 - 55
: xlCIGray80 '56
End Enum

Enum pcol 'Spalten in jeder Programmzeile
  pcol_zeile = 0                        'Zeilennummer
  pcol_label
  pcol_opcode
  pcol_argument
  pcol_kommentar
End Enum

Public dbg                As Boolean      'Debug Modus
Public i                  As Integer      'Ausgabebereich Index
Public pc                 As Variant      'Programmzähler

Sub interpreter()
Dim b_end                 As Boolean      'Programmzeile leer?
Dim p                     As Integer      'Programm Index
Dim r                     As Integer      'Unterprogramm Stack Index
Dim ustack(1 To 100)      As Integer      'Unterprogramm Stack
Dim acc                   As Long         'Akkumulator
Dim st                    As Object       'Symboltabelle (Labels)
Dim op                    As String       'OpCode
Dim s                     As String
Dim v                     As Variant

'Initialisierungen

Range("Ausgabebereich").Resize(65536).ClearContents
i = 0

If pc = "" Then
  pc = 1
  debug_ausgabe ("Programmzähler wurde auf 1 initialisiert.")
End If

'Lade Symboltabelle
Set st = CreateObject("Scripting.Dictionary")
p = 1
b_end = (Range("Programmcode").Offset(p, pcol_label) = "" And _
         Range("Programmcode").Offset(p, pcol_opcode) = "" And _
         Range("Programmcode").Offset(p, pcol_argument) = "")
Do Until b_end
  s = Range("Programmcode").Offset(p, pcol_label)
  If s <> "" Then
    If st.exists(s) Then
      Call debug_ausgabe("Identische Label '" & s & "' in Zeilen " & st(s) & " und " & p & ". Abbruch!",
True)
      Exit Sub
    End If
    st(s) = p
    debug_ausgabe ("Label '" & s & "' := " & p)
  End If
  p = p + 1
  b_end = (Range("Programmcode").Offset(p, pcol_label) = "" And _
           Range("Programmcode").Offset(p, pcol_opcode) = "" And _
           Range("Programmcode").Offset(p, pcol_argument) = "")
Loop

'Interpretiere das Programm

Do

continue_do:

  If Not IsNumeric(pc) Then
    If st.exists(pc) Then
```

```
      pc = st(pc)
      debug_ausgabe ("Programmzähler wurde auf Zeile " & pc & " gesetzt.")

    Else
      Call debug_ausgabe("Programmzähler hat ungültiges Label '" & pc & "'. Abbruch!", True)
      Exit Sub
    End If
  End If

  op = Range("Programmcode").Offset(pc, pcol_opcode)
  Select Case op
  Case "add"
    v = Range("Programmcode").Offset(pc, pcol_argument)
    If Not IsNumeric(v) Then
      If st.exists(v) Then
        v = st(v)
      Else
        Call debug_ausgabe("Unbekanntes Argument '" & v & "' in Zeile " & pc & ". Abbruch!", True)
        Exit Sub
      End If
    End If
    acc = acc + Range("Programmcode").Offset(v, pcol_argument)
    debug_ausgabe ("acc := acc + " & Range("Programmcode").Offset(v, pcol_argument))
  Case "beq"
    If acc = 0 Then
      pc = Range("Programmcode").Offset(pc, pcol_argument)
      debug_ausgabe ("acc = 0 -> verzweige nach " & pc & ".")
      GoTo continue_do
    Else
      debug_ausgabe ("acc != 0 -> keine Verzweigung.")
    End If
  Case "bgr"
    If acc > 0 Then
      pc = Range("Programmcode").Offset(pc, pcol_argument)
      debug_ausgabe ("acc > 0 -> verzweige nach " & pc & ".")
      GoTo continue_do
    Else
      debug_ausgabe ("acc <= 0 -> keine Verzweigung.")
    End If
  Case "ble"
    If acc < 0 Then
      pc = Range("Programmcode").Offset(pc, pcol_argument)
      debug_ausgabe ("acc < 0 -> verzweige nach " & pc & ".")
      GoTo continue_do
    Else
      debug_ausgabe ("acc >= 0 -> keine Verzweigung.")
      End If
  Case "bsa"
    r = r + 1
    ustack(r) = pc + 1
    pc = Range("Programmcode").Offset(pc, pcol_argument)
    debug_ausgabe ("Unterprogrammaufruf '" & pc & _
        "'. Rücksprung wurde auf Zeile " & ustack(r) & _
        " gesetzt. Stackindex " & r & ".")
    GoTo continue_do
  Case "bun"
    pc = Range("Programmcode").Offset(pc, pcol_argument)
    debug_ausgabe ("Verzweige nach " & pc & ".")
    GoTo continue_do
  Case "cla"
    acc = 0
    debug_ausgabe ("acc := 0")
  Case "dac"
    acc = acc - 1
    debug_ausgabe ("acc := acc - 1")
  Case "div"
    v = Range("Programmcode").Offset(pc, pcol_argument)
    If Not IsNumeric(v) Then
      If st.exists(v) Then
        v = st(v)
      Else
        Call debug_ausgabe("Unbekanntes Argument '" & v & "' in Zeile " & pc & ". Abbruch!", True)
        Exit Sub
      End If
    End If
    acc = acc / Range("Programmcode").Offset(v, pcol_argument)
    debug_ausgabe ("acc := acc / " & Range("Programmcode").Offset(v, pcol_argument))
  Case "hlt"
    Call debug_ausgabe("Programmende erreicht in Zeile " & pc & ".", True)
    Exit Sub
  Case "iac"
    acc = acc + 1
    debug_ausgabe ("acc := acc + 1")
  Case "lda"
    v = Range("Programmcode").Offset(pc, pcol_argument)
    If Not IsNumeric(v) Then
      If st.exists(v) Then
        v = st(v)
      Else
        debug_ausgabe ("Unbekanntes Argument '" & v & "' in Zeile " & pc & ". Abbruch!")
```

```
        End
      End If
    End If
    acc = Range("Programmcode").Offset(v, pcol_argument)
    debug_ausgabe ("acc := " & acc)
  Case "mul"
    v = Range("Programmcode").Offset(pc, pcol_argument)
    If Not IsNumeric(v) Then
      If st.exists(v) Then
        v = st(v)
      Else
        debug_ausgabe ("Unbekanntes Argument '" & v & "' in Zeile " & pc & ". Abbruch!")
        End
      End If
    End If
    acc = acc * Range("Programmcode").Offset(v, pcol_argument)
    debug_ausgabe ("acc := acc * " & Range("Programmcode").Offset(v, pcol_argument))
  Case "out"
    v = Range("Programmcode").Offset(pc, pcol_argument)
    If Not IsNumeric(v) Then
      If st.exists(v) Then
        v = st(v)
      Else
        debug_ausgabe ("Unbekanntes Argument '" & v & "' in Zeile " & pc & ". Abbruch!")
        End
      End If
    End If
    Range("Ausgabebereich").Offset(i) = Range("Programmcode").Offset(v, pcol_argument)
    i = i + 1
  Case "ret"
    pc = ustack(r)
    r = r - 1
    debug_ausgabe ("Unterprogrammrücksprung nach '" & pc & _
        "'. Stackindex " & r & ".")
    GoTo continue_do
  Case "sta"
    v = Range("Programmcode").Offset(pc, pcol_argument)
    If Not IsNumeric(v) Then
      If st.exists(v) Then
        v = st(v)
      Else
        Call debug_ausgabe("Unbekanntes Argument '" & v & "' in Zeile " & pc & ". Abbruch!", True)
        Exit Sub
      End If
    End If
    Range("Programmcode").Offset(v, pcol_argument) = acc
    debug_ausgabe ("Argument in Zeile " & v & " wurde auf acc = " & acc & " gesetzt.")
  Case "sub"
    v = Range("Programmcode").Offset(pc, pcol_argument)
    If Not IsNumeric(v) Then
      If st.exists(v) Then
        v = st(v)
      Else
        Call debug_ausgabe("Unbekanntes Argument '" & v & "' in Zeile " & pc & ". Abbruch!", True)
        Exit Sub
      End If
    End If
    acc = acc - Range("Programmcode").Offset(v, pcol_argument)
    debug_ausgabe ("acc := acc - " & Range("Programmcode").Offset(v, pcol_argument))
  Case Else
    Call debug_ausgabe("Ungültiger OpCode '" & op & "' in Zeile " & pc & ". Abbruch!", True)
    Exit Sub
  End Select

  pc = pc + 1

  b_end = (Range("Programmcode").Offset(pc, pcol_label) = "" And _
          Range("Programmcode").Offset(pc, pcol_opcode) = "" And _
          Range("Programmcode").Offset(pc, pcol_argument) = "")

Loop Until b_end

End Sub

Sub debug_ausgabe(s As String, Optional force As Boolean = False)
If dbg Or force Then
  Range("Ausgabebereich").Offset(i) = s
  i = i + 1
End If
End Sub
```

Mortality Annuities

If you like to calculate annuities of a mortality table you have several choices::

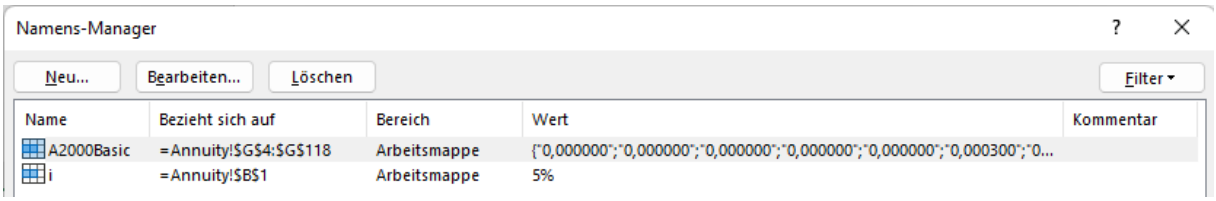
	A	B	C	D	E
1	i	5%			
2	Age	Formula	VBA	NPV Offset	NPV Index
3	65	11,31296262	11,31296262	11,31296262	11,31296262
4	66	10,99859672	10,99859672	10,99859672	10,99859672
5	67	10,66517834	10,66517834	10,66517834	10,66517834
6	68	10,31155279	10,31155279	10,31155279	10,31155279
7	69	10,04809227	10,04809227	10,04809227	10,04809227
8	70	9,765813149	9,765813149	9,765813149	9,765813149
9	71	9,463371231	9,463371231	9,463371231	9,463371231
10	72	9,139326319	9,139326319	9,139326319	9,139326319
50	112	0,34531908	0,34531908	0,34531908	0,34531908
51	113	0,20861678	0,20861678	0,20861678	0,20861678
52	114	0,095238095	0,095238095	0,095238095	0,095238095

Worksheet Formulas		
Range	Formula	
A4:A52	**A4**	=A3+1
B3:B52	**B3**	=SUMPRODUCT(TRANSPOSE((1+i) ^ -TRANSPOSE(ROW(OFFSET(A2000Basic, A3, , ROWS(A2000Basic) - A3)) - A3 - ROW(INDEX(A2000Basic, 1)) + 1)), EXP(MMULT(1 - (ROW(OFFSET(A2000Basic, A3, , ROWS(A2000Basic) - A3)) < TRANSPOSE(ROW(OFFSET(A2000Basic, A3, , ROWS(A2000Basic) - A3)))), LN(1 - OFFSET(A2000Basic, A3, , ROWS(A2000Basic) - A3)))))
C3:C52	**C3**	=LiveAnnuityPV(A3,i,A2000Basic)
D3:D52	**D3**	=NPV(i,OFFSET(A2000Basic,A3,1+A3,120-A3+1))
E3:E52	**E3**	=NPV(i,INDEX($H$4:$DR$118,1+A3,1+A3):INDEX($H$118:$DR$118,1,1+A3))
H4,I5,J6,..,DR118	**H4**	=1-$G4
H5:H118,I6:I118,..,DQ118	**H5**	=(1-$G5)*H4

Complex Array Formula (Worst)

Apply a complex one-cell array formula in cells B3:B52.

The names are defined as follows:



Simple user defined function in VBA (Better)

A better way by far is a simple user-defined VBA function which is being called in cells C3:C52.

```
Function LiveAnnuityPV(lYears As Long, dInterestRate As Double, _
    rMortalityTable As Range) As Double
'(C) (P) by Bernd Plumhoff 22-Mar-2014 PB V0.1
Dim j As Long
Dim dSum As Double, dProd As Double, dPV As Double

dProd = 1#
dPV = 1#
For j = 1 To rMortalityTable.Count - lYears
    dPV = dPV / (1# + dInterestRate)
    dProd = dProd * (1# - rMortalityTable(j + lYears))
    dSum = dSum + dPV * dProd
Next j

LiveAnnuityPV = dSum

End Function
```

Pre-calculated Table plus an NPV Formula (Maybe Best)

The fastest and maybe best approach seems to be a pre-calculated table and an NPV formula in cells D3:D52.

	F	G	H	I	J	K	L	M	N
2			Precalculated values						
3	Age	Male qx	0	1	2	3	4	5	6
4	0	0,000000	1,000000						
5	1	0,000000	1	1,000000					
6	2	0,000000	1	1	1,000000				
7	3	0,000000	1	1	1	1,000000			
8	4	0,000000	1	1	1	1	1,000000		
9	5	0,000300	0,9997	0,9997	0,9997	0,9997	0,9997	0,999700	
10	6	0,000300	0,99940009	0,99940009	0,99940009	0,99940009	0,99940009	0,99940009	0,999700

Please note: The OFFSET worksheet formula is slightly faster than the INDEX worksheet formula. On the other hand it is volatile, i.e. it is getting recalculated each time you press F9 and not just when an input value is getting updated.

With FastExcel © you can profile your worksheet. In this case we see that the NPV approach is about 45 times faster than the suboptimal array formula and about 27 times faster than the VBA solution:

	A	B	C	D	E	F	G	H	I	J	K
13	***WorkSheet Areas Profile for***			**Annuity**							
14	**Address**		**Count of**		**Calc Millisecs**		**MicroSecs**	**Area/Col %**	**FormulaLength**		
15	**Area**	**Formula**	**ArrayCells**	**CondFmt**	**Area/Col**	**Sheet**	**/Formula**	**of Sheet**	**Total**	**Average**	**Maximum**
133	**Column**										
134	B	50	50	0	12,01	26,43	240,29	45,5%	14.987	299,74	301
135	C	50	0	0	7,21	26,43	144,27	27,3%	1.593	31,86	32
137	E	50	0	0	0,31	26,43	6,27	1,2%	3.429	68,58	69
138	D	50	0	0	0,27	26,43	5,42	1,0%	2.279	45,58	46

Financial Mathematics – Options

A classic book on option valuation is: Les Clewlow and Chris Strickland: Implementing Derivatives Models (ISBN 0471966517). Unfortunately, this book contains a variety of errors, and the 8-page error list from November 23, 2000, is unfortunately no longer publicly available. Some corrections for the first edition:

Page	Row	Correction
xvi	9, column 3	0v should be C
12	15	Si,j = Su^jd^(**n**-j)
24	2 from top in figure 2.12	set coefficients
70	4 from bottom in figure 3.13	for j =Nj-2 downto -N**j+1** do
70	Insert before last row of figure 3.13	C[1,-Nj] = C[1,-Nj + 1] - lambda_L
75	4 from bottom in figure 3.16	for j =Nj-2 downto -N**j+1** do
75	Insert before last row of figure 3.16	C[1,-Nj] = C[1,-Nj + 1] - lambda_L
85	2 from bottom in figure 4.2	SD = sqrt((-sum_CT2 + sum_CT * sum_CT / M) **)** * exp(-2*r*T) / (M - 1)
89	2 from bottom in figure 4.5	SD = sqrt((-sum_CT2 + sum_CT * sum_CT / M) **)** * exp(-2*r*T) / (M - 1)
99	Delete row 9 from top	lnS = ln(S) = 4.6052 **<- delete this row**
113	13 from top in figure 4.19	**Insert before for j loop** V1 = sig1*sig1; V2 = sig2*sig2
113	18 from top in figure 4.19	**Insert before for i loop** Vt1 = V1; Vt2 = V2
118	Last row	a = **m/N** * ln(Gt) ...
120	11 from bottom in figure 4.24	G = productSt **^(1**/N)

I implemented some algorithms presented in this book with Excel / VBA.

The binomial Method

	A	B	C	D	E	F	G	H	I	J	K	M	O
1	**Chapter 2 - The Binomial Method**												
2													
3	Multiplicative Binomial Tree Valuation of a European Call												
4	K	T	S	r	N	u	d					Function Result	Correct Result
5	100	1	100	0,06	3	1,1	0,9091					10,14546215	10,14546215
6													
7	Multiplicative Binomial Tree Valuation of an American Put												
8	K	T	S	r	N	u	d					Function Result	Correct Result
9	100	1	100	0,06	3	1,1	0,9091					4,654346769	4,654346769
10													
11	General Additive Binomial Valuation of a European Call												
12	K	T	S	sig	r	N						Function Result	Correct Result
13	100	1	100	0,2	0,06	3						11,59199121	11,59199121
14													
15	General Additive Binomial Valuation of an American Put												
16	K	T	S	sig	r	N						Function Result	Correct Result
17	100	1	100	0,2	0,06	3						6,162109199	6,162109199
18													
19	Equal Jump General Additive Binomial Valuation of an American Put												
20	K	T	S	sig	r	N						Function Result	Correct Result
21	100	1	100	0,2	0,06	3						6,162109199	6,162109199
22													
23	General Additive Binomial Valuation of an American Down-and-Out Call												
24	K	T	S	sig	r	H	N					Function Result	Correct Result
25	100	1	100	0,2	0,06	95	3					9,995775103	9,995775103
26													
27	American Spread Call Option by Two-variable Binomial												
28	K	T	S1	S2	sig1	sig2	div1	div2	rho	r	N	Function Result	Correct Result
29	1	1	100	100	0,2	0,3	0,03	0,04	0,5	0,06	3	10,04478653	10,04478653

Chapter 2 Program Code

```
Function European_Call_MBTV( _
           K As Double, _
           T As Double, _
           S As Double, _
           r As Double, _
           N As Long, _
           u As Double, _
           d As Double) As Double

'Multiplicative Binomial Tree Valuation of a European Call
'Clewlow/Strickland: Implementing Derivatives Models, page 13
'ISBN 0-471-96651-7

Dim dt                        As Double
Dim p                         As Double
Dim disc                      As Double
ReDim St(0 To N) As Double
ReDim C(0 To N) As Double
Dim i As Long, j              As Long

'precompute constants

dt = T / N
p = (Exp(r * dt) - d) / (u - d)
disc = Exp(-r * dt)

'Debug.Print dt, p, disc
'initialise asset prices at maturity time step N

St(0) = S * d ^ N
For j = 1 To N
  St(j) = St(j - 1) * u / d
Next j

'Debug.Print St(0), St(1), St(2), St(3)
'initialise option values at maturity

For j = 0 To N
  C(j) = St(j) - K: If C(j) < 0# Then C(j) = 0#
Next j

'Debug.Print C(0), C(1), C(2), C(3)
'step back through the tree

For i = N - 1 To 0 Step -1
  For j = 0 To i
    C(j) = disc * (p * C(j + 1) + (1 - p) * C(j))
    'Debug.Print C(j);
  Next j
  'Debug.Print
Next i

European_Call_MBTV = C(0)

End Function

Function American_Put_MBTV( _
           K As Double, _
           T As Double, _
           S As Double, _
           r As Double, _
           N As Long, _
           u As Double, _
           d As Double) As Double

'Multiplicative Binomial Tree Valuation of an American Put
'Clewlow/Strickland: Implementing Derivatives Models, page 15

Dim dt                        As Double
Dim p                         As Double
Dim disc                      As Double
ReDim St(0 To N) As Double
ReDim C(0 To N) As Double
Dim i As Long, j              As Long

'precompute constants

dt = T / N
p = (Exp(r * dt) - d) / (u - d)
disc = Exp(-r * dt)

'Debug.Print dt, p, disc

'initialise asset prices at maturity time step N

St(0) = S * d ^ N
```

```
For j = 1 To N
  St(j) = St(j - 1) * u / d
Next j

'Debug.Print St(0), St(1), St(2), St(3)

'initialise option values at maturity

For j = 0 To N
  C(j) = K - St(j): If C(j) < 0# Then C(j) = 0#
Next j

'Debug.Print C(0), C(1), C(2), C(3)

'step back through the tree applying the early exercise condition

For i = N - 1 To 0 Step -1
  For j = 0 To i
    C(j) = disc * (p * C(j + 1) + (1 - p) * C(j))
    St(j) = St(j) / d
    If C(j) < K - St(j) Then C(j) = K - St(j)
    'Debug.Print C(j);
  Next j
  'Debug.Print
Next i

American_Put_MBTV = C(0)

End Function

Function European_Call_GABV( _
           K As Double, _
           T As Double, _
           S As Double, _
           sig As Double, _
           r As Double, _
           N As Long) As Double

'General Additive Binomial Valuation of a European Call
'Clewlow/Strickland: Implementing Derivatives Models, page 22

Dim dt                      As Double
Dim pu As Double, pd        As Double
Dim dxu As Double, dxd As Double, nu As Double
Dim disc                    As Double
ReDim St(0 To N) As Double
ReDim C(0 To N) As Double
Dim i As Long, j            As Long

'set coefficients - Trigeorgis

dt = T / N
nu = r - 0.5 * sig ^ 2#
dxu = Sqr(sig ^ 2# * dt + (nu * dt) ^ 2#)
dxd = -dxu
pu = 0.5 + 0.5 * (nu * dt / dxu)
pd = 1# - pu

'precompute constants

disc = Exp(-r * dt)

'initialise asset prices at maturity N

St(0) = S * Exp(N * dxd)
For j = 1 To N
  St(j) = St(j - 1) * Exp(dxu - dxd)
Next j
'Debug.Print St(0), St(1), St(2), St(3)

'initialise option values at maturity

For j = 0 To N
  C(j) = St(j) - K: If C(j) < 0# Then C(j) = 0#
Next j
'Debug.Print C(0), C(1), C(2), C(3)

'step back through the tree

For i = N - 1 To 0 Step -1
  For j = 0 To i
    C(j) = disc * (pu * C(j + 1) + pd * C(j))
    'Debug.Print C(j);
  Next j
  'Debug.Print
Next i

European_Call_GABV = C(0)

End Function
```

```
Function American_Put_GABV( _
           K As Double, _
           T As Double, _
           S As Double, _
           sig As Double, _
           r As Double, _
           N As Long) As Double

'General Additive Binomial Valuation of an American Put
'Clewlow/Strickland: Implementing Derivatives Models, page 24

Dim dt                        As Double
Dim pu As Double, pd          As Double
Dim dxu As Double, dxd As Double, nu As Double
Dim disc                      As Double
ReDim St(0 To N) As Double
ReDim C(0 To N) As Double
Dim i As Long, j              As Long

'set coefficients - Trigeorgis

dt = T / N
nu = r - 0.5 * sig ^ 2#
dxu = Sqr(sig ^ 2 * dt + (nu * dt) ^ 2#)
dxd = -dxu
pu = 0.5 + 0.5 * (nu * dt / dxu)
pd = 1# - pu

'precompute constants

disc = Exp(-r * dt)

'initialise asset prices at maturity N

St(0) = S * Exp(N * dxd)
For j = 1 To N
  St(j) = St(j - 1) * Exp(dxu - dxd)
Next j
'Debug.Print St(0), St(1), St(2), St(3)

'initialise option values at maturity

For j = 0 To N
  C(j) = K - St(j): If C(j) < 0# Then C(j) = 0#
Next j
'Debug.Print C(0), C(1), C(2), C(3)

'step back through the tree applying the early exercise condition

For i = N - 1 To 0 Step -1
  For j = 0 To i
    C(j) = disc * (pu * C(j + 1) + pd * C(j))
    'adjust asset price to current time step
    St(j) = St(j) / Exp(dxd)
    'apply the early exercise condition
    If K - St(j) > C(j) Then C(j) = K - St(j)
    'Debug.Print C(j);
  Next j
  'Debug.Print
Next i

American_Put_GABV = C(0)

End Function

Function American_Put_EJGABV( _
           K As Double, _
           T As Double, _
           S As Double, _
           sig As Double, _
           r As Double, _
           N As Long) As Double

'Equal Jump General Additive Binomial Valuation of an American Put
'Clewlow/Strickland: Implementing Derivatives Models, page 28

Dim dt                        As Double
Dim pu As Double, pd          As Double
Dim dx As Double, nu          As Double
Dim disc As Double, edx       As Double
Dim dpu As Double, dpd        As Double
ReDim St(-N To N) As Double
ReDim C(-N To N) As Double
Dim i As Long, j              As Long

'set coefficients - Trigeorgis

dt = T / N
nu = r - 0.5 * sig ^ 2#
```

```
dx = Sqr(sig ^ 2# * dt + (nu * dt) ^ 2#)
pu = 0.5 + 0.5 * (nu * dt / dx)
pd = 1# - pu

'precompute constants

disc = Exp(-r * dt)
dpu = disc * pu
dpd = disc * pd
edx = Exp(dx)

'initialise asset prices at maturity N

St(-N) = S * Exp(-N * dx)
For j = -N + 1 To N
  St(j) = St(j - 1) * edx
Next j

'initialise option values at maturity

For j = -N To N Step 2
  C(j) = K - St(j): If C(j) < 0# Then C(j) = 0#
Next j

'step back through the tree applying the early exercise condition

For i = N - 1 To 0 Step -1
  For j = -i To i Step 2
    C(j) = dpd * C(j - 1) + dpu * C(j + 1)
    'apply the early exercise condition
    If K - St(j) > C(j) Then C(j) = K - St(j)
  Next j
Next i

American_Put_EJGABV = C(0)

End Function

Function American_Down_and_Out_Call_GABV( _
           K As Double, _
           T As Double, _
           S As Double, _
           sig As Double, _
           r As Double, _
           H As Double, _
           N As Long) As Double

'General Additive Binomial Valuation of an American Down-and-Out Call
'Clewlow/Strickland: Implementing Derivatives Models, page 41

Dim dt                        As Double
Dim pu As Double, pd          As Double
Dim dpu As Double, dpd        As Double
Dim dxd As Double, dxu As Double, nu As Double
Dim edxud                     As Double
Dim edxd                      As Double
Dim disc                      As Double
ReDim St(0 To N) As Double
ReDim C(0 To N) As Double
Dim i As Long, j              As Long

'set coefficients - Trigeorgis

dt = T / N
nu = r - 0.5 * sig ^ 2#
dxu = Sqr(sig ^ 2# * dt + (nu * dt) ^ 2#)
dxd = -dxu
pu = 0.5 + 0.5 * (nu * dt / dxu)
pd = 1# - pu

'precompute constants

disc = Exp(-r * dt)
dpu = disc * pu
dpd = disc * pd
edxud = Exp(dxu - dxd)
edxd = Exp(dxd)

'initialise asset prices at maturity N

St(0) = S * Exp(N * dxd)
For j = 1 To N
  St(j) = St(j - 1) * edxud
Next j

'initialise option values at maturity

For j = 0 To N
  If St(j) > H Then
    C(j) = St(j) - K: If C(j) < 0# Then C(j) = 0#
```

```
  Else
    C(j) = 0#
  End If
Next j

'step back through the tree applying the barrier
'and early exercise condition

For i = N - 1 To 0 Step -1
  For j = 0 To i
    'adjust asset price to current time step
    St(j) = St(j) / edxd

    If St(j) > H Then
      C(j) = dpd * C(j) + dpu * C(j + 1)

      'apply the early exercise condition
      If C(j) < St(j) - K Then C(j) = St(j) - K
    Else
      C(j) = 0#
    End If
  Next j
Next i

American_Down_and_Out_Call_GABV = C(0)

End Function

Function American_Spread_Call_Option_bTvB( _
           K As Double, _
           T As Double, _
           S1 As Double, _
           S2 As Double, _
           sig1 As Double, _
           sig2 As Double, _
           div1 As Double, _
           div2 As Double, _
           rho As Double, _
           r As Double, _
           N As Long) As Double

'American Spread Call Option by Two-variable Binomial
'Clewlow/Strickland: Implementing Derivatives Models, page 46

Dim dt                        As Double
Dim puu As Double, pdu        As Double
Dim pud As Double, pdd        As Double
Dim dx1 As Double, dx2        As Double
Dim nu1 As Double, nu2        As Double
Dim edx1 As Double, edx2      As Double
Dim disc                      As Double
ReDim s1t(-N To N) As Double
ReDim s2t(-N To N) As Double
ReDim C(-N To N, -N To N) As Double
Dim i As Long, j As Long, k1 As Long

'precompute constants

dt = T / N
nu1 = r - div1 - 0.5 * sig1 ^ 2#
nu2 = r - div2 - 0.5 * sig2 ^ 2#
dx1 = sig1 * Sqr(dt)
dx2 = sig2 * Sqr(dt)
disc = Exp(-r * dt)
puu = (dx1 * dx2 + (dx2 * nu1 + dx1 * nu2 + rho * sig1 * sig2) * dt) / _
      (4 * dx1 * dx2) * disc
pud = (dx1 * dx2 + (dx2 * nu1 - dx1 * nu2 - rho * sig1 * sig2) * dt) / _
      (4 * dx1 * dx2) * disc
pdu = (dx1 * dx2 - (dx2 * nu1 - dx1 * nu2 + rho * sig1 * sig2) * dt) / _
      (4 * dx1 * dx2) * disc
pdd = (dx1 * dx2 - (dx2 * nu1 + dx1 * nu2 - rho * sig1 * sig2) * dt) / _
      (4 * dx1 * dx2) * disc
edx1 = Exp(dx1)
edx2 = Exp(dx2)

'initialise asset prices at time step N

s1t(-N) = S1 * Exp(-N * dx1)
s2t(-N) = S2 * Exp(-N * dx2)
For j = -N + 1 To N
  s1t(j) = s1t(j - 1) * edx1
  s2t(j) = s2t(j - 1) * edx2
Next j

'initialise option values at maturity

For j = -N To N Step 2
  For k1 = -N To N Step 2
    C(j, k1) = s1t(j) - s2t(k1) - K
    If C(j, k1) < 0# Then C(j, k1) = 0#
```

```
  Next k1
Next j

'step back through the tree applying the early exercise condition

For i = N - 1 To 0 Step -1
  For j = -i To i Step 2
    For k1 = -i To i Step 2
      C(j, k1) = pdd * C(j - 1, k1 - 1) + pud * C(j + 1, k1 - 1) + _
                 pdu * C(j - 1, k1 + 1) + puu * C(j + 1, k1 + 1)
      'apply the early exercise condition
      If s1t(j) - s2t(k1) - K > C(j, k1) Then C(j, k1) = s1t(j) - s2t(k1) - K
    Next k1
  Next j
Next i

American_Spread_Call_Option_bTvB = C(0, 0)

End Function
```

Trinomial Trees and Finite Difference Methods

	A	B	C	D	E	F	G	H	I	M	N	O
31	**Chapter 3 - Trinomial Trees and Finite Difference Methods**											
32												
33	European Call Option by Trinomial Tree											
34	K	T	S	sig	r	div	N	dx		Function Result		Correct Result
35	100	1	100	0,2	0,06	0,03	3	0,2		8,425336177		8,425336177
36												
37	European Call Option by Explicit Finite Difference Method											
38	K	T	S	sig	r	div	N	Nj	dx	Function Result		Correct Result
39	100	1	100	0,2	0,06	0,03	3	3	0,2	8,545479558		8,545479558
40												
41	American Put Option by Explicit Finite Difference Method											
42	K	T	S	sig	r	div	N	Nj	dx	Function Result		Correct Result
43	100	1	100	0,2	0,06	0,03	3	3	0,2	6,005828214		6,005828214
44												
45	American Put Option by Implicit Finite Difference Method											
46	K	T	S	sig	r	div	N	Nj	dx	Function Result		Correct Result
47	100	1	100	0,2	0,06	0,03	3	3	0,2	4,922117823		4,922117823
48												
49	American Put Option by Crank-Ncolson Finite Difference Method											
50	K	T	S	sig	r	div	N	Nj	dx	Function Result		Correct Result
51	100	1	100	0,2	0,06	0,03	3	3	0,2	5,418377337		5,418377337

Chapter 3 Program Code

```
Function European_Call_Option_bTT( _
           K As Double, _
           T As Double, _
           S As Double, _
           sig As Double, _
           r As Double, _
           div As Double, _
           N As Long, _
           dx As Double) As Double

'European Call Option by Trinomial Tree
'Clewlow/Strickland: Implementing Derivatives Models, page 54

Dim dt                        As Double
Dim pu As Double, pm As Double, pd As Double
Dim nu As Double, edx         As Double
Dim disc                      As Double
ReDim St(-N To N) As Double
ReDim C(0 To N, -N To N) As Double
Dim i As Long, j              As Long

'set coefficients - Trigeorgis

dt = T / N
nu = r - div - 0.5 * sig ^ 2#
edx = Exp(dx)
pu = 0.5 * ((sig ^ 2# * dt + nu ^ 2# * dt ^ 2#) / dx ^ 2# + nu * dt / dx)
pm = 1# - (sig ^ 2# * dt + nu ^ 2# * dt ^ 2#) / dx ^ 2#
pd = 0.5 * ((sig ^ 2# * dt + nu ^ 2# * dt ^ 2#) / dx ^ 2# - nu * dt / dx)
disc = Exp(-r * dt)
```

```
'initialise asset prices at maturity N

St(-N) = S * Exp(-N * dx)
For j = -N + 1 To N
  St(j) = St(j - 1) * edx
Next j

'initialise option values at maturity

For j = -N To N
  If St(j) - K > 0# Then
    C(N, j) = St(j) - K
  Else
    C(N, j) = 0#
  End If
Next j

'step back through lattice

For i = N - 1 To 0 Step -1
  For j = -i To i
    C(i, j) = disc * (pu * C(i + 1, j + 1) + pm * C(i + 1, j) + pd * C(i + 1, j - 1))
  Next j
Next i

European_Call_Option_bTT = C(0, 0)

End Function


Function European_Call_Option_bEFDM( _
           K As Double, _
           T As Double, _
           S As Double, _
           sig As Double, _
           r As Double, _
           div As Double, _
           N As Long, _
           Nj As Long, _
           dx As Double) As Double

'European Call Option by Explicit Finite Difference Method
'Clewlow/Strickland: Implementing Derivatives Models, page 60

Dim dt                       As Double
Dim pu As Double, pm As Double, pd As Double
Dim nu As Double, edx        As Double
ReDim St(-Nj To Nj) As Double
ReDim C(0 To N, -Nj To Nj) As Double
Dim i As Long, j             As Long

'set coefficients - Trigeorgis

dt = T / N
nu = r - div - 0.5 * sig ^ 2#
edx = Exp(dx)
pu = 0.5 * dt * ((sig / dx) ^ 2# + nu / dx)
pm = 1# - dt * ((sig / dx) ^ 2# + r)
pd = 0.5 * dt * ((sig / dx) ^ 2# - nu / dx)

'initialise asset prices at maturity N

St(-Nj) = S * Exp(-Nj * dx)
For j = -Nj + 1 To Nj
  St(j) = St(j - 1) * edx
Next j

'initialise option values at maturity

For j = -Nj To Nj
  If St(j) - K > 0# Then
    C(N, j) = St(j) - K
  Else
    C(N, j) = 0#
  End If
Next j

'step back through lattice

For i = N - 1 To 0 Step -1
  For j = -Nj + 1 To Nj - 1
    C(i, j) = pu * C(i + 1, j + 1) + pm * C(i + 1, j) + pd * C(i + 1, j - 1)
  Next j
  'boundary conditions
  C(i, -Nj) = C(i, -Nj + 1)
  C(i, Nj) = C(i, Nj - 1) + St(Nj) - St(Nj - 1)
Next i

European_Call_Option_bEFDM = C(0, 0)
```

```
End Function


Function American_Put_Option_bEFDM( _
           K As Double, _
           T As Double, _
           S As Double, _
           sig As Double, _
           r As Double, _
           div As Double, _
           N As Long, _
           Nj As Long, _
           dx As Double) As Double

'American Put Option by Explicit Finite Difference Method
'Clewlow/Strickland: Implementing Derivatives Models, page 62

Dim dt                        As Double
Dim pu As Double, pm As Double, pd As Double
Dim nu As Double, edx         As Double
ReDim St(-Nj To Nj) As Double
ReDim C(0 To 1, -Nj To Nj) As Double
Dim i As Long, j              As Long

'Precompute constants

dt = T / N
nu = r - div - 0.5 * sig ^ 2#
edx = Exp(dx)
pu = 0.5 * dt * ((sig / dx) ^ 2# + nu / dx)
pm = 1# - dt * (sig / dx) ^ 2# - r * dt
pd = 0.5 * dt * ((sig / dx) ^ 2# - nu / dx)

'Initialise asset prices at maturity N

St(-Nj) = S * Exp(-Nj * dx)
For j = -Nj + 1 To Nj
  St(j) = St(j - 1) * edx
Next j

'Initialise option values at maturity

For j = -Nj To Nj
  If K - St(j) > 0# Then
    C(0, j) = K - St(j)
  Else
    C(0, j) = 0#
  End If
Next j

'Step back through lattice

For i = N - 1 To 0 Step -1

  For j = -Nj + 1 To Nj - 1
    C(1, j) = pu * C(0, j + 1) + pm * C(0, j) + pd * C(0, j - 1)
  Next j

  'Boundary conditions

  C(1, -Nj) = C(1, -Nj + 1) + St(-Nj + 1) - St(-Nj)
  C(1, Nj) = C(1, Nj - 1)

  'Apply early exercise condition

  For j = -Nj To Nj
    C(0, j) = K - St(j)
    If C(0, j) < C(1, j) Then C(0, j) = C(1, j)
  Next j

Next i

American_Put_Option_bEFDM = C(0, 0)

End Function


Function American_Put_Option_bIFDM( _
           K As Double, _
           T As Double, _
           S As Double, _
           sig As Double, _
           r As Double, _
           div As Double, _
           N As Long, _
           Nj As Long, _
           dx As Double) As Double

'American Put Option by Implicit Finite Difference Method
```

```
'Clewlow/Strickland: Implementing Derivatives Models, page 69

Dim dt                        As Double
Dim pu As Double, pm As Double, pd As Double
Dim nu As Double, edx         As Double
Dim lambda_L As Double, lambda_U As Double
ReDim St(-Nj To Nj) As Double
ReDim C(0 To 1, -Nj To Nj) As Double
Dim i As Long, j              As Long
ReDim pmp(-Nj To Nj) As Double
ReDim pp(-Nj To Nj) As Double

'Precompute constants

dt = T / N
nu = r - div - 0.5 * sig ^ 2#
edx = Exp(dx)
pu = -0.5 * dt * ((sig / dx) ^ 2# + nu / dx)
pm = 1# + dt * (sig / dx) ^ 2# + r * dt
pd = -0.5 * dt * ((sig / dx) ^ 2# - nu / dx)

'Initialise asset prices at maturity N

St(-Nj) = S * Exp(-Nj * dx)
For j = -Nj + 1 To Nj
  St(j) = St(j - 1) * edx
Next j

'Initialise option values at maturity

For j = -Nj To Nj
  If K - St(j) > 0# Then
    C(0, j) = K - St(j)
  Else
    C(0, j) = 0#
  End If
Next j

'Compute derivative boundary condition

lambda_L = St(-Nj) - St(-Nj + 1)
lambda_U = 0#

'Step back through lattice

For i = N - 1 To 0 Step -1

  'Solve implicit tridiagonal system

  'Substitute boundary condition at j = -Nj into j = -Nj + 1

  pmp(-Nj + 1) = pm + pd
  pp(-Nj + 1) = C(0, -Nj + 1) + pd * lambda_L

  'Eliminate upper diagonal

  For j = -Nj + 2 To Nj - 1

    pmp(j) = pm - pu * pd / pmp(j - 1)
    pp(j) = C(0, j) - pp(j - 1) * pd / pmp(j - 1)

  Next j

  'Use boundary condition at j = Nj and equation at j = Nj - 1

  C(1, Nj) = (pp(Nj - 1) + pmp(Nj - 1) * lambda_U) / (pu + pmp(Nj - 1))
  C(1, Nj - 1) = C(1, Nj) - lambda_U

  'Back-substitution

  For j = Nj - 2 To -Nj + 1 Step -1
    C(1, j) = (pp(j) - pu * C(1, j + 1)) / pmp(j)
  Next j
  C(1, -Nj) = C(1, -Nj + 1) - lambda_L

  'Apply early exercise condition

  For j = -Nj To Nj
    C(0, j) = K - St(j)
    If C(0, j) < C(1, j) Then C(0, j) = C(1, j)
  Next j

Next i

American_Put_Option_bIFDM = C(0, 0)

End Function
```

```
Function American_Put_Option_bCNFDM( _
           K As Double, _
           T As Double, _
           S As Double, _
           sig As Double, _
           r As Double, _
           div As Double, _
           N As Long, _
           Nj As Long, _
           dx As Double) As Double

'American Put Option by Crank-Nicolson Finite Difference Method
'Clewlow/Strickland: Implementing Derivatives Models, page 74

Dim dt                        As Double
Dim pu As Double, pm As Double, pd As Double
Dim nu As Double, edx         As Double
Dim lambda_L As Double, lambda_U As Double
ReDim St(-Nj To Nj) As Double
ReDim C(0 To 1, -Nj To Nj) As Double
Dim i As Long, j              As Long
ReDim pmp(-Nj To Nj) As Double
ReDim pp(-Nj To Nj) As Double

'Precompute constants
dt = T / N
nu = r - div - 0.5 * sig ^ 2#
edx = Exp(dx)
pu = -0.25 * dt * ((sig / dx) ^ 2# + nu / dx)
pm = 1# + 0.5 * dt * (sig / dx) ^ 2# + 0.5 * r * dt
pd = -0.25 * dt * ((sig / dx) ^ 2# - nu / dx)

'Initialise asset prices at maturity N
St(-Nj) = S * Exp(-Nj * dx)
For j = -Nj + 1 To Nj
  St(j) = St(j - 1) * edx
Next j

'Initialise option values at maturity
For j = -Nj To Nj
  If K - St(j) > 0# Then
    C(0, j) = K - St(j)
  Else
    C(0, j) = 0#
  End If
Next j

'Compute derivative boundary condition
lambda_L = St(-Nj) - St(-Nj + 1)
lambda_U = 0#

'Step back through lattice
For i = N - 1 To 0 Step -1

  'Solve Crank-Nicolson tridiagonal system
  'Substitute boundary condition at j = -Nj into j = -Nj + 1
  pmp(-Nj + 1) = pm + pd
  pp(-Nj + 1) = -pu * C(0, -Nj + 2) - (pm - 2) * C(0, -Nj + 1) - pd * C(0, -Nj) + pd * lambda_L

  'Eliminate upper diagonal
  For j = -Nj + 2 To Nj - 1

    pmp(j) = pm - pu * pd / pmp(j - 1)
    pp(j) = -pu * C(0, j + 1) - (pm - 2) * C(0, j) - pd * C(0, j - 1) - pp(j - 1) * pd / pmp(j - 1)

  Next j

  'Use boundary condition at j = Nj and equation at j = Nj - 1
  C(1, Nj) = (pp(Nj - 1) + pmp(Nj - 1) * lambda_U) / (pu + pmp(Nj - 1))
  C(1, Nj - 1) = C(1, Nj) - lambda_U

  'Back-substitution
  For j = Nj - 2 To -Nj + 1 Step -1
    C(1, j) = (pp(j) - pu * C(1, j + 1)) / pmp(j)
  Next j
  C(1, -Nj) = C(1, -Nj + 1) - lambda_L

  'Apply early exercise condition
  For j = -Nj To Nj
    C(0, j) = K - St(j)
    If C(0, j) < C(1, j) Then C(0, j) = C(1, j)
  Next j

Next i

American_Put_Option_bCNFDM = C(0, 0)

End Function
```

Monte Carlo Simulation

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
53	**Chapter 4 - Monte Carlo Simulation**														
54															
55	European Call Option by Monte Carlo with Black-Scholes														
56	K	T	S	sig	r	div	N	M					Function Result		Correct Result
57	100	1	100	0,2	0,06	0,03	10	100000					9,128737999		~9,1
58															
59	European Call Option by Monte Carlo with Black-Scholes and Antithetic Variance Reduction														
60	K	T	S	sig	r	div	N	M					Function Result		Correct Result
61	100	1	100	0,2	0,06	0,03	10	100000					9,134721275		~9,1
62															
63	European Call Option by Monte Carlo with Black-Scholes and Delta-based Control Variate														
64	K	T	S	sig	r	div	N	M					Function Result		Correct Result
65	100	1	100	0,2	0,06	0,03	10	100000					9,138508265		~9,1
66															
67	European Call Option by Monte Carlo with Black-Scholes and Antithetic and Delta-based Control Variates														
68	K	T	S	sig	r	div	N	M					Function Result		Correct Result
69	100	1	100	0,2	0,06	0,03	10	100000					9,14387695		~9,1
70															
71	European Call Option by Monte Carlo with Black-Scholes and Antithetic, Delta- and Gamma- based Control Variates														
72	K	T	S	sig	r	div	N	M					Function Result		Correct Result
73	100	1	100	0,2	0,06	0,03	10	100000					9,138388379		~9,1
74															
75	European Spread Call Option by Monte Carlo with Black-Scholes														
76	K	T	S1	S2	sig1	sig2	div1	div2	rho	r	N	M	Function Result		Correct Result
77	1	1	100	110	0,2	0,3	0,03	0,04	0,5	0,06	1	100000	6,533024448		~6,5
78															
79	European Spread Call Option by Monte Carlo with Black-Scholes and Stochastic Volatilities														
80	K	T	S1	S2	sig1	sig2	div1	div2	rhoz				Function Result		Correct Result
81	1	1	100	110	0,2	0,3	0,03	0,04	1	0,5	0,2	0,01	7,816963084		~7,0389
82	alpha1	alpha2	Vbar1	Vbar2	xi1	xi2	r		0,5	1	0,01	0,3			
83	1	2	0,04	0,09	0,05	0,06	0,06		0,2	0,01	1	0,3			
84	N	M							0,01	0,3	0,3	1			
85	10	10000													
86															
87	European Down and Out Call Option by Monte Carlo with Black-Scholes														
88	K	T	S	sig	r	div	H	N	M				Function Result		Correct Result
89	100	1	100	0,2	0,06	0,03	99	10	100000				5,059147995		~5

Monte Carlo Simulation Program Code

```
#Const MODULE_TEST = False     'Used for optional compiling of
'debugging/test parts - set to FALSE for production

Function European_Call_bMCBS( _
           K As Double, _
           T As Double, _
           S As Double, _
           sig As Double, _
           r As Double, _
           div As Double, _
           N As Long, _
           M As Long) As Double

'European Call Option by Monte Carlo with Black-Scholes
'Clewlow/Strickland: Implementing Derivatives Models, page 85
'12.04.2021 corrected: sum_CT2 = sum_CT2 + CT * CT

Dim dt                        As Double
Dim nudt As Double, sigsdt    As Double
Dim lnS As Double, lnSt       As Double
Dim sum_CT As Double, sum_CT2 As Double
Dim e                         As Double
Dim St As Double, CT          As Double
Dim SD As Double, SE          As Double
Dim i As Long, j              As Long

'Precompute constants

dt = T / N
```

```
nudt = (r - div - 0.5 * sig ^ 2#) * dt
sigsdt = sig * Sqr(dt)
lnS = Log(S)

sum_CT = 0#
sum_CT2 = 0#

For j = 1 To M

  'For each simulation

  lnSt = lnS

  For i = 1 To N

    'For each time step

    '        e = Application.WorksheetFunction.NormSInv(Rnd()) 'Standard normal sample
    e = Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + _
        Rnd() + Rnd() + Rnd() + Rnd() - 6#    'Standard normal sample
    lnSt = lnSt + nudt + sigsdt * e    'Evolve the stock price

  Next i

  St = Exp(lnSt)
  CT = St - K: If CT < 0# Then CT = 0#
  sum_CT = sum_CT + CT
  sum_CT2 = sum_CT2 + CT * CT

Next j

SD = Sqr((sum_CT2 - sum_CT * sum_CT / M) * Exp(-2# * r * T) / (M - 1))    'Standard deviation
SE = SD / Sqr(M)    'Standard error
European_Call_bMCBS = sum_CT / M * Exp(-r * T)

End Function

Function European_Call_bMCBS_AVR( _
           K As Double, _
           T As Double, _
           S As Double, _
           sig As Double, _
           r As Double, _
           div As Double, _
           N As Long, _
           M As Long) As Double

'European Call Option by Monte Carlo with Black-Scholes and Antithetic Variance Reduction
'Clewlow/Strickland: Implementing Derivatives Models, page 89

Dim dt                      As Double
Dim nudt As Double, sigsdt  As Double
Dim lnS As Double, LnSt1 As Double, LnSt2 As Double
Dim sum_CT As Double, sum_CT2 As Double
Dim e                       As Double
Dim St1 As Double, St2      As Double
Dim CT As Double, CT1 As Double, CT2 As Double
Dim SD As Double, SE        As Double
Dim i As Long, j            As Long

'Precompute constants

dt = T / N
nudt = (r - div - 0.5 * sig ^ 2#) * dt
sigsdt = sig * Sqr(dt)
lnS = Log(S)

sum_CT = 0#
sum_CT2 = 0#

For j = 1 To M

  'For each simulation

  LnSt1 = lnS
  LnSt2 = lnS

  For i = 1 To N

    'For each time step

    '        e = Application.WorksheetFunction.NormSInv(Rnd()) 'Standard normal sample
    e = Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + _
        Rnd() + Rnd() + Rnd() + Rnd() - 6#    'Standard normal sample
    LnSt1 = LnSt1 + nudt + sigsdt * e    'Evolve the stock price
    LnSt2 = LnSt2 + nudt + sigsdt * -e    'Evolve the stock price

  Next i

  St1 = Exp(LnSt1)
```

```
  St2 = Exp(LnSt2)
  CT1 = St1 - K: If CT1 < 0# Then CT1 = 0#
  CT2 = St2 - K: If CT2 < 0# Then CT2 = 0#
  CT = (CT1 + CT2) / 2#
  sum_CT = sum_CT + CT
  sum_CT2 = sum_CT + CT * CT

Next j

SD = Sqr(sum_CT * sum_CT / M - sum_CT2) * Exp(-2# * r * T) / (M - 1)    'Standard deviation
SE = SD / Sqr(M)    'Standard error
European_Call_bMCBS_AVR = sum_CT / M * Exp(-r * T)

End Function

Function European_Call_bMCBS_DCV( _
           K As Double, _
           T As Double, _
           S As Double, _
           sig As Double, _
           r As Double, _
           div As Double, _
           N As Long, _
           M As Long) As Double

'European Call Option by Monte Carlo with Black-Scholes and Delta-based Control Variate
'Clewlow/Strickland: Implementing Derivatives Models, page 97

Dim dt                        As Double
Dim nudt As Double, sigsdt As Double, erddt As Double
Dim sum_CT As Double, sum_CT2 As Double
Dim e                         As Double
Dim St As Double, Stn         As Double
Dim CT                        As Double
Dim SD As Double, SE          As Double
Dim beta1 As Double, t1       As Double
Dim cv                        As Double
Dim delta1                    As Double
Dim i As Long, j              As Long

'Dim eps As Variant
'eps = Array(0.5391, 0.1692, 0.4583, 0.4164, 0.9223, 0.1597, 0.4011, -0.2382, 1.2152, -0.5506)

'Precompute constants

dt = T / N
nudt = (r - div - 0.5 * sig ^ 2#) * dt
sigsdt = sig * Sqr(dt)
erddt = Exp((r - div) * dt)
beta1 = -1#
'Debug.Print "dt = " & dt & ", nudt = " & nudt & ", sigsdt = " & sigsdt & _
  '  ", erddt = " & erddt & ", beta1 = " & beta1

sum_CT = 0#
sum_CT2 = 0#

For j = 1 To M

  'For each simulation

  St = S
  cv = 0#

  For i = 1 To N

    'For each time step

    t1 = CDbl(i - 1) * dt
    delta1 = Black_Scholes_Delta(St, t1, K, T, sig, r, div)
    '        Debug.Print "e = " & e & ", St = " & St & ", delta1 = " & delta1 & ", cv = " & cv
    '        e = Application.WorksheetFunction.NormSInv(Rnd()) 'Standard normal sample
    e = Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + _
        Rnd() + Rnd() + Rnd() + Rnd() - 6#    'Standard normal sample
    '        e = eps(i - 1)
    Stn = St * Exp(nudt + sigsdt * e)
    cv = cv + delta1 * (Stn - St * erddt)    '* exp(T - (t1 + dt)) 'inflation factor *exp can be omitted
    St = Stn

  Next i

  '    Debug.Print "e = " & e & ", St = " & St & ", delta1 = " & delta1 & ", cv = " & cv
  CT = St - K: If CT < 0# Then CT = 0#
  CT = CT + beta1 * cv
  sum_CT = sum_CT + CT
  sum_CT2 = sum_CT + CT * CT
  '    Debug.Print "CT = " & CT & ", CT2 = " & CT * CT

Next j

SD = Sqr(sum_CT * sum_CT / M - sum_CT2) * Exp(-2# * r * T) / (M - 1)    'Standard deviation
```

```
SE = SD / Sqr(M)     'Standard error
European_Call_bMCBS_DCV = sum_CT / M * Exp(-r * T)

End Function

Function European_Call_bMCBS_ADCV( _
           K As Double, _
           T As Double, _
           S As Double, _
           sig As Double, _
           r As Double, _
           div As Double, _
           N As Long, _
           M As Long) As Double

'European Call Option by Monte Carlo with Black-Scholes and Antithetic and Delta-based Control Variates
'Clewlow/Strickland: Implementing Derivatives Models, page 100

Dim dt                        As Double
Dim nudt As Double, sigsdt As Double, erddt As Double
Dim sum_CT As Double, sum_CT2 As Double
Dim e                         As Double
Dim St1 As Double, Stn1       As Double
Dim St2 As Double, Stn2       As Double
Dim CT As Double, CT1 As Double, CT2 As Double
Dim SD As Double, SE          As Double
Dim beta1 As Double, t1       As Double
Dim cv1 As Double, cv2        As Double
Dim delta1 As Double, delta2 As Double
Dim i As Long, j              As Long

'Dim eps As Variant
'eps = Array(0.5391, 0.1692, 0.4583, 0.4164, 0.9223, 0.1597, 0.4011, -0.2382, 1.2152, -0.5506)

'Precompute constants

dt = T / N
nudt = (r - div - 0.5 * sig ^ 2#) * dt
sigsdt = sig * Sqr(dt)
erddt = Exp((r - div) * dt)
beta1 = -1#
'Debug.Print "dt = " & dt & ", nudt = " & nudt & ", sigsdt = " & sigsdt & _
  '", erddt = " & erddt & ", beta1 = " & beta1

sum_CT = 0#
sum_CT2 = 0#

For j = 1 To M

  'For each simulation

  St1 = S
  St2 = S
  cv1 = 0#
  cv2 = 0#

  For i = 1 To N

    'For each time step

    t1 = CDbl(i - 1) * dt
    delta1 = Black_Scholes_Delta(St1, t1, K, T, sig, r, div)
    delta2 = Black_Scholes_Delta(St2, t1, K, T, sig, r, div)
    '        Debug.Print "e = " & e & ", St = " & St & ", delta1 = " & delta1 & ", cv = " & cv
    '        e = Application.WorksheetFunction.NormSInv(Rnd()) 'Standard normal sample
    e = Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + _
        Rnd() + Rnd() + Rnd() + Rnd() - 6#    'Standard normal sample
    '        e = eps(i - 1)
    Stn1 = St1 * Exp(nudt + sigsdt * e)
    Stn2 = St2 * Exp(nudt + sigsdt * -e)
    cv1 = cv1 + delta1 * (Stn1 - St1 * erddt)    '* exp(T - (t1 + dt)) 'inflation factor *exp can be
omitted
    cv2 = cv2 + delta2 * (Stn2 - St2 * erddt)    '* exp(T - (t1 + dt)) 'inflation factor *exp can be
omitted
    St1 = Stn1
    St2 = Stn2

  Next i

  '    Debug.Print "e = " & e & ", St = " & St & ", delta1 = " & delta1 & ", cv = " & cv
  CT1 = St1 - K: If CT1 < 0# Then CT1 = 0#
  CT2 = St2 - K: If CT2 < 0# Then CT2 = 0#
  CT = (CT1 + beta1 * cv1 + CT2 + beta1 * cv2) / 2#
  sum_CT = sum_CT + CT
  sum_CT2 = sum_CT + CT * CT
  '    Debug.Print "CT = " & CT & ", CT2 = " & CT * CT

Next j

SD = Sqr(sum_CT * sum_CT / M - sum_CT2) * Exp(-2# * r * T) / (M - 1)    'Standard deviation
```

```
SE = SD / Sqr(M)     'Standard error
European_Call_bMCBS_ADCV = sum_CT / M * Exp(-r * T)

End Function

Function European_Call_bMCBS_ADGCV( _
           K As Double, _
           T As Double, _
           S As Double, _
           sig As Double, _
           r As Double, _
           div As Double, _
           N As Long, _
           M As Long) As Double

'European Call Option by Monte Carlo with Black-Scholes and Antithetic, Delta-
'and Gamma-based Control Variates
'Clewlow/Strickland: Implementing Derivatives Models, page 102

Dim dt                        As Double
Dim nudt As Double, sigsdt    As Double
Dim erddt As Double, egamma   As Double
Dim sum_CT As Double, sum_CT2 As Double
Dim e                         As Double
Dim St1 As Double, Stn1       As Double
Dim St2 As Double, Stn2       As Double
Dim CT As Double, CT1 As Double, CT2 As Double
Dim SD As Double, SE          As Double
Dim beta1 As Double, beta2 As Double, t1 As Double
Dim cv1 As Double, cv2        As Double
Dim delta1 As Double, delta2 As Double
Dim gamma1 As Double, gamma2 As Double
Dim i As Long, j              As Long

'Dim eps As Variant
'eps = Array(0.5391, 0.1692, 0.4583, 0.4164, 0.9223, 0.1597, 0.4011, -0.2382, 1.2152, -0.5506)

'Precompute constants

dt = T / N
nudt = (r - div - 0.5 * sig ^ 2#) * dt
sigsdt = sig * Sqr(dt)
erddt = Exp((r - div) * dt)
egamma = Exp((2# * (r - div) + sig * sig) * dt) - 2# * erddt + 1#
beta1 = -1#
beta2 = -0.5
'Debug.Print "dt = " & dt & ", nudt = " & nudt & ", sigsdt = " & sigsdt & _
  '", erddt = " & erddt & ", beta1 = " & beta1

sum_CT = 0#
sum_CT2 = 0#

For j = 1 To M

  'For each simulation

  St1 = S
  St2 = S
  cv1 = 0#
  cv2 = 0#

  For i = 1 To N

    'For each time step

    'Compute hedge sensitivities
    t1 = CDbl(i - 1) * dt
    delta1 = Black_Scholes_Delta(St1, t1, K, T, sig, r, div)
    delta2 = Black_Scholes_Delta(St2, t1, K, T, sig, r, div)
    gamma1 = Black_Scholes_Gamma(St1, t1, K, T, sig, r, div)
    gamma2 = Black_Scholes_Gamma(St2, t1, K, T, sig, r, div)

    'Evolve asset prices
    '        Debug.Print "e = " & e & ", St = " & St & ", delta1 = " & delta1 & ", cv = " & cv
    '        e = Application.WorksheetFunction.NormSInv(Rnd()) 'Standard normal sample
    e = Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + _
        Rnd() + Rnd() + Rnd() + Rnd() - 6#    'Standard normal sample
    '        e = eps(i - 1)
    Stn1 = St1 * Exp(nudt + sigsdt * e)
    Stn2 = St2 * Exp(nudt + sigsdt * -e)

    'Accumulate control variates
    cv1 = cv1 + delta1 * (Stn1 - St1 * erddt) + _
          delta2 * (Stn2 - St2 * erddt)
    cv2 = cv2 + gamma1 * ((Stn1 - St1) ^ 2 - St1 ^ 2 * egamma) + _
          gamma2 * ((Stn2 - St2) ^ 2 - St2 ^ 2 * egamma)
    St1 = Stn1
    St2 = Stn2

  Next i
```

```
  '    Debug.Print "e = " & e & ", St = " & St & ", delta1 = " & delta1 & ", cv = " & cv
  CT1 = St1 - K: If CT1 < 0# Then CT1 = 0#
  CT2 = St2 - K: If CT2 < 0# Then CT2 = 0#
  CT = (CT1 + CT2 + beta1 * cv1 + beta2 * cv2) / 2#
  sum_CT = sum_CT + CT
  sum_CT2 = sum_CT + CT * CT
  '    Debug.Print "CT = " & CT & ", CT2 = " & CT * CT

Next j

SD = Sqr(sum_CT * sum_CT / M - sum_CT2) * Exp(-2# * r * T) / (M - 1)    'Standard deviation
SE = SD / Sqr(M)    'Standard error
European_Call_bMCBS_ADGCV = sum_CT / M * Exp(-r * T)

End Function

Function European_Spread_bMCBS( _
          K As Double, _
          T As Double, _
          S1 As Double, _
          S2 As Double, _
          sig1 As Double, _
          sig2 As Double, _
          div1 As Double, _
          div2 As Double, _
          rho As Double, _
          r As Double, _
          N As Long, _
          M As Long) As Double

'European Spread Option by Monte Carlo with Black-Scholes
'Clewlow/Strickland: Implementing Derivatives Models, page 110

Dim dt                      As Double
Dim nu1dt As Double, nu2dt  As Double
Dim sig1sdt As Double, sig2sdt As Double
Dim sum_CT As Double, sum_CT2 As Double
Dim e1 As Double, e2        As Double
Dim z1 As Double, z2        As Double
Dim St1 As Double, St2      As Double
Dim CT                      As Double
Dim SD As Double, SE        As Double
Dim srho                    As Double
Dim i As Long, j            As Long

'Precompute constants

dt = T / N
nu1dt = (r - div1 - 0.5 * sig1 ^ 2#) * dt
nu2dt = (r - div2 - 0.5 * sig2 ^ 2#) * dt
sig1sdt = sig1 * Sqr(dt)
sig2sdt = sig2 * Sqr(dt)
srho = Sqr(1# - rho ^ 2#)

sum_CT = 0#
sum_CT2 = 0#

For j = 1 To M

  'For each simulation

  St1 = S1
  St2 = S2

  For i = 1 To N

    'For each time step

    e1 = Application.WorksheetFunction.NormSInv(Rnd())    'Standard normal sample
    e2 = Application.WorksheetFunction.NormSInv(Rnd())    'Standard normal sample
    '        e1 = Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + _
             '            Rnd() + Rnd() + Rnd() + Rnd() - 6# 'Standard normal sample
    '        e2 = Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + _
             '            Rnd() + Rnd() + Rnd() + Rnd() - 6# 'Standard normal sample
    z1 = e1
    z2 = rho * e1 + srho * e2
    St1 = St1 * Exp(nu1dt + sig1sdt * z1)
    St2 = St2 * Exp(nu2dt + sig2sdt * z2)

  Next i

  CT = St1 - St2 - K: If CT < 0# Then CT = 0#
  sum_CT = sum_CT + CT
  sum_CT2 = sum_CT + CT * CT

Next j

SD = Sqr(sum_CT * sum_CT / M - sum_CT2) * Exp(-2# * r * T) / (M - 1)    'Standard deviation
SE = SD / Sqr(M)    'Standard error
```

```
European_Spread_bMCBS = sum_CT / M * Exp(-r * T)

End Function

Function European_Spread_bMCBS_SV( _
           K As Double, _
           T As Double, _
           S1 As Double, _
           S2 As Double, _
           sig1 As Double, _
           sig2 As Double, _
           div1 As Double, _
           div2 As Double, _
           alpha1 As Double, _
           alpha2 As Double, _
           Vbar1 As Double, _
           Vbar2 As Double, _
           xi1 As Double, _
           xi2 As Double, _
           rhoz As Variant, _
           r As Double, _
           N As Long, _
           M As Long) As Double

'European Spread Option by Monte Carlo with Black-Scholes and Stochastic Volatilities
'Clewlow/Strickland: Implementing Derivatives Models, page 113

Dim dt                       As Double
Dim xi1sdt As Double, xi2sdt As Double
Dim alpha1dt As Double, alpha2dt As Double
Dim sum_CT As Double, sum_CT2 As Double
Dim LnS1 As Double, LnS2     As Double
Dim LnSt1 As Double, LnSt2   As Double
Dim v1 As Double, v2         As Double
Dim vt1 As Double, vt2       As Double
Dim z1 As Double, z2         As Double
Dim St1 As Double, St2       As Double
Dim CT                       As Double
Dim SD As Double, SE         As Double
Dim sdt                      As Double
Dim i As Long, j             As Long
Dim z                        As Variant

'Precompute constants

dt = T / N
alpha1dt = alpha1 * dt
alpha2dt = alpha2 * dt
xi1sdt = xi1 * Sqr(dt)
xi2sdt = xi2 * Sqr(dt)
sdt = Sqr(dt)
LnS1 = Log(S1)
LnS2 = Log(S2)
v1 = sig1 * sig1
v2 = sig2 * sig2

sum_CT = 0#
sum_CT2 = 0#

For j = 1 To M

  'For each simulation

  LnSt1 = LnS1
  LnSt2 = LnS2
  vt1 = v1
  vt2 = v2

  'Generate correlated normals
  z = RandCorr(N, rhoz)

  For i = 1 To N    'For each time step

    'Simulate variances first
    vt1 = vt1 + alpha1dt * (Vbar1 - vt1) + xi1sdt * Sqr(vt1) * z(i, 3)
    vt2 = vt2 + alpha2dt * (Vbar2 - vt2) + xi2sdt * Sqr(vt2) * z(i, 4)

    'Simulate asset prices
    LnSt1 = LnSt1 + (r - div1 - vt1 / 2#) * dt + Sqr(vt1) * sdt * z(i, 1)
    LnSt2 = LnSt2 + (r - div2 - vt2 / 2#) * dt + Sqr(vt2) * sdt * z(i, 2)

  Next i

  St1 = Exp(LnSt1)
  St2 = Exp(LnSt2)

  CT = St1 - St2 - K: If CT < 0# Then CT = 0#
  sum_CT = sum_CT + CT
  sum_CT2 = sum_CT + CT * CT
```

```
Next j

SD = Sqr(sum_CT * sum_CT / M - sum_CT2) * Exp(-2# * r * T) / (M - 1)    'Standard deviation
SE = SD / Sqr(M)    'Standard error
European_Spread_bMCBS_SV = sum_CT / M * Exp(-r * T)

End Function

Function European_Down_and_Out_Call_bMCBS( _
          K As Double, _
          T As Double, _
          S As Double, _
          sig As Double, _
          r As Double, _
          div As Double, _
          H As Long, _
          N As Long, _
          M As Long) As Double

'European Down and Out Call Option by Monte Carlo with Black-Scholes
'Clewlow/Strickland: Implementing Derivatives Models, page 116

Dim dt                      As Double
Dim nudt As Double, sigsdt  As Double
Dim lnS As Double, lnSt     As Double
Dim sum_CT As Double, sum_CT2 As Double
Dim e                       As Double
Dim St As Double, CT        As Double
Dim SD As Double, SE        As Double
Dim i As Long, j            As Long
Dim bBarrierCrossed         As Boolean

#If MODULE_TEST Then
  e = Rnd(-1)
  Randomize 1    'maybe even additional parameter to function
#End If

'Precompute constants
dt = T / N
nudt = (r - div - sig * sig / 2#) * dt
sigsdt = sig * Sqr(dt)

sum_CT = 0#
sum_CT2 = 0#

For j = 1 To M

  'For each simulation
  St = S
  bBarrierCrossed = False

  For i = 1 To N

    'For each time step
    '   e = Application.WorksheetFunction.NormSInv(Rnd()) 'Standard normal sample
    e = Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + _
        Rnd() + Rnd() + Rnd() + Rnd() - 6#    'Standard normal sample
    St = St * Exp(nudt + sigsdt * e)    'Evolve the stock price
    If St <= H Then
      bBarrierCrossed = True
      Exit For
    End If

  Next i

  If bBarrierCrossed Then
    CT = 0#
  Else
    CT = St - K
    If CT < 0# Then CT = 0#
  End If

  sum_CT = sum_CT + CT
  sum_CT2 = sum_CT + CT * CT

Next j

SD = Sqr(Abs(sum_CT * sum_CT / M - sum_CT2)) * Exp(-2# * r * T) / (M - 1)    'Standard deviation
SE = SD / Sqr(M)    'Standard error
European_Down_and_Out_Call_bMCBS = sum_CT / M * Exp(-r * T)

End Function

Function European_Arithmetic_Asian_Call_bMCwGAOCV( _
          K As Double, _
          T As Variant, _
          S As Double, _
          sig As Double, _
          r As Double, _
          div As Double, _
```

```
            N As Long, _
            M As Long) As Double

'European Arithmetic Asian Call Option by Monte Carlo
'with a Geometric Asian Call Option Control Variate
'Clewlow/Strickland: Implementing Derivatives Models, page 120

Dim sumSt As Double, productSt As Double
Dim sum_CT As Double, sum_CT2 As Double
Dim e                         As Double
Dim SD As Double, SE          As Double
Dim St As Double, CT          As Double
Dim A As Double, G            As Double
Dim i As Long, j              As Long

ReDim nudt(0 To N) As Double
ReDim sigsdt(0 To N) As Double

'Precompute constants

For i = 1 To N
  'For each fixing
  nudt(i) = (r - div - sig * sig / 2#) * (T(i) - T(i - 1))
  sigsdt(i) = sig * Sqr(T(i) - T(i - 1))
Next i

sum_CT = 0#
sum_CT2 = 0#

For j = 1 To M

  'For each simulation
  St = S
  sumSt = 0#
  productSt = 1#

  For i = 1 To N

    'For each fixing
    '        e = Application.WorksheetFunction.NormSInv(Rnd()) 'Standard normal sample
    e = Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + Rnd() + _
        Rnd() + Rnd() + Rnd() + Rnd() - 6#    'Standard normal sample
    St = St * Exp(nudt(i) + sigsdt(i) * e)    'Evolve the stock price
    sumSt = sumSt + St
    productSt = productSt * St

  Next i

  A = sumSt / N
  G = productSt ^ (1# / N)
  CT = A - K: If CT < 0# Then CT = 0#
  If G > K Then CT = CT - G + K
  sum_CT = sum_CT + CT
  sum_CT2 = sum_CT + CT * CT

Next j

SD = Sqr(sum_CT * sum_CT / M - sum_CT2) * Exp(-2# * r * T) / (M - 1)    'Standard deviation
SE = SD / Sqr(M)    'Standard error
European_Arithmetic_Asian_Call_bMCwGAOCV = sum_CT / M * Exp(-r * T)
'+ Geometric_Asian_Call(K, T, S, sig, r, div, N)

End Function

Function Black_Scholes_Delta( _
            St As Double, _
            t1 As Double, _
            K As Double, _
            T As Double, _
            sig As Double, _
            r As Double, _
            div As Double) As Double

With Application.WorksheetFunction
  Black_Scholes_Delta = Exp(-div * (T - t1)) * .NormSDist((Log(St / K) + _
                                                  (r - div + sig * sig / 2#) * (T - t1)) / (sig *
Sqr(T - t1)))
End With

End Function

Function Black_Scholes_Gamma( _
            St As Double, _
            t1 As Double, _
            K As Double, _
            T As Double, _
            sig As Double, _
            r As Double, _
            div As Double) As Double
```

```
With Application.WorksheetFunction
  Black_Scholes_Gamma = Exp(-div * (T - t1)) * _
                        .NormDist((Log(St / K) + (r - div + sig * sig / 2#) * _
                                   (T - t1)) / (sig * Sqr(T - t1)), _
                                  0, 1, False) / (St * sig * Sqr(T - t1))
End With

End Function

Function RandCorr(N As Long, vVarCovar As Variant) As Variant
'Returns Ubound(vVarCovar,1) correlated random number vectors of length n.
'vVarCovar is a square matrix containing the variance/covariance matrix.
'Please notice that you will only get a "proxy" correlation, not an exact one.
'Reverse("moc.LiborPlus.www") V0.2 PB 06-Nov-2009
Dim vA As Variant, d         As Double
Dim i As Long, j As Long, K As Long, M As Long

With Application.WorksheetFunction
  vA = .Transpose(.Transpose(vVarCovar))
  M = UBound(vA, 1)
  If M <> UBound(vA, 2) Then
    RandCorr = CVErr(xlErrRef)
    Exit Function
  End If

  ReDim Db(1 To M, 1 To M) As Double
  For j = 1 To M
    d = 0#
    For K = 1 To j - 1
      d = d + Db(j, K) * Db(j, K)
    Next K
    Db(j, j) = vA(j, j) - d
    If Db(j, j) <= 0# Then
      RandCorr = CVErr(xlErrNum)
      Exit Function
    End If
    Db(j, j) = Sqr(Db(j, j))

    For i = j + 1 To M
      d = 0#
      For K = 1 To j - 1
        d = d + Db(i, K) * Db(j, K)
      Next K
      Db(i, j) = (vA(i, j) - d) / Db(j, j)
    Next i
  Next j

  ReDim vR(1 To N, 1 To M) As Variant
  For i = 1 To N
    For j = 1 To M
      vR(i, j) = .NormSInv(Rnd())
    Next j
  Next i
  vR = .MMult(vR, Db)
  RandCorr = vR
End With
End Function
```

Generate all Permutations of an Array – *Quickperm*

If you need to generate all permutations of an array I suggest to use the algorithm Quickperm. It is one of the most efficient permutation algorithms. It is based on swaps of single array elements and inspired by Heap sort.

The version presented here is the CountDown variant. For the Excel / VBA implementation all indices were increased by one so that arrays start with index 1.

<u>Example</u>

Input:

	A	B	C	D
1	1	2	3	4
2	A	B	C	

Output:

	A	B	C	D
1	1	2	3	4
2	2	1	3	4
3	3	1	2	4
4	1	3	2	4
5	2	3	1	4
6	3	2	1	4
7	3	2	4	1
8	2	3	4	1
9	4	3	2	1
10	3	4	2	1
11	2	4	3	1
12	4	2	3	1
13	4	1	3	2
14	1	4	3	2
15	3	4	1	2
16	4	3	1	2
17	1	3	4	2
18	3	1	4	2
19	2	1	4	3
20	1	2	4	3
21	4	2	1	3
22	2	4	1	3
23	1	4	2	3
24	4	1	2	3
25	A	B	C	
26	B	A	C	
27	C	A	B	
28	A	C	B	
29	B	C	A	
30	C	B	A	

Input | Output

Links

Quickperm:

Quickperm – Die originale Website: https://www.quickperm.org/

Baeldung – Generate All Permutations of an Array. Ein hilfreiches Tutorial (engl.) welches in Quickperm mündet: https://www.baeldung.com/cs/array-generate-all-permutations

Permutationen mit Nebenbedingungen:

Rosetta Code – Permutations with some identical elements:
https://rosettacode.org/wiki/Permutations_with_some_identical_elements

Computer Science – Enumerating all partial permutations of given length in lexicographic order:
https://cs.stackexchange.com/questions/133662/enumerating-all-partial-permutations-of-given-length-in-lexicographic-order

Quickperm Program Code

```
Public r As Long 'Output row

Sub QuickPerm(a As Variant)
'Generates all permutations of array a.
'Quickperm is one of the most efficient permutation algorithms.
'It is based on swapping and inspired by Heap sort.
'Countdown variant, indexes increased by 1.
'Algorithm originally from https://www.quickperm.org, migrated to VBA.
'See also: https://www.baeldung.com/cs/array-generate-all-permutations
'Version 0.1 28-Jun-2024
Dim i As Long, idx As Long, j As Long, n As Long, v As Variant
With Application.WorksheetFunction
a = .Transpose(.Transpose(a)) 'Assuming horizontal range
Call VisitPerm(a)
n = UBound(a)
ReDim p(1 To n + 1) As Long
For i = 1 To n + 1: p(i) = i: Next i 'Initialize p()
idx = 2
Do While idx < n + 1
  p(idx) = p(idx) - 1
  If idx Mod 2 = 0 Then
    j = p(idx)
  Else
    j = 1
  End If
  v = a(j): a(j) = a(idx): a(idx) = v 'Swap a(j) and a(idx)
  Call VisitPerm(a)
  idx = 2
  Do While p(idx) = 1
    p(idx) = idx
    idx = idx + 1
  Loop
Loop
End With
End Sub

Sub VisitPerm(a As Variant)
'Print current permutation in immediate window and on sheet Output.
'You can analyze the permutation or do other things as well.
Dim i As Long
For i = 1 To UBound(a)
  Debug.Print a(i);
  wsO.Cells(r, i) = a(i)
Next i
Debug.Print
r = r + 1
End Sub
```

Weight Calculation

Suppose you have three different products, each with 3 variations with different weights, which also exist in varying quantities. The product PR exists with the weights of 404 g (there are 7 pieces), 401 g (four times), and 398 g (five times), and there are corresponding other products BB and BO:

	A	B	C	D	E	F
1	Count			Weight		
2	PR	BB	BO	PR	BB	BO
3	7	8	6	404	130	895
4	4	3	1	401	128	894
5	5	4	2	398	125	891

Now, three sets of three products (one from each of the three products) with identical total weights should be determined, and the remaining quantity of products should be output:

	A	B	C	D	E	F	G	H	I	J	K
1	#	Total	PR	BB	BO	PR count	BB count	BO count	PR weight	BB weight	BO weight
2	1	1429	404	130	895	4	5	3	404	130	895
3			404	130	895	4	3	1	401	128	894
4			404	130	895	5	4	2	398	125	891
5	2	1427	404	128	895	4	8	3	404	130	895
6			404	128	895	4	0	1	401	128	894
7			404	128	895	5	4	2	398	125	891
8	3	1423	404	128	891	4	8	6	404	130	895
9			404	128	891	4	1	0	401	128	894
10			404	125	894	5	3	0	398	125	891
98	33	1418	398	125	895	7	8	3	404	130	895
99			398	125	895	4	3	1	401	128	894
100			398	125	895	2	1	2	398	125	891

There are many alternatives for determining all possible withdrawals. An attachment mentions a very simple, but quite labor-intensive option (see the first program code), which naively goes through all possible combinations. The second program code shows a Monte Carlo simulation that uses the function UniqRandInt and, with 500,000 iterations, most likely (but not certainly) determines all possibilities. A third option would be to use a permutation function that only examines all possible 84 * 84 * 20 = 141,120 permutations.

One (of 12 different) combined withdrawal with the smallest remaining weight sum is:

	A	B	C	D	E	F	G	H	I	J	K
1	Start:										
2	Count			Weight							
3	PR	BB	BO	PR	BB	BO					
4	7	8	6	404	130	895					
5	4	3	1	401	128	894					
6	5	4	2	398	125	891					
7											
8	Drawing:										
9	#	Total	PR	BB	BO	PR count	BB count	BO count	PR weight	BB weight	BO weight
10	1	1429	404	130	895	4	5	3	404	130	895
11			404	130	895	4	3	1	401	128	894
12			404	130	895	5	4	2	398	125	891
13	1	1429	404	130	895	4	5	3	404	130	895
14			404	130	895	4	3	1	401	128	894
15			404	130	895	5	4	2	398	125	891
16	14	1420	404	125	891	6	8	6	404	130	895
17			401	128	891	2	2	0	401	128	894
18			401	125	894	5	2	0	398	125	891
19											
20	Remaining:										
21	Count										
22	PR	BB	BO								
23	0	2	0								
24	2	2	0								
25	5	2	0								

All 12 different withdrawal combinations – the numbers refer to the above-mentioned output variant:

	M	N	O	P	Q
8	All 12 draw combinations:				
9		Sum	First Draw	Second Draw	Third Draw
10	1	12834	1	1	14
11	2	12834	1	1	16
12	3	12834	1	1	21
13	4	12834	1	1	24
14	5	12834	1	2	23
15	6	12834	1	3	19
16	7	12834	1	5	7
17	8	12834	1	5	13
18	9	12834	1	5	20
19	10	12834	1	6	19
20	11	12834	1	9	12
21	12	12834	2	5	19

Reasonable Extensions and Generalisations

The quickly developed first solution above could be expanded and generalized according to the following approaches:

https://stackoverflow.com/questions/54669041/vba-write-all-permutations-of-numbers-to-an-array (auch hier: https://www.vitoshacademy.com/vba-nested-loops-with-recursion/)

https://www.physicsforums.com/threads/loop-with-variable-nesting-depth-and-variable-count-at-each-level.1046986/

https://www.codeproject.com/Tips/759707/Generating-dynamically-nested-loops

https://stackoverflow.com/questions/1737289/dynamic-nested-loops-level

AllFirstDraws Program Code

```
'Calculates 3 * 3 - tuples of same total weights.
'(C) (P) by Bernd Plumhoff 26-Jun-2024 PB V0.4

Sub AllFirstDraws()
Dim i As Long, j As Long, k As Long
Dim i2 As Long, j2 As Long, k2 As Long
Dim i3 As Long, j3 As Long, k3 As Long
Dim m As Long, n As Long, t As Long, v As Long

Dim oGetRidofDupes           As Object

Dim vCount                   As Variant
Dim vWeight                  As Variant

Dim state                    As SystemState

With Application.WorksheetFunction
  Set state = New SystemState
  wsI.Cells.EntireColumn.AutoFit
  wsO.Cells.ClearContents
  Set oGetRidofDupes = CreateObject("Scripting.Dictionary")
  i = 1
  Do While wsI.Cells(2, i) <> ""
    i = i + 1
  Loop
  n = (i - 1) \ 2
  vCount = .Transpose(Range(wsI.Cells(3, 1), wsI.Cells(3, n).End(xlDown)))
  vWeight = .Transpose(Range(wsI.Cells(3, n + 1), wsI.Cells(3, 2 * n).End(xlDown)))
  For i = 1 To n
    k = 0
    For j = 1 To UBound(vCount, 2)
      k = k + vCount(j, i)
    Next j
    If k < n Then
      Call MsgBox("Not enough items in column " & i, vbOKOnly, "Error")
      Exit Sub
    End If
  Next i
  m = j - 1
  'Debug.Print "n = " & n, "m = " & m
  'Now we know the dimensions
  ReDim sItem(1 To n) As String
  wsO.Cells(1, 1) = "#"
  wsO.Cells(1, 2) = "Total"
  For i = 1 To n
    sItem(i) = wsI.Cells(2, i)
    wsO.Cells(1, i + 2) = sItem(i)
    wsO.Cells(1, n + 2 + i) = sItem(i) & " count"
    wsO.Cells(1, 2 * n + 2 + i) = sItem(i) & " weight"
  Next i

  ReDim lPermutWeight(1 To n, 1 To n * m) As Long
  ReDim lPermutIdx(1 To n) As Long
  ReDim lPermutSubGroupIdx(1 To n, 1 To n * m) As Long

  For i = 1 To n
```

```
  t = 0
  For j = 1 To m
    For k = 1 To .Min(n, vCount(i, j))
      t = t + 1
      lPermutWeight(i, t) = vWeight(i, j)
      lPermutSubGroupIdx(i, t) = j
    Next k
  Next j
  lPermutIdx(i) = t
Next i

v = 2
For i = 1 To lPermutIdx(1)
  For j = 1 To lPermutIdx(1)
    If j <> i Then
      For k = 1 To lPermutIdx(1)
        If k <> j And k <> i Then
          For i2 = 1 To lPermutIdx(2)
            For j2 = 1 To lPermutIdx(2)
              If j2 <> i2 Then
                For k2 = 1 To lPermutIdx(2)
                  If k2 <> j2 And k2 <> i2 Then
                    For i3 = 1 To lPermutIdx(3)
                      For j3 = 1 To lPermutIdx(3)
                        If j3 <> i3 Then
                          For k3 = 1 To lPermutIdx(3)
                            If k3 <> j3 And k3 <> i3 Then
                              'Debug.Print lPermutWeight(1, i) & " + " & lPermutWeight(2, i2) & _
                               '" + " & lPermutWeight(3, i3) & " ?= " & lPermutWeight(1, j) & _
                               '" + " & lPermutWeight(2, j2) & " + " & lPermutWeight(3, j3) & _
                               '" And " & lPermutWeight(1, i) & " + " & lPermutWeight(2, i2) & _
                               '" + " & lPermutWeight(3, i3) & " ?= " & lPermutWeight(1, k) & _
                               '" + " & lPermutWeight(2, k2) & " + " & lPermutWeight(3, k3)
                              If lPermutWeight(1, i) + lPermutWeight(2, i2) + lPermutWeight(3, i3) = _
                                 lPermutWeight(1, j) + lPermutWeight(2, j2) + lPermutWeight(3, j3) And _
                                 lPermutWeight(1, i) + lPermutWeight(2, i2) + lPermutWeight(3, i3) = _
                                 lPermutWeight(1, k) + lPermutWeight(2, k2) + lPermutWeight(3, k3) Then
                               If Not oGetRidofDupes.exists(lPermutWeight(1, i) & "|" & _
                                                            lPermutWeight(2, i2) & "|" & _
                                                            lPermutWeight(3, i3) & "|" & _
                                                            lPermutWeight(1, j) & "|" & _
                                                            lPermutWeight(2, j2) & "|" & _
                                                            lPermutWeight(3, j3) & "|" & _
                                                            lPermutWeight(1, k) & "|" & _
                                                            lPermutWeight(2, k2) & "|" & _
                                                            lPermutWeight(3, k3)) Then
                                 oGetRidofDupes(lPermutWeight(1, i) & "|" & lPermutWeight(2, i2) & _
                                                "|" & lPermutWeight(3, i3) & "|" & _
                                                lPermutWeight(1, j) & "|" & _
                                                lPermutWeight(2, j2) & "|" & _
                                                lPermutWeight(3, j3) & "|" & _
                                                lPermutWeight(1, k) & "|" & _
                                                lPermutWeight(2, k2) & "|" & _
                                                lPermutWeight(3, k3)) = 1
                                 oGetRidofDupes(lPermutWeight(1, i) & "|" & lPermutWeight(2, i2) & _
                                                "|" & lPermutWeight(3, i3) & "|" & _
                                                lPermutWeight(1, k) & "|" & _
                                                lPermutWeight(2, k2) & "|" & _
                                                lPermutWeight(3, k3) & "|" & _
                                                lPermutWeight(1, j) & "|" & _
                                                lPermutWeight(2, j2) & "|" & _
                                                lPermutWeight(3, j3)) = 1
                                 oGetRidofDupes(lPermutWeight(1, j) & "|" & _
                                                lPermutWeight(2, j2) & "|" & _
                                                lPermutWeight(3, j3) & "|" & _
                                                lPermutWeight(1, i) & "|" & _
                                                lPermutWeight(2, i2) & "|" & _
                                                lPermutWeight(3, i3) & "|" & _
                                                lPermutWeight(1, k) & "|" & _
                                                lPermutWeight(2, k2) & "|" & _
                                                lPermutWeight(3, k3)) = 1
                                 oGetRidofDupes(lPermutWeight(1, j) & "|" & _
                                                lPermutWeight(2, j2) & "|" & _
                                                lPermutWeight(3, j3) & "|" & _
                                                lPermutWeight(1, k) & "|" & _
                                                lPermutWeight(2, k2) & "|" & _
                                                lPermutWeight(3, k3) & "|" & _
                                                lPermutWeight(1, i) & "|" & _
                                                lPermutWeight(2, i2) & "|" & _
                                                lPermutWeight(3, i3)) = 1
                                 oGetRidofDupes(lPermutWeight(1, k) & "|" & _
                                                lPermutWeight(2, k2) & "|" & _
                                                lPermutWeight(3, k3) & "|" & _
                                                lPermutWeight(1, i) & "|" & _
                                                lPermutWeight(2, i2) & "|" & _
                                                lPermutWeight(3, i3) & "|" & _
                                                lPermutWeight(1, j) & "|" & _
                                                lPermutWeight(2, j2) & "|" & _
                                                lPermutWeight(3, j3)) = 1
                                 oGetRidofDupes(lPermutWeight(1, k) & "|" & _
```

```
                                    lPermutWeight(2, k2) & "|" & _
                                    lPermutWeight(3, k3) & "|" & _
                                    lPermutWeight(1, j) & "|" & _
                                    lPermutWeight(2, j2) & "|" & _
                                    lPermutWeight(3, j3) & "|" & _
                                    lPermutWeight(1, i) & "|" & _
                                    lPermutWeight(2, i2) & "|" & _
                                    lPermutWeight(3, i3)) = 1
                                wsO.Cells(v, 1) = (v + 1) \ n
                                wsO.Cells(v, 2) = lPermutWeight(1, i) + lPermutWeight(2, i2) + _
                                                  lPermutWeight(3, i3)
                                wsO.Cells(v, 3) = lPermutWeight(1, i)
                                wsO.Cells(v, 4) = lPermutWeight(2, i2)
                                wsO.Cells(v, 5) = lPermutWeight(3, i3)
                                wsO.Cells(v + 1, 3) = lPermutWeight(1, j)
                                wsO.Cells(v + 1, 4) = lPermutWeight(2, j2)
                                wsO.Cells(v + 1, 5) = lPermutWeight(3, j3)
                                wsO.Cells(v + 2, 3) = lPermutWeight(1, k)
                                wsO.Cells(v + 2, 4) = lPermutWeight(2, k2)
                                wsO.Cells(v + 2, 5) = lPermutWeight(3, k3)
                                wsO.Cells(v, 6) = vCount(1, 1) - _
                                                  IIf(lPermutSubGroupIdx(1, i) = 1, 1, 0) - _
                                                  IIf(lPermutSubGroupIdx(1, j) = 1, 1, 0) - _
                                                  IIf(lPermutSubGroupIdx(1, k) = 1, 1, 0)
                                wsO.Cells(v, 7) = vCount(2, 1) - _
                                                  IIf(lPermutSubGroupIdx(2, i2) = 1, 1, 0) - _
                                                  IIf(lPermutSubGroupIdx(2, j2) = 1, 1, 0) - _
                                                  IIf(lPermutSubGroupIdx(2, k2) = 1, 1, 0)
                                wsO.Cells(v, 8) = vCount(3, 1) - _
                                                  IIf(lPermutSubGroupIdx(3, i3) = 1, 1, 0) - _
                                                  IIf(lPermutSubGroupIdx(3, j3) = 1, 1, 0) - _
                                                  IIf(lPermutSubGroupIdx(3, k3) = 1, 1, 0)
                                wsO.Cells(v + 1, 6) = vCount(1, 2) - _
                                                      IIf(lPermutSubGroupIdx(1, i) = 2, 1, 0) - _
                                                      IIf(lPermutSubGroupIdx(1, j) = 2, 1, 0) - _
                                                      IIf(lPermutSubGroupIdx(1, k) = 2, 1, 0)
                                wsO.Cells(v + 1, 7) = vCount(2, 2) - _
                                                      IIf(lPermutSubGroupIdx(2, i2) = 2, 1, 0) - _
                                                      IIf(lPermutSubGroupIdx(2, j2) = 2, 1, 0) - _
                                                      IIf(lPermutSubGroupIdx(2, k2) = 2, 1, 0)
                                wsO.Cells(v + 1, 8) = vCount(3, 2) - _
                                                      IIf(lPermutSubGroupIdx(3, i3) = 2, 1, 0) - _
                                                      IIf(lPermutSubGroupIdx(3, j3) = 2, 1, 0) - _
                                                      IIf(lPermutSubGroupIdx(3, k3) = 2, 1, 0)
                                wsO.Cells(v + 2, 6) = vCount(1, 3) - _
                                                      IIf(lPermutSubGroupIdx(1, i) = 3, 1, 0) - _
                                                      IIf(lPermutSubGroupIdx(1, j) = 3, 1, 0) - _
                                                      IIf(lPermutSubGroupIdx(1, k) = 3, 1, 0)
                                wsO.Cells(v + 2, 7) = vCount(2, 3) - _
                                                      IIf(lPermutSubGroupIdx(2, i2) = 3, 1, 0) - _
                                                      IIf(lPermutSubGroupIdx(2, j2) = 3, 1, 0) - _
                                                      IIf(lPermutSubGroupIdx(2, k2) = 3, 1, 0)
                                wsO.Cells(v + 2, 8) = vCount(3, 3) - _
                                                      IIf(lPermutSubGroupIdx(3, i3) = 3, 1, 0) - _
                                                      IIf(lPermutSubGroupIdx(3, j3) = 3, 1, 0) - _
                                                      IIf(lPermutSubGroupIdx(3, k3) = 3, 1, 0)
                                wsO.Cells(v, 9) = vWeight(1, 1)
                                wsO.Cells(v, 10) = vWeight(2, 1)
                                wsO.Cells(v, 11) = vWeight(3, 1)
                                wsO.Cells(v + 1, 9) = vWeight(1, 2)
                                wsO.Cells(v + 1, 10) = vWeight(2, 2)
                                wsO.Cells(v + 1, 11) = vWeight(3, 2)
                                wsO.Cells(v + 2, 9) = vWeight(1, 3)
                                wsO.Cells(v + 2, 10) = vWeight(2, 3)
                                wsO.Cells(v + 2, 11) = vWeight(3, 3)
                                v = v + 3
                            End If
                        End If
                    End If
                Next k3
            End If
        Next j3
    Next i3
End If
Next k2
End If
Next j2
Next i2
End If
Next k
End If
Next j
Next i
wsO.Cells.EntireColumn.AutoFit
End With
End Sub
```

AllFirstDraws Monte Carlo Program Code

```
'Calculates 3 * 3 - tuples of same total weights.
'(C) (P) by Bernd Plumhoff 07-Jul-2024 PB V0.41

Sub AllFirstDraws()
Dim i                        As Long
Dim j                        As Long
Dim k                        As Long
Dim i2                       As Long
Dim j2                       As Long
Dim k2                       As Long
Dim i3                       As Long
Dim j3                       As Long
Dim k3                       As Long
Dim m                        As Long
Dim n                        As Long
Dim t                        As Long
Dim u                        As Long
Dim v                        As Long
Dim x                        As Long

Dim v1                       As Variant
Dim v2                       As Variant
Dim v3                       As Variant

Dim oGetRidofDupes           As Object

Dim vCount                   As Variant
Dim vWeight                  As Variant

Dim state                    As SystemState

With Application.WorksheetFunction
  Set state = New SystemState
  wsI.Cells.EntireColumn.AutoFit
  wsO.Cells.ClearContents
  Set oGetRidofDupes = CreateObject("Scripting.Dictionary")
  i = 1
  Do While wsI.Cells(2, i) <> ""
    i = i + 1
  Loop
  n = (i - 1) \ 2
  vCount = .Transpose(Range(wsI.Cells(3, 1), wsI.Cells(3, n).End(xlDown)))
  vWeight = .Transpose(Range(wsI.Cells(3, n + 1), wsI.Cells(3, 2 * n).End(xlDown)))
  For i = 1 To n
    k = 0
    For j = 1 To UBound(vCount, 2)
      k = k + vCount(j, i)
    Next j
    If k < n Then
      Call MsgBox("Not enough items in column " & i, vbOKOnly, "Error")
      Exit Sub
    End If
  Next i
  m = j - 1
  'Debug.Print "n = " & n, "m = " & m
  'Now we know the dimensions
  ReDim sItem(1 To n) As String
  wsO.Cells(1, 1) = "#"
  wsO.Cells(1, 2) = "Total"
  For i = 1 To n
    sItem(i) = wsI.Cells(2, i)
    wsO.Cells(1, i + 2) = sItem(i)
    wsO.Cells(1, n + 2 + i) = sItem(i) & " count"
    wsO.Cells(1, 2 * n + 2 + i) = sItem(i) & " weight"
  Next i

  ReDim lPermutWeight(1 To n, 1 To n * m) As Long
  ReDim lPermutIdx(1 To n) As Long
  ReDim lPermutSubGroupIdx(1 To n, 1 To n * m) As Long

  For i = 1 To n
    t = 0
    For j = 1 To m
      For k = 1 To .Min(n, vCount(i, j))
        t = t + 1
        lPermutWeight(i, t) = vWeight(i, j)
        lPermutSubGroupIdx(i, t) = j
      Next k
    Next j
    lPermutIdx(i) = t
  Next i

  v1 = Rnd(-1)     'With this initialization we only need 195026 iterations for 33 solutions
  Randomize (1)

  v = 2
```

```
For u = 1 To 500000
  v1 = VBUniqRandInt(3, lPermutIdx(1))
  v2 = VBUniqRandInt(3, lPermutIdx(2))
  v3 = VBUniqRandInt(3, lPermutIdx(3))
  i = v1(1)
  j = v1(2)
  k = v1(3)
  i2 = v2(1)
  j2 = v2(2)
  k2 = v2(3)
  i3 = v3(1)
  j3 = v3(2)
  k3 = v3(3)
  'Debug.Print lPermutWeight(1, i) & " + " & lPermutWeight(2, i2) & " + " & _
   'lPermutWeight(3, i3) & " ?= " & lPermutWeight(1, j) & " + " & lPermutWeight(2, j2) & _
   '" + " & lPermutWeight(3, j3) & " And " & lPermutWeight(1, i) & " + " & _
   'lPermutWeight(2, i2) & " + " & lPermutWeight(3, i3) & " ?= " & lPermutWeight(1, k) & _
   '" + " & lPermutWeight(2, k2) & " + " & lPermutWeight(3, k3)
  If lPermutWeight(1, i) + lPermutWeight(2, i2) + lPermutWeight(3, i3) = _
     lPermutWeight(1, j) + lPermutWeight(2, j2) + lPermutWeight(3, j3) And _
     lPermutWeight(1, i) + lPermutWeight(2, i2) + lPermutWeight(3, i3) = _
     lPermutWeight(1, k) + lPermutWeight(2, k2) + lPermutWeight(3, k3) Then
    If Not oGetRidofDupes.exists(lPermutWeight(1, i) & "|" & lPermutWeight(2, i2) & "|" & _
                                 lPermutWeight(3, i3) & "|" & lPermutWeight(1, j) & "|" & _
                                 lPermutWeight(2, j2) & "|" & lPermutWeight(3, j3) & "|" & _
                                 lPermutWeight(1, k) & "|" & lPermutWeight(2, k2) & "|" & _
                                 lPermutWeight(3, k3)) Then
      oGetRidofDupes(lPermutWeight(1, i) & "|" & lPermutWeight(2, i2) & "|" & _
                     lPermutWeight(3, i3) & "|" & lPermutWeight(1, j) & "|" & _
                     lPermutWeight(2, j2) & "|" & lPermutWeight(3, j3) & "|" & _
                     lPermutWeight(1, k) & "|" & lPermutWeight(2, k2) & "|" & _
                     lPermutWeight(3, k3)) = 1
      oGetRidofDupes(lPermutWeight(1, i) & "|" & lPermutWeight(2, i2) & "|" & _
                     lPermutWeight(3, i3) & "|" & lPermutWeight(1, k) & "|" & _
                     lPermutWeight(2, k2) & "|" & lPermutWeight(3, k3) & "|" & _
                     lPermutWeight(1, j) & "|" & lPermutWeight(2, j2) & "|" & _
                     lPermutWeight(3, j3)) = 1
      oGetRidofDupes(lPermutWeight(1, j) & "|" & lPermutWeight(2, j2) & "|" & _
                     lPermutWeight(3, j3) & "|" & lPermutWeight(1, i) & "|" & _
                     lPermutWeight(2, i2) & "|" & lPermutWeight(3, i3) & "|" & _
                     lPermutWeight(1, k) & "|" & lPermutWeight(2, k2) & "|" & _
                     lPermutWeight(3, k3)) = 1
      oGetRidofDupes(lPermutWeight(1, j) & "|" & lPermutWeight(2, j2) & "|" & _
                     lPermutWeight(3, j3) & "|" & lPermutWeight(1, k) & "|" & _
                     lPermutWeight(2, k2) & "|" & lPermutWeight(3, k3) & "|" & _
                     lPermutWeight(1, i) & "|" & lPermutWeight(2, i2) & "|" & _
                     lPermutWeight(3, i3)) = 1
      oGetRidofDupes(lPermutWeight(1, k) & "|" & lPermutWeight(2, k2) & "|" & _
                     lPermutWeight(3, k3) & "|" & lPermutWeight(1, i) & "|" & _
                     lPermutWeight(2, i2) & "|" & lPermutWeight(3, i3) & "|" & _
                     lPermutWeight(1, j) & "|" & lPermutWeight(2, j2) & "|" & _
                     lPermutWeight(3, j3)) = 1
      oGetRidofDupes(lPermutWeight(1, k) & "|" & lPermutWeight(2, k2) & "|" & _
                     lPermutWeight(3, k3) & "|" & lPermutWeight(1, j) & "|" & _
                     lPermutWeight(2, j2) & "|" & lPermutWeight(3, j3) & "|" & _
                     lPermutWeight(1, i) & "|" & lPermutWeight(2, i2) & "|" & _
                     lPermutWeight(3, i3)) = 1
      wsO.Cells(v, 1) = (v + 1) \ n
      wsO.Cells(v, 2) = lPermutWeight(1, i) + lPermutWeight(2, i2) + lPermutWeight(3, i3)
      wsO.Cells(v, 3) = lPermutWeight(1, i)
      wsO.Cells(v, 4) = lPermutWeight(2, i2)
      wsO.Cells(v, 5) = lPermutWeight(3, i3)
      wsO.Cells(v + 1, 3) = lPermutWeight(1, j)
      wsO.Cells(v + 1, 4) = lPermutWeight(2, j2)
      wsO.Cells(v + 1, 5) = lPermutWeight(3, j3)
      wsO.Cells(v + 2, 3) = lPermutWeight(1, k)
      wsO.Cells(v + 2, 4) = lPermutWeight(2, k2)
      wsO.Cells(v + 2, 5) = lPermutWeight(3, k3)
      wsO.Cells(v, 6) = vCount(1, 1) - IIf(lPermutSubGroupIdx(1, i) = 1, 1, 0) - _
                        IIf(lPermutSubGroupIdx(1, j) = 1, 1, 0) - _
                        IIf(lPermutSubGroupIdx(1, k) = 1, 1, 0)
      wsO.Cells(v, 7) = vCount(2, 1) - IIf(lPermutSubGroupIdx(2, i2) = 1, 1, 0) - _
                        IIf(lPermutSubGroupIdx(2, j2) = 1, 1, 0) - _
                        IIf(lPermutSubGroupIdx(2, k2) = 1, 1, 0)
      wsO.Cells(v, 8) = vCount(3, 1) - IIf(lPermutSubGroupIdx(3, i3) = 1, 1, 0) - _
                        IIf(lPermutSubGroupIdx(3, j3) = 1, 1, 0) - _
                        IIf(lPermutSubGroupIdx(3, k3) = 1, 1, 0)
      wsO.Cells(v + 1, 6) = vCount(1, 2) - IIf(lPermutSubGroupIdx(1, i) = 2, 1, 0) - _
                            IIf(lPermutSubGroupIdx(1, j) = 2, 1, 0) - _
                            IIf(lPermutSubGroupIdx(1, k) = 2, 1, 0)
      wsO.Cells(v + 1, 7) = vCount(2, 2) - IIf(lPermutSubGroupIdx(2, i2) = 2, 1, 0) - _
                            IIf(lPermutSubGroupIdx(2, j2) = 2, 1, 0) - _
                            IIf(lPermutSubGroupIdx(2, k2) = 2, 1, 0)
      wsO.Cells(v + 1, 8) = vCount(3, 2) - IIf(lPermutSubGroupIdx(3, i3) = 2, 1, 0) - _
                            IIf(lPermutSubGroupIdx(3, j3) = 2, 1, 0) - _
                            IIf(lPermutSubGroupIdx(3, k3) = 2, 1, 0)
      wsO.Cells(v + 2, 6) = vCount(1, 3) - IIf(lPermutSubGroupIdx(1, i) = 3, 1, 0) - _
                            IIf(lPermutSubGroupIdx(1, j) = 3, 1, 0) - _
                            IIf(lPermutSubGroupIdx(1, k) = 3, 1, 0)
      wsO.Cells(v + 2, 7) = vCount(2, 3) - IIf(lPermutSubGroupIdx(2, i2) = 3, 1, 0) - _
```

```
                              IIf(lPermutSubGroupIdx(2, j2) = 3, 1, 0) - _
                              IIf(lPermutSubGroupIdx(2, k2) = 3, 1, 0)
        wsO.Cells(v + 2, 8) = vCount(3, 3) - IIf(lPermutSubGroupIdx(3, i3) = 3, 1, 0) - _
                              IIf(lPermutSubGroupIdx(3, j3) = 3, 1, 0) - _
                              IIf(lPermutSubGroupIdx(3, k3) = 3, 1, 0)
        wsO.Cells(v, 9) = vWeight(1, 1)
        wsO.Cells(v, 10) = vWeight(2, 1)
        wsO.Cells(v, 11) = vWeight(3, 1)
        wsO.Cells(v + 1, 9) = vWeight(1, 2)
        wsO.Cells(v + 1, 10) = vWeight(2, 2)
        wsO.Cells(v + 1, 11) = vWeight(3, 2)
        wsO.Cells(v + 2, 9) = vWeight(1, 3)
        wsO.Cells(v + 2, 10) = vWeight(2, 3)
        wsO.Cells(v + 2, 11) = vWeight(3, 3)
        v = v + 3
        If v > 98 Then Exit For
      End If
  End If
Next u
Debug.Print u & " iterations, " & v \ n & " solutions"
wsO.Cells.EntireColumn.AutoFit
End With
End Sub
```

CombinationsWithMinRemainingWeight Program Code

```
Sub CombinationsWithMinRemainingWeight()

Dim i                           As Long
Dim j                           As Long
Dim k                           As Long
Dim m                           As Long
Dim maxsum                      As Long
Dim n                           As Long
Dim sum(1 To 33)                As Long
Dim t                           As Long
Dim u                           As Long
Dim v                           As Long
Dim w                           As Long

Dim vCount                      As Variant
Dim vC(1 To 33)                 As Variant
Dim vCi(1 To 3)                 As Variant

Dim state                       As SystemState

With Application.WorksheetFunction
Set state = New SystemState
i = 1
Do While wsI.Cells(2, i) <> ""
  i = i + 1
Loop
n = (i - 1) \ 2
vCount = .Transpose(.Transpose(Range(wsI.Cells(3, 1), wsI.Cells(3, n).End(xlDown))))
For i = 1 To n
  k = 0
  For j = 1 To UBound(vCount, 2)
    k = k + vCount(j, i)
  Next j
  If k < n Then
    Call MsgBox("Not enough items in column " & i, vbOKOnly, "Error")
    Exit Sub
  End If
Next i
m = j - 1

i = 2
t = wsO.Cells(i, 1)
Do While t <> 0
  sum(t) = wsO.Cells(i, 2)
  vC(t) = .Transpose(.Transpose(Range(wsO.Cells(i, 6), wsO.Cells(i + 2, 8))))
  i = i + 3
  t = wsO.Cells(i, 1)
Loop

t = 0
maxsum = 0
For i = 1 To 33
  vCi(1) = vC(i)
  For j = 1 To 33
    vCi(2) = vCi(1)
    For m = 1 To 3
      For n = 1 To 3
        If vCi(1)(m, n) < vCount(m, n) - vC(j)(m, n) Then GoTo Label_Next_j
        vCi(2)(m, n) = vCi(1)(m, n) - vCount(m, n) + vC(j)(m, n)
      Next n
```

```
    Next m
    For k = 1 To 33
      vCi(3) = vCi(2)
      For m = 1 To 3
        For n = 1 To 3
          If vCi(2)(m, n) < vCount(m, n) - vC(k)(m, n) Then GoTo Label_Next_k
          vCi(3)(m, n) = vCi(2)(m, n) - vCount(m, n) + vC(k)(m, n)
        Next n
      Next m

      If maxsum <= 3 * (sum(i) + sum(j) + sum(k)) Then
        maxsum = 3 * (sum(i) + sum(j) + sum(k))
        t = t + 1
        Debug.Print t, maxsum, i, j, k
      End If

Label_Next_k:
    Next k
Label_Next_j:
  Next j
Next i

End With

End Sub
```

A maintenance-free Database

One big issue with Microsoft Excel is the intrinsic mixture of data and of programs (formulas). VBA can help a bit here, but if you really want to separate the data from the programs you need to use a database.

My former colleague Mike R. developed a tiny but powerful simple database. It consists of two tables and two stored procedures. I use this with MS SQL Server. You can feed data into it via Excel and retrieve data with Excel.

This database was implemented to

- Support selected processes with a simple "central" data repository
- Get rid of individual spreadsheets which have to be maintained with high effort
- Simplify the task of data retrieval and of data storage
- Enable additional calculations over defined periods of time and "slice & dice" calculations
- Precisely define user or process access rights

This database is a simple all-purpose database. After the initial setup there is no further need for database table maintenance or index maintenance (apart from deleting old records to avoid the database becoming too big).

Limitations

- The database is not designed for mass storage – you are advised to define a regular data deletion (maintenance) procedure for each new field / data you regularly add to this database.
- The design is simple and field orientated. There is no support for curves or for surfaces – which means you cannot store nor retrieve curves or surfaces in one go. All points would have to be dealt with individually. This is possible but it creates quite an overhead to the database engine. So if this needs to be used it should only be temporary and not for mass storage.
- Fields are stored as string values. The user needs to convert all data into the required data types

Responsibilities

There are 3 levels of access or responsibilities:

- Technology and help desk support – After initial setup the database is backed up on a daily basis, repair or restore can be done within two hours, new users with read or read/write access can be introduced or old users can be deleted within 24h.
- Super user with read/write access – he is the database owner who defines new fields and feeds the database with approved rate sources.
- Other users with read-only access.

System Documentation

Technology is responsible for the database setup, user setup and for repair or restore of the database. And Technology has to ensure that this database is up and running with a monthly average availability of 99% during normal working hours. New users with read-only or read/write access can be introduced or old users can be deleted within 24h.

The super user is the owner of this database, all access right changes have to be approved by him. Only he should have read/write access to the database.

For setup details refer to the Appendix SQL Code. Please notice that the database consists of only two tables and two stored procedures. The two stored procedures provide a sophisticated write and read access to the database. Since they are stored on the server side the client (user) does not need to invest a big effort but he can easily make use of them. The database structure is not intended to change over time. This means that the database maintenance effort is reduced to a possible minimum: Only if the database gets too big old records will have to be deleted. If the super user defines a regular maintenance procedure which deletes old data records for all fields which are regularly added to the database the effort on top of this should be zero.

Please notice: Technology should be able to rebuild or to recreate this database within two hours. This includes corruption of database indices.

User Documentation

The access structure to the database is fairly simple: There is a class of super users (normally one only) with write access. All other users have only read access.

Super User with Read/Write Access

The super user is the owner and the only user with write access to the database. This can be reflected by a relevant AD group. This is not intended to be enhanced for other super users. The super user is responsible for all field definitions and for all feeds into the database. Later we might need to delete old unnecessary records, too. He is the only party to approve additional users or super users.

For the Excel write interface into the database refer to Appendix VBA Code.

Normal User with Read-Only Access

For the Excel read interface from the database refer to Appendix VBA Code. This can be reflected at the relevant AD group.

Examples to retrieve static information:

The Excel function call

```
=sb_get_param("BPIZC610","PX_CLOSE_1D","20110121","RILO")
```

would result in "29.205". Please note that this value is a string value! In order to get a number of type double you would need to convert the result yourself, for example:

```
=--sb_get_param("BPIZC610","PX_CLOSE_1D","20110121","RILO")
```

The Excel function call

```
=sb_get_param("FR0010850719","SECURITY_NAME","19000101","Bloomberg")
```

would result in "COFP4.379 02/17".

Static data is stored with the date 1-Jan-1900. Please notice that the database would also have returned this result if you had called it with a younger date than that. This is because the sb_get_param function returns the youngest entry which is older or equal to the request date. You would need to use the sb_get_paramarray function to retrieve the complete database record (date included) to check the exact date of the returned value.

Example to retrieve market (dynamic) data:

The Excel function call

```
=sb_get_param_array("US172967EZ03","PRICE_MID","20101223","Xtrakter")
```

would result in {"US172967EZ03","PRICE_MID","Xtrakter",40535,"102.172"} Here 40535 is the numerical date of 23-Dec-2010.

Appendix – SQL Code

```
CREATE TABLE [dbo].[param] (
 [identifier] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
 [param] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
 [source] [varchar] (30) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
 [fromDate] [datetime] NULL ,
 [toDate] [datetime] NULL ,
 [value] [varchar] (500) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
 [updTime] [datetime] NULL
) ON [PRIMARY]

GO

CREATE TABLE [dbo].[param_details] (
 [source] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
 [param] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
 [idYN] [char] (1) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
 [rename] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
 [priority] [int] NULL ,
 [inputYN] [char] (1) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY]

GO

CREATE  UNIQUE  INDEX [param_pk] ON [dbo].[param]([identifier], [param], [toDate], [fromDate], [source]) ON
[PRIMARY]

GO

CREATE  INDEX [param_val] ON [dbo].[param]([param], [value], [identifier], [toDate]) ON [PRIMARY]

GO
```

```
SET QUOTED_IDENTIFIER ON

GO

SET ANSI_NULLS ON

GO

CREATE proc get_param
 @identifier  varchar(100),
 @param  varchar(100),
 @source  varchar(30),
 @dated   datetime
as
 if @identifier > ''
 begin
  select identifier, param, source, fromDate, value
  from param
  where identifier = @identifier
   and param like isnull( @param,param )
   and source like isnull( @source,source )
   and ( @dated >= fromDate
    and @dated < isnull( toDate, '1 jan 3000' )
    or @dated is null and toDate is null )
  union
  select identifier, param, source, fromDate, value
  from param
  where identifier = @identifier
   and param like isnull( @param,param )
   and source like isnull( @source,source )
   and fromDate is null
 end
 else
 begin
  select identifier, param, source, fromDate, value
  from param
  where param like isnull( @param,param )
   and source like isnull( @source,source )
   and ( @dated >= fromDate
    and @dated < isnull( toDate, '1 jan 3000' )
    or @dated is null and toDate is null )
  union
  select identifier, param, source, fromDate, value
  from param
  where param like isnull( @param,param )
   and source like isnull( @source,source )
   and fromDate is null
 end

GO

CREATE procedure [dbo].[set_param]
 @identifier   varchar(100),
 @param   varchar(100),
 @source   varchar(30),
 @dated    datetime,
 @value    varchar(500),
 @stopLoop   char(1) = 'n'
as
 set nocount on

 declare @vf  datetime,
  @vt  datetime,
  @rename  varchar(100),
  @priority int

 -- ignore if bad params
 if  isnull( @param,'' ) = ''
  or isnull( @identifier,'' ) = ''
  or isnull( @value,'' ) = ''
  or isnull( @source,'' ) = ''
  or @value like '#N/A%'
  or @value ='?'
 begin
  return
 end

 select @rename = rename,
  @priority = priority
 from param_details
 where source = @source
  and param = @param
  and @stopLoop = 'n'

 if @rename > ''
 begin
  if not exists(
   -- an existing value from a higher priority source
   select 1
   from param p1
```

```
  where p1.identifier = @identifier
   and p1.toDate is null
      and p1.fromDate >= @dated
   and p1.param in (
    select param
    from param_details
    where source=@source
     and rename = @rename
     and param != @param
     and priority < @priority ) )
 begin
  exec set_param @identifier, @rename, @source, @dated, @value, 'y'
 end
end

if @source = 'input'
begin
 select @dated = getdate()
end

-- if static value (ie not time dependant)
if @dated is null
begin

 -- if value unchanged then return
 if exists(
  select 1
  from param
  where identifier = @identifier
   and param = @param
   and source = @source
   and value = @value
   and fromDate is null
   and toDate is null )
 begin
  return
 end

 -- update value to new value if it exists
 update param
 set value = @value,
  updTime = getdate()
 where identifier = @identifier
  and param = @param
  and source = @source
  and fromDate is null

 -- insert new value if not
 if @@rowcount = 0
 begin
  insert param ( identifier, param, source,  fromDate, toDate, value,  updTime )
  values ( @identifier, @param, @source, null, null, @value, getdate() )
 end
end
else
begin

 -- if value unchanged then return
 if exists(
  select 1
  from param
  where identifier = @identifier
   and param = @param
   and source = @source
   and @dated >= fromDate
   and @dated < isnull( toDate, '1 jan 3000' )
   and value = @value )
 begin
  return
 end

 -- get dates of existing record
 select @vf = fromDate,
  @vt = toDate
 from param
 where identifier = @identifier
  and param = @param
  and source = @source
  and @dated >= fromDate
  and @dated < isnull( toDate, '1 jan 3000' )

 -- if before any current records
 if @vt is null
 begin
  -- get next date
  select @vt = min( fromDate )
  from param
  where identifier = @identifier
   and param = @param
   and source = @source
```

```
    and fromDate > @dated
  end

  -- update fromDate of any existing param for this source on this date
  update param
  set toDate = @dated,
   updTime = getdate()
  where identifier = @identifier
   and param = @param
   and source = @source
   and fromDate = @vf

  -- add new value
  insert param ( identifier, param, source,  fromDate, toDate, value,  updTime )
  select @identifier, @param, @source, @dated, @vt, @value, getdate()

  -- if overwriting old data
  delete param
  where identifier = @identifier
   and param = @param
   and source = @source
   and fromDate = @dated
   and value != @value
 end

GO
```

Appendix – VBA Code

```
'Necessary reference: Microsoft ActiveX Data Objects 2.8 Library [for ADODB.Connection]
'Necessary reference: Microsoft Forms 2.0 Object Library [for DataObject]

Dim Gcn As New ADODB.Connection
Dim GsServerName As String, GsDatabaseName As String

Sub sb_open_DB()
'Change History:
'Version Date       Programmer Change
'1.00    21/12/2010 Bernd      Create

If Gcn.State = 0 Then

    'Specify the OLE DB provider.
    Gcn.Provider = "sqloledb"
    GsServerName = "SBSERVER\SB_01"
    GsDatabaseName = "SULPROBIL"

    'Set SQLOLEDB connection properties.
    Gcn.Properties("Data Source").Value = GsServerName
    Gcn.Properties("Initial Catalog").Value = GsDatabaseName

    'Windows NT authentication.
    Gcn.Properties("Integrated Security").Value = "SSPI"

    'Open the database.
    Gcn.Open

End If
End Sub

Function sb_set_param(sIdentifier As String, sParam As String, sSource As String, _
                      Optional ByVal sDated As String = "19000101", Optional sValue As String = "") As
Boolean
'Stores data in database
'Change History:
'Version Date       Programmer Change
'1.00    26/08/2009 Bernd      Create
'1.01    03/06/2011 Bernd      Make 4th param ByVal because it get changed

Dim stSQL As String

If sValue = "" Then
    sValue = "null"
Else
    sValue = "'" & sValue & "'"
End If

If sDated = "19000101" Then
    sDated = "null"
Else
    sDated = "'" & sDated & "'"
End If

stSQL = "exec set_param '" & sIdentifier & _
        "', '" & sParam & _
```

```
        "', '" & sSource & _
        "', " & sDated & _
        ", " & sValue

'On Error GoTo errorexit

If Gcn.state = 0 Then
    Call sb_open_DB
End If

Gcn.Execute (stSQL)
sb_set_param = True

Exit Function

errorexit:
sb_set_param = False

End Function

Sub sb_delete(dtFrom As Date, dtTo As Date, _
               Optional sSource As String = "Markit")
'Delete database records younger than CdtFrom and older than CdtTo.
'Change History:
'Version Date       Programmer Change
'1.00    08/01/2011 Bernd      Create
'Const CdtFrom = #1/1/1900# 'Remember: #MM/DD/YYYY# is Excel's internal date format!
'Const CdtTo = #3/1/2011#  'Remember: #MM/DD/YYYY# is Excel's internal date format!

Dim stSQL As String

Debug.Print "From " & Format(dtFrom, "DD-MMM-YYYY") & " to " & Format(dtTo, "DD-MMM-YYYY")

stSQL = "delete from param where fromDate > '" & Format(dtFrom, "YYYYMMDD") & _
        "' and toDate < '" & Format(dtTo, "YYYYMMDD") & _
        "' and source = '" & sSource & "'"

Debug.Print stSQL

If Gcn.state = 0 Then
    Call sb_open_DB
End If

Gcn.Execute (stSQL)

Debug.Print "Finished."

End Sub

Function sb_get_param(sIdentifier As String, sParam As String, _
                sDated, _
                Optional sSource As String = "Bloomberg") As Variant
'Retrieves data from database
'Change History:
'Version Date       Programmer Change
'1.00    21/12/2010 Bernd      Create

Dim stSQL As String
Dim vdbreturn As Variant

stSQL = "exec get_param '" & sIdentifier & _
        "', '" & sParam & _
        "', '" & sSource & _
        "', '" & sDated & "'"

On Error GoTo errorexit

If Gcn.State = 0 Then
    Call sb_open_DB
End If

vdbreturn = Gcn.Execute(stSQL)
sb_get_param = vdbreturn(4)

Exit Function

errorexit:
On Error GoTo 0
sb_get_param = CVErr(xlErrValue)

End Function

Function sb_get_param_array(sIdentifier As String, sParam As String, _
                sDated, _
                Optional sSource As String = "Bloomberg") As Variant
'Retrieves data from database
'Return variant contains:
'1 - Identifier, for example "US912828HU78"
'2 - Parameter (field), for example "PRICE_MID"
'3 - Source, for example "Bloommberg"
```

```
'4 - Date, for example #12/23/2010#
'5 - Value, for example "100.32"
'Change History:
'Version Date        Programmer Change
'1.00    23/12/2010 Bernd      Create

Dim vdbreturn As Variant
Dim vreturn(1 To 5) As Variant
Dim stSQL As String

stSQL = "exec get_param '" & sIdentifier & _
        "', '" & sParam & _
        "', '" & sSource & _
        "', '" & sDated & "'"

On Error GoTo errorexit

If Gcn.State = 0 Then
    Call sb_open_DB
End If

vdbreturn = Gcn.Execute(stSQL)
vreturn(1) = vdbreturn(0)
vreturn(2) = vdbreturn(1)
vreturn(3) = vdbreturn(2)
vreturn(4) = vdbreturn(3)
vreturn(5) = vdbreturn(4)
sb_get_param_array = vreturn

Exit Function

errorexit:
On Error GoTo 0
sb_get_param_array = CVErr(xlErrValue)

End Function
```

Excursus: Compare Correlation Matrices

"Perl: The Swiss–Army chainsaw of scripting languages." [Henry Spencer]

Abstract

A few years ago, I developed this Perl program to compare correlation matrices for an AlgoOne client. Over time, I expanded the program to read the RMLinks.cfg file, allowing new risk factors to be automatically included.
The implementation was done in Perl rather than Excel/VBA because the execution time needed to be kept short—the matrices contained around 7,000 risk factors—and the program had to be capable of running in batch mode.

Implemention Approach

My implemention approach was:

1. Read first matrix
   Checks:
   Matrix quadratic?
   Risk factor order left->right (top row) == top->bottom (leftmost column)?
   Diagonals == 1 (warning)?
   No NC category (warning if there is)?
   Matrix symmetric: M(i,j) == M(j,i) for all i,j?
   [Not for DC files because not given there.]

2. Read second matrix
   Checks identical to above

3. Risk factors in both matrices identical?
   Warn about risk factors which are in first matrix but not in second and vice versa
   Highlight outliers per category
   Highlight outliers per ccy

Parameters

b - breaches: do not report differences between the two input matrices but breaches beyond tolerances.
d - debug [level] gives debugging information at detail level "level".
   level 1: -.
   level 2: -.
   level 3: Print all elements of matrices 1 and 2.
f - read deviation file [-f needs to be followed by a valid filename]. Reads min and max values for all slices for differences which should be ignored during comparison. See option -w to get format example.
h - help: list parameters and their explanation.

i - ignore risk factors in a given file [-i needs to be followed by a valid filename].
m - set max rank index [default is 6 (=return highest 3 and lowest 3 of each slice); m needs to be even and >= 4 !.
n - tolerate risk factor category NC.
r - set Algo risk factor category file [default is "./RMLinks.cfg"].
s - summarize findings, no detailed warnings or error messages.
t - read file with tolerated changes for each matrix element and apply tolerance check.
v - print version.
w - write deviation file with min and max values of all slices. This file is comma-separated to be easily readable via Excel. It can be amended and used with option -f later. [-w needs to be followed by a valid filename, preferrably ending with ".csv"].
x - read translation table [-x needs to be followed by a valid filename]. Risk factor names of matrix 1 will be translated by second name in comma-separated row.

Sample Program Call

A typical call of this program by a shell script could look like:

```
$PERL_LIB/compareCM.pl \
-i $PERL_LIB/Ignored_RiskFactors.csv \
-m 6 \
-n \
-r $ALGO_TOP/RMLinks.cfg \
-w $PERL_LIB/VCV_Deviation_Matrix.csv \
$VCV_PREVIOUS \
$VCV_CURRENT
```

compareCM.pl Program Code

```
#!/usr/bin/perl
#
# Implementation Approach:
#
# 1. Read first matrix
#    Checks:
#    Matrix quadratic?
#    Risk factor order left->right (top row) == top->bottom (leftmost column)?
#    Diagonals == 1 (warning)?
#    No NC category (warning if there is)?
#    Matrix symmetric: M(i,j) == M(j,i) for all i,j?
#    [Not for DC files because not given there.]
#
# 2. Read second matrix
#    Checks identical to above
#
# 3. Risk factors in both matrices identical?
#    Warn about risk factors which are in first matrix but not in second and vice versa
#    Highlight outliers per category
#    Highlight outliers per ccy
#
# Version Date       Author         Comment
# 1.18    26/12/2018 Bernd Plumhoff Anonymized version
my $version = "'1.18,26/12/2018";

use strict;
use warnings;
use List::Util qw[min max];
use feature "switch";
use Getopt::Std;

##########################################################
#                                                        #
#                   Process parameters                   #
#                                                        #
##########################################################

our $opt_b;           # b - breaches: do not report differences between the two input matrices
```

```
                        #     but breaches beyond tolerances
our $opt_d;             # d - debug ["level"] gives debugging information at detail level "level"
                        #     level 1: -
                        #     level 2: -
                        #     level 3: Print all elements of matrices 1 and 2
our $opt_f;             # f - read deviation file [-f needs to be followed by a valid filename]
                        #     Reads min and max values for all slices for differences which should
                        #     be ignored during comparison. See option -w to get format example.
our $opt_h;             # h - help: list parameters and their explanation
our $opt_i;             # i - ignore risk factors in a given file [-i needs to be followed by
                        #     a valid filename]
our $opt_m;             # m - set max rank index [default is 6 (=return highest 3
                        #     and lowest 3 of each slice); m needs to be even and >= 4 !
our $opt_n;             # n - tolerate risk factor category NC
our $opt_r;             # r - set Algo risk factor category file [default is "./RMLinks.cfg"]
our $opt_s;             # s - summarize findings, no detailed warnings or error messages
our $opt_t;             # t - read tolerance file and apply tolerance check
our $opt_v;             # v - print version
our $opt_w;             # w - write deviation file with min and max values of all slices.
                        #     This file is ","-separated to be easily readable via Excel.
                        #     It can be amended and used with option -f later
                        #     [-w needs to be followed by a valid filename,
                        #     preferrably ending with ".csv"]
our $opt_x;             # x - read translation table [-x needs to be followed by a valid filename].
                        #     Risk factor names of matrix 1 will be translated by second name in
                        #     comma-separated row

getopts('bd:f:hi:m:nr:st:vw:x:');

if (defined $opt_h) {
   print "$0 parameters:\n" .
      "b - breaches: do not report differences between the two input matrices\n" .
      "    but breaches beyond tolerances\n" .
      "d - debug [\"level\"] gives debugging information at detail level \"level\"\n" .
      "    level 1: -\n" .
      "    level 2: -\n" .
      "    level 3: Print all elements of matrices 1 and 2\n" .
      "f - read deviation file [-f needs to be followed by a valid filename]\n" .
      "    Reads min and max values for all slices for differences which should\n" .
      "    be ignored during comparison. See option -w to get format example.\n" .
      "h - help: list parameters and their explanation\n" .
      "i - ignore risk factors in a given file [-i needs to be followed by\n" .
      "    a valid filename]\n" .
      "m - set max rank index [default is 6 (=return highest 3\n" .
      "    and lowest 3 of each slice); m needs to be even and >= 4 !\n" .
      "n - tolerate risk factor category NC\n" .
      "r - set Algo risk factor category file [default is \"./RMLinks.cfg\"]\n" .
      "s - summarize findings, no detailed warnings or error messages\n" .
      "t - read file with tolerated changes for each matrix element and apply\n" .
       "    tolerance check\n" .
      "v - print version\n" .
      "w - write deviation file with min and max values of all slices.\n" .
      "    This file is \",\"-separated to be easily readable via Excel.\n" .
      "    It can be amended and used with option -f later\n" .
      "    [-w needs to be followed by a valid filename,\n" .
      "    preferrably ending with \".csv\"]\n" .
      "x - read translation table [-x needs to be followed by a valid filename].\n" .
      "    Risk factor names of matrix 1 will be translated by second name in\n" .
      "    comma-separated row\n";
   exit(0);
}

if (defined $opt_v) {
   print "$0\tVersion: $version\n";
   exit(0);
}

$opt_m ||= 6;
if ($opt_m < 4 || $opt_m % 2 != 0) {
   die "$0: parameter -m needs to be followed by an even number >= 4!\n";
}

# Initialize risk factor categories from file
# Please note that you will find info to fill that file in $ALGO_TOP\cfg\
$opt_r ||= "./RMLinks.cfg";

our $doutfile;
if (defined $opt_w) {
   if (defined $opt_f) {
      die "$0: Do not use option -w together with option -f!\n";
   }
   open($doutfile, ">",  $opt_w) || die "$0: Can't open $opt_w: $!";
   $opt_b ||= "";
   print $doutfile "$0,$version,Min,Max,Total,Tolerance Breach Down,Tolerance Breach Up," .
      "$opt_b,Do not change cells on the left." .
      " They are used by an implicit format check.\n";
}

our $epsilon = 0.000001;   # Tolerance for floating number rounding errors. Please note that the
              # value 1e-6 has been chosen carefully: it coincides with the precision
```

```
            # of printf's format %f (see sub printarr below, please).

##############################################################
#                                                            #
#                    Read deviation file                     #
#                                                            #
##############################################################

our $dinfile;
our $line;
our @fields=();
our @devglobal=();      # Stores global deviations as read by -f option if any
our %devrfcat=();       # Stores deviations per category if any
our %devccy=();         # Stores deviations per ccy if they exist
our %devrfcpair=();     # Stores deviations per category pair if they exist
if (defined $opt_f) {
   open($dinfile, "<",  $opt_f) || die "$0: Can't open $opt_f: $!";
   @fields = split(",", <$dinfile>);
   if ($0 ne $fields[0] || $version ne $fields[1] . ";" . $fields[2]) {
      print STDERR "$0: Warning: $0,$version expected but $fields[0]," .
            "$fields[1],$fields[2] found.\n";
   }
   if (! $opt_b && $fields[5] || $opt_b && ! $fields[5]) {
      $opt_b ||= "";
      die "$0: Option -b setting ($opt_b) does not match deviation file setting ($fields[5])";
   }
   while($line = <$dinfile>) {
      @fields = split(";", substr($line,0,length($line)-2));
      given($fields[0]) {
         when ("GLOBAL") {
            $devglobal[0] = $fields[3];
            $devglobal[1] = $fields[4];
         }
         when ("CATEGORY") {
            $devrfcat{$fields[1]}[0] = $fields[3];
            $devrfcat{$fields[1]}[1] = $fields[4];
         }
         when ("CURRENCY") {
            $devccy{$fields[1]}[0] = $fields[3];
            $devccy{$fields[1]}[1] = $fields[4];
         }
         when ("CATEGORYPAIR") {
            $devrfcpair{$fields[1] . "," . $fields[2]}[0] = $fields[3];
            $devrfcpair{$fields[1] . "," . $fields[2]}[1] = $fields[4];
         }
         default {
            die "$0: I don't know what to do with deviation definition $fields[0]";
         }
      }
   }
   close $dinfile;
}

##############################################################
#                                                            #
#   Read file with riskfactors which should get ignored      #
#                                                            #
##############################################################

our %rfignore=();    # Stores risk factors to be ignored
if (defined $opt_i) {
   open($dinfile, "<",  $opt_i) || die "$0: Can't open $opt_i: $!";
   while($line = <$dinfile>) {
      $line =~ s/[\r\n]+//g;
      $rfignore{$line} = 1;
   }
   close $dinfile;
}

##############################################################
#                                                            #
#    Read file with translation table for risk factors       #
#                                                            #
##############################################################

our %rftrans=(); # Stores risk factors to be translated
if (defined $opt_x) {
   open($dinfile, "<",  $opt_x) || die "$0: Can't open $opt_x: $!";
   while($line = <$dinfile>) {
      $line =~ s/[\r\n]+//g;
      @fields = split(",", $line);
      $rftrans{$fields[0]} = $fields[1];
   }
   close $dinfile;
}

##############################################################
#                                                            #
#                    Read tolerance matrix                   #
#                                                            #
```

```
##########################################################

our $tinfile;
our @tnames=();      # risk factor names in tolerance matrix
our %hashtnames=();  # hash table of risk factor names and their position
our @tol=();      # array with tolerances (per risk factor)
our $i;           # row index for loops
our $j;           # column index in loops
if (defined $opt_t) {
    open($tinfile, "<",  $opt_t) || die "$0: Can't open $opt_t: $!";
    $line = <$tinfile>;
    $line =~ s/[\r\n]+//g;
    @tnames = split(",", $line);
    for ($i=1; $i<scalar(@tnames); $i++) {
        $hashtnames{$tnames[$i]} = $i;
    }
    while($line = <$tinfile>) {
        $line =~ s/[\r\n]+//g;
        @fields=split(",", $line);

        # Matrix quadratic?
        if (scalar(@tnames) != scalar(@fields)) {
            print STDERR "$ARGV[0]: matrix is not quadratic: # of columns (" . @tnames .
                   ") != # of fields in row $. (" . @fields . ")!\n";
            exit(1);
        }

        # Risk factor order left->right (top row) == top->bottom (leftmost column)?
        if ($fields[0] ne @tnames[$. - 1]) {
            print STDERR "$ARGV[0], Row $.: risk factor \"$fields[0]\" does not match" .
                " corresponding column header \"@tnames[$. - 1]\".\n";
        }

        # No NC category (warning if there is)?
        ! $opt_n && ! $opt_s && get_rf_category($fields[0]) eq "NC" &&
            print STDERR "$ARGV[0], Row $.: risk factor $fields[0] is linked to category NC.\n";

        # Please note that we only fill triangle above diagonal since we check for symmetry
        for ($i=$. - 2; $i<scalar(@tnames) - 1; $i++) {
            $tol[$. - 2][$i] = $fields[$i + 1];
            defined $opt_d && $opt_d > 2 && print "Tolerance[" . ($. - 2) . "][$i]=" .
                $fields[$i + 1] . "\n";
        }
        for ($i=0; $i<$. - 1; $i++) {
            # Matrix symmetric: M(i,j) == M(j,i) for all i,j?
            if ($fields[$i + 1] != $tol[$i][$. - 2]) {
                print STDERR "$ARGV[0], Row $.: M[" . ($. - 2) . "][" . $i ."] != M[" . $i .
                       "][" . ($. - 2) . "]: " . $fields[$i + 1] . "!=" .
                       $tol[$i][$. - 2] . ".\n";
            }
        }
    }

    # Matrix quadratic?
    if (scalar(@tnames) != $.) {
        print STDERR "$ARGV[0]: matrix is not quadratic: # of columns (" .
            scalar(@tnames) . ") != # of rows ($.)!\n";
        exit(1);
    }

    close $tinfile;

}
our %rfcatpat=();    # links search pattern to risk factor category
our %rfcat=();       # speeds up get_rf_category
our $rfcatcount=0;   # counts risk factor categories in order to build @rfcatrank

our @m1=();       # First matrix
our @m2=();       # Second matrix
our $m2val;       # Temporary value of second matrix
our @m3=();       # Diff matrix: Second minus First
our @names1=();      # risk factor names in matrix 1
our @names2=();      # risk factor names in matrix 2
our %rfnames1=();    # to store row number of risk factor names in matrix 1
our %rfnames2=();    # to store row number of risk factor names in matrix 2
our $ccy;
our $t;           # tolerance

# Variables to process DC file
our @hnames=();      # helper array to split up risk factor names in DC file
our $rf1;
our $rf2;
our $old_rf;      # for row change comparisons of the DC matrices
our $all_rfnames_read;  # Boolean
our $start_next_row;# Boolean
our $is_upper_right_triangle;
    # Boolean
# Variables to rank outliers detected
our @diagnotone1=();# ranks diagonal values <> 1 of matrix 1
our $dno1idx = 0;    # number of elements in @diagnotone1
```

```
our @diagnotone2=(); # ranks diagonal values <> 1 of matrix 2
our $dno2idx = 0;    # number of elements in @diagnotone2
our @totaldiff=();   # ranks all differences of matrix 2 minus matrix 1
our $tdidx = 0;      # number of elements in @totaldiff
our %rfcatrank=();
our %rfcatidx=();    # number of elements in @{$rfcatrank{$rfcatidx{rfname}}}
our %rfcccyrank=();
our %rfcccyidx=();   # number of elements in @{$rfccyrank{$rfcccyidx{rfname}}}
our %rfcpairrank=();
our %rfcpairidx=(); # number of elements in @{$rfcpairrank{$rfcpairidx{rfname}}}

init_rf_category($opt_r);

##############################################################
#                                                            #
#                    1. Read first matrix                    #
#                                                            #
##############################################################

open (FILE, "<", $ARGV[0]) || die "$0: Cannot open $ARGV[0]: $!";
$line = <FILE>;
$line =~ s/[\r\n]+//g;
if (substr($line,0,1) eq ",") {

    # We read a CSV file

    @names1 = split(",", $line);
    for ($i=1; $i<@names1; $i++) {
        $rftrans{$names1[$i]} ||= $names1[$i];
        $rfnames1{$rftrans{$names1[$i]}} = $i;
    }

    while($line = <FILE>) {
        $line =~ s/[\r\n]+//g;
        @fields=split(",", $line);

        # Matrix quadratic?
        if (scalar(@names1) != scalar(@fields)) {
            print STDERR "$ARGV[0]: matrix is not quadratic: # of columns (" . @names1 .
                   ") != # of fields in row $. (" . @fields . ")!\n";
            exit(1);
        }

        # Risk factor order left->right (top row) == top->bottom (leftmost column)?
        if ($fields[0] ne @names1[$. - 1]) {
            print STDERR "$ARGV[0], Row $.: risk factor \"$fields[0]\" does not match" .
                " corresponding column header \"@names1[$. - 1]\".\n";
        }

        # No NC category (warning if there is)?
        ! $opt_n && ! $opt_s && get_rf_category($fields[0]) eq "NC" &&
            print STDERR "$ARGV[0], Row $.: risk factor $fields[0] is linked to category NC.\n";

        # Diagonals == 1 (warning)?
        if (abs($fields[$. - 1] - 1) > $epsilon) {
            ($dno1idx, @diagnotone1) = rankinsert($fields[$. - 1],
                          "$ARGV[0], Row $.: $fields[0]",
                          2, -2, 0, $dno1idx, @diagnotone1)
                          unless $rfignore{$fields[0]};
            ! $opt_s && print STDERR "$ARGV[0], Row $.: Diagonal element is not equal to 1: M[" .
                     ($. - 1) . "," . ($. - 1) . "] == " . $fields[$. - 1] . ".\n";
        }

        # Please note that we only fill triangle above diagonal since we check for symmetry
        for ($i=$. - 2; $i<scalar(@names1) - 1; $i++) {
            $m1[$. - 2][$i] = $fields[$i + 1];
            defined $opt_d && $opt_d > 2 && print "Matrix1[" . ($. - 2) . "][$i]=" .
                $fields[$i + 1] . "\n";
        }
        for ($i=0; $i<$. - 1; $i++) {
            # Matrix symmetric: M(i,j) == M(j,i) for all i,j?
            if ($fields[$i + 1] != $m1[$i][$. - 2]) {
                print STDERR "$ARGV[0], Row $.: M[" . ($. - 2) . "][" . $i ."] != M[" . $i .
                       "][" . ($. - 2) . "]: " . $fields[$i + 1] . "!=" .
                       $m1[$i][$. - 2] . ".\n";
            }
        }
    }

    # Matrix quadratic?
    if (scalar(@names1) != $.) {
        print STDERR "$ARGV[0]: matrix is not quadratic: # of columns (" .
            scalar(@names1) . ") != # of rows ($.)!\n";
        exit(1);
    }

} elsif (substr($line,0,1) eq "\*") {

    # We read a DC file
```

```
$all_rfnames_read = 0;  # We have not identified all risk factor names yet
$i = 1;
$is_upper_right_triangle = 0;
$old_rf = "";

while($line = <FILE>) {
   $line = substr($line,0,length($line)-2);
   next if (($line  =~ /^\*/));
   next if (($line  =~ /^$/));

   @fields = split(",", $line);
   @hnames = split(/\./, $fields[0]);
   if ($is_upper_right_triangle) {
      $rf1 = $hnames[0] . "." . $hnames[1];
      $rf2 = $hnames[2] . "." . $hnames[3];
   } else {
      $rf2 = $hnames[0] . "." . $hnames[1];
      $rf1 = $hnames[2] . "." . $hnames[3];
   }

   if ($. == 4) {
      if ($rf1 eq $old_rf) {
         # DC file is of upper right triangle form
         $is_upper_right_triangle = 1;
         $rf2 = $rf1;
         $rf1 = $hnames[0] . "." . $hnames[1];
      }
   }

   if ($rf2 ne $old_rf) {
      $start_next_row = 1;
   } else {
      $start_next_row = 0;
   }
   $old_rf = $rf2;

   if ($start_next_row) {
      if ($. > 3) {
         $all_rfnames_read = 1;
      }
      if ($rf1 ne $rf2) {
         print STDERR "$ARGV[0], Row $.: matrix is not quadratic: After change of 2." .
               " risk factor name the next diagonal element has" .
               " to have (name 1 == name 2)!\n";
         exit(1);
      }
      # Diagonals == 1 (warning)?
      if (abs($fields[1] - 1) > $epsilon) {
         ($dno1idx, @diagnotone1) = rankinsert($fields[1],
                        "$ARGV[0], Row $.: $rf1",
                        2, -2, 0, $dno1idx, @diagnotone1)
                        unless $rfignore{$rf1};
         ! $opt_s && print STDERR "$ARGV[0], Row $.: Diagonal element is not equal to 1: M[" .
               ($rfnames1{$rftrans{$rf1}}) . "," . ($rfnames1{$rftrans{$rf2}}) .
               "] == " . $fields[1] . ".\n";
      }
   }

   if (! $all_rfnames_read) {
      $rftrans{$rf1} ||= $rf1;
      $names1[$i] = $rftrans{$rf1};
      if (! defined $rfnames1{$rftrans{$rf1}}) {
         $rfnames1{$rftrans{$rf1}} = $i++;
      }
   } else {
      # Risk factor order left->right (top row) == top->bottom (leftmost column)?
      ! defined $rftrans{$rf1} &&
         print STDERR "$ARGV[0], Row $.: risk factor \"$rf1\" does not exist.\n";
      ! defined $rftrans{$rf2} &&
         print STDERR "$ARGV[0], Row $.: risk factor \"$rf2\" does not exist.\n";
   }

   # No NC category (warning if there is)?
   ! $opt_n && ! $opt_s && get_rf_category($rf1) eq "NC" &&
      print STDERR "$ARGV[0], Row $.: risk factor $rf1 is linked to category NC.\n";
   ! $opt_n && ! $opt_s && get_rf_category($rf2) eq "NC" &&
      print STDERR "$ARGV[0], Row $.: risk factor $rf2 is linked to category NC.\n";

   # Please note that we only fill triangle above diagonal
   $m1[$rfnames1{$rftrans{$rf2}} - 1][$rfnames1{$rftrans{$rf1}} - 1] = $fields[1];
   defined $opt_d && $opt_d > 2 && print "Matrix1[" . $rfnames1{$rftrans{$rf2}} .
                  "][$rfnames1{$rftrans{$rf1}}]=" . $fields[1] . "\n";
}

# Matrix quadratic?
if (scalar(@names1) != sqrt(2*$. - 3.75) + 0.5) {
   print STDERR "$ARGV[0]: matrix is not quadratic: # of columns (" . scalar(@names1) .
         ") != # of rows (" . (sqrt(2*$. - 3.75) + 0.5) . ")!\n";
   # exit(1);
}
```

```
}

close(FILE);

! $opt_s && $dno1idx && printarr("Diagonal in matrix 1 not 1:", $dno1idx, @diagnotone1);

##############################################################
#                                                            #
#                    2. Read second matrix                   #
#                                                            #
##############################################################
our $m1name = $ARGV[0];
shift;
$all_rfnames_read = 0;
open (FILE, "<", $ARGV[0]) || die "Cannot open $ARGV[0]: $!";

$line = <FILE>;
$line =~ s/[\r\n]+//g;
if (substr($line,0,1) eq ",") {

    # We read a CSV file

    @names2 = split(",", $line);
    for ($i=1; $i<@names2; $i++) {
        $rfnames2{$names2[$i]} = $i;
    }

    while($line = <FILE>) {
        $line =~ s/[\r\n]+//g;
        @fields=split(",", $line);

        # Matrix quadratic?
        if (scalar(@names2) != scalar(@fields)) {
            print STDERR "$ARGV[0]: matrix is not quadratic: # of columns (" . @names2 .
                   ") != # of fields in row $. (" . @fields . ")!\n";
            exit(1);
        }

        # Risk factor order left->right (top row) == top->bottom (leftmost column)?
        if ($fields[0] ne @names2[$. - 1]) {
            print STDERR "$ARGV[0], Row $.: risk factor \"$fields[0]\" does not match" .
                " corresponding column header \"@names2[$. - 1]\".\n";
        }

        # Warn about risk factors which are in first matrix but not in second
        #if (! (defined $rfnames1{$rftrans{$fields[0]}})) {
        #   print STDERR "$ARGV[0], Row $.: risk factor \"$fields[0]\" does not exist in " .
        #       "$m1name" . "\n";
        #}

        # No NC category (warning if there is)?
        ! $opt_n && ! $opt_s && get_rf_category($fields[0]) eq "NC" &&
            print STDERR "$ARGV[0], Row $.: risk factor $fields[0] is linked to category NC.\n";

        # Diagonals == 1 (warning)?
        if (abs($fields[$. - 1] - 1) > $epsilon) {
            ($dno2idx, @diagnotone2) = rankinsert($fields[$. - 1],
                          "$ARGV[0], Row $.: $fields[0]",
                          2, -2, 0, $dno2idx, @diagnotone2)
                          unless $rfignore{$fields[0]};
            ! $opt_s && print STDERR "$ARGV[0], Row $.: Diagonal element is not equal to 1: M[" .
                       ($. - 1) . "," . ($. - 1) . "] == " . $fields[$. - 1] . ".\n";
        }

        # Please note that we only fill triangle above diagonal since we check for symmetry
        for ($i=$. - 2; $i<scalar(@names2) - 1; $i++) {
            $m2[$. - 2][$i] = $fields[$i + 1];
            defined $opt_d && $opt_d > 2 && print "Matrix2[" . ($. - 2) . "][$i]=" .
                $fields[$i + 1] . "\n";
        }
        for ($i=0; $i<$. - 1; $i++) {
            # Matrix symmetric: M(i,j) == M(j,i) for all i,j?
            if ($fields[$i + 1] != $m2[$i][$. - 2]) {
                print STDERR "$ARGV[0], Row $.: M[" . ($. - 2) . "][" . $i ."] != M[" . $i .
                       "][" . ($. - 2) . "]: " . $fields[$i + 1] . "!=" .
                       $m2[$i][$. - 2] . ".\n";
            }
        }
    }

    # Matrix quadratic?
    if (scalar(@names2) != $.) {
        print STDERR "$ARGV[0]: matrix is not quadratic: # of columns (" . scalar(@names2) .
            ") != # of rows ($.)!\n";
        exit(1);
    }

} elsif (substr($line,0,1) eq "\*") {
```

```
# We read a DC file

$all_rfnames_read = 0;  # We have not identified all risk factor names yet
$i = 1;
$is_upper_right_triangle = 0;
$old_rf = "";

while($line = <FILE>) {
   $line = substr($line,0,length($line)-2);
   next if (($line  =~ /^\*/));
   next if (($line  =~ /^$/));

   @fields = split(",", $line);
   @hnames = split(/\./, $fields[0]);
   if ($is_upper_right_triangle) {
      $rf1 = $hnames[0] . "." . $hnames[1];
      $rf2 = $hnames[2] . "." . $hnames[3];
   } else {
      $rf2 = $hnames[0] . "." . $hnames[1];
      $rf1 = $hnames[2] . "." . $hnames[3];
   }

   if ($. == 4) {
      if ($rf1 eq $old_rf) {
         # DC file is of upper right triangle form
         $is_upper_right_triangle = 1;
         $rf2 = $rf1;
         $rf1 = $hnames[0] . "." . $hnames[1];
      }
   }

   if ($rf2 ne $old_rf) {
      $start_next_row = 1;
   } else {
      $start_next_row = 0;
   }
   $old_rf = $rf2;

   if ($start_next_row) {
      if ($. > 3) {
         $all_rfnames_read = 1;
      }
      if ($rf1 ne $rf2) {
         print STDERR "$ARGV[0], Row $.: matrix is not quadratic: After change of 2." .
            " risk factor name the next diagonal element has to be" .
            " given (name 1 == name 2)!\n";
         exit(1);
      }
      # Diagonals == 1 (warning)?
      if (abs($fields[1] - 1) > $epsilon) {
         ($dno2idx, @diagnotone2) = rankinsert($fields[1],
                     "$ARGV[0], Row $.: $rf1",
                     2, -2, 0, $dno2idx, @diagnotone2)
                     unless $rfignore{$rf1};
         ! $opt_s && print STDERR "$ARGV[0], Row $.: Diagonal element is" .
               " not equal to 1: M[" .
               ($rfnames2{$rf1}) . "," . ($rfnames2{$rf2}) . "] == " .
               $fields[1] . ".\n";
      }
   }

   if (! $all_rfnames_read) {
      $names2[$i] = $rf1;
      if (defined $rfnames2{$rf1}) {
         print STDERR "$ARGV[0], Row $.: risk factor \"$rf1\" already exists.\n";
      } else {
         $rfnames2{$rf1} = $i++;
      }
   } else {
      # Risk factor order left->right (top row) == top->bottom (leftmost column)?
      if (! (defined $rfnames2{$rf1})) {
         print STDERR "$ARGV[0], Row $.: risk factor \"$rf1\" does not exist.\n";
      }
      if (! (defined $rfnames2{$rf2})) {
         print STDERR "$ARGV[0], Row $.: risk factor \"$rf2\" does not exist.\n";
      }
   }

   # No NC category (warning if there is)?
   ! $opt_n && ! $opt_s && get_rf_category($rf1) eq "NC" &&
      print STDERR "$ARGV[0], Row $.: risk factor $rf1 is linked to category NC.\n";
   ! $opt_n && ! $opt_s && get_rf_category($rf2) eq "NC" &&
      print STDERR "$ARGV[0], Row $.: risk factor $rf2 is linked to category NC.\n";

   # Please note that we only fill triangle above diagonal
   $m2[$rfnames2{$rf2} - 1][$rfnames2{$rf1} - 1] = $fields[1];
   defined $opt_d && $opt_d > 2 && print "Matrix2[" . $rfnames2{$rf2} .
                  "][$rfnames2{$rf1}]=" . $fields[1] . "\n";
}
```

```
    # Matrix quadratic?
    if (scalar(@names2) != sqrt(2*$. - 3.75) + 0.5) {
        print STDERR "$ARGV[0]: matrix is not quadratic: # of columns (" . scalar(@names2) .
                ") != # of rows (" . (sqrt(2*$. - 3.75) + 0.5) . ")!\n";
        # exit(1);
    }

}

close(FILE);

! $opt_s && $dno2idx && printarr("Diagonal in matrix 2 not 1:", $dno2idx, @diagnotone2);

# Warn about risk factors which are in first matrix but not in second
foreach (keys %rfnames1) {
    if (! defined $rfnames2{$_}) {
        print STDERR "Risk factors in $m1name but not in $ARGV[0]\n";
        last;
    }
}
foreach (keys %rfnames1) {
    if (! defined $rfnames2{$_}) {
        if ($_ eq $rftrans{$_}) {
            print STDERR $rftrans{$_} . "\n";
        } else {
            print STDERR $rftrans{$_} . ",$_\n";
        }
    }
}

# Warn about risk factors which are in second matrix but not in first
foreach (keys %rfnames2) {
    if (! defined $rfnames1{$_}) {
        print STDERR "Risk factors in $ARGV[0] but not in $m1name\n";
        last;
    }
}
foreach (keys %rfnames2) {
    if (! defined $rfnames1{$_}) {
        print STDERR $_ . "\n";
    }
}

##############################################################
#                                                            #
#                      3. Compare Matrices                   #
#                                                            #
##############################################################

# Calculate differences
defined $opt_d && $opt_d > 2 && print "#Rows=" . scalar(@names1) . "\n";
for ($i=0; $i<scalar(@names1); $i++) {
    defined $opt_d && $opt_d > 2 && print "Row," . $i . ",Start\n";
    defined $opt_d && $opt_d > 2 && print "#Columns=" . scalar(@names1) . "\n";
    for ($j=$i; $j<scalar(@names1); $j++) {
        defined $opt_d && $opt_d > 2 && print "Row," . $i . ",Column," . $j . ",Start\n";
        if (defined $names1[$i + 1] && defined $names1[$j + 1] &&
            defined $rfnames2{$names1[$i + 1]} && defined $rfnames2{$names1[$j + 1]}) {
            $m2val = $m2[min($rfnames2{$names1[$i + 1]} - 1,
                      $rfnames2{$names1[$j + 1]} - 1)][max($rfnames2{$names1[$i + 1]} - 1,
                    $rfnames2{$names1[$j + 1]} - 1)];
            $m3[$i][$j] = $m2val - $m1[$i][$j];
            if (defined $hashtnames{$names1[$i + 1]} && defined $hashtnames{$names1[$j + 1]}) {
                $t = $tol[min($hashtnames{$names1[$i + 1]} - 1,
                    $hashtnames{$names1[$j + 1]} - 1)][max($hashtnames{$names1[$i + 1]} - 1,
                    $hashtnames{$names1[$j + 1]} - 1)];
            } else {
                $t = 0;
            }
            defined $opt_d && $opt_d > 2 && print "Matrix3[" . $i .
                            "][$j]=" . $m3[$i][$j] . ",Tol=$t\n";
            ($tdidx, @totaldiff) = rankinsert($m3[$i][$j],
                get_rf_category($names1[$i + 1]) . ",$names1[$i + 1]," .
                get_rf_category($names1[$j + 1]) . ",$names1[$j + 1]," .
                $m1[$i][$j] . "," . $m2val,
                $devglobal[0], $devglobal[1], $t,
                $tdidx, @totaldiff)
                unless $rfignore{$names1[$i + 1]} || $rfignore{$names1[$j + 1]};
            # If you like to understand the next statement please have a look at
            # http://perldoc.perl.org/perllol.html
            ($rfcatidx{get_rf_category($names1[$i + 1])},
                @{$rfcatrank{get_rf_category($names1[$i + 1])}}) = rankinsert($m3[$i][$j],
                get_rf_category($names1[$i + 1]) . ",$names1[$i + 1]," .
                get_rf_category($names1[$j + 1]) . ",$names1[$j + 1]," .
                $m1[$i][$j] . "," . $m2val,
                $devrfcat{get_rf_category($names1[$i + 1])}[0],
                $devrfcat{get_rf_category($names1[$i + 1])}[1], $t,
                $rfcatidx{get_rf_category($names1[$i + 1])},
                @{$rfcatrank{get_rf_category($names1[$i + 1])}})
                unless $rfignore{$names1[$i + 1]} || $rfignore{$names1[$j + 1]};
```

```
            if (get_rf_category($names1[$i + 1]) ne get_rf_category($names1[$j + 1])) {
                ($rfcatidx{get_rf_category($names1[$j + 1])},
                    @{$rfcatrank{get_rf_category($names1[$j + 1])}}) =
                    rankinsert($m3[$i][$j],
                    get_rf_category($names1[$i + 1]) . ",$names1[$i + 1]," .
                    get_rf_category($names1[$j + 1]) . ",$names1[$j + 1]," .
                    $m1[$i][$j] . "," . $m2val,
                    $devrfcat{get_rf_category($names1[$j + 1])}[0],
                    $devrfcat{get_rf_category($names1[$j + 1])}[1], $t,
                    $rfcatidx{get_rf_category($names1[$j + 1])},
                    @{$rfcatrank{get_rf_category($names1[$j + 1])}})
                    unless $rfignore{$names1[$i + 1]} || $rfignore{$names1[$j + 1]};
            }
            $rfcpairidx{get_rf_category($names1[$i + 1]) . "," .
                get_rf_category($names1[$j + 1])} ||= 0;
            ($rfcpairidx{get_rf_category($names1[$i + 1]) . "," .
                get_rf_category($names1[$j + 1])},
                @{$rfcpairrank{get_rf_category($names1[$i + 1]) . "," .
                get_rf_category($names1[$j + 1])}}) = rankinsert($m3[$i][$j],
                get_rf_category($names1[$i + 1]) . ",$names1[$i + 1]," .
                get_rf_category($names1[$j + 1]) . ",$names1[$j + 1]," .
                $m1[$i][$j] . "," . $m2val,
                $devrfcpair{get_rf_category($names1[$i + 1]) . "," .
                get_rf_category($names1[$j + 1])}[0],
                $devrfcpair{get_rf_category($names1[$i + 1]) . "," .
                get_rf_category($names1[$j + 1])}[1], $t,
                $rfcpairidx{get_rf_category($names1[$i + 1]) . "," .
                get_rf_category($names1[$j + 1])},
                @{$rfcpairrank{get_rf_category($names1[$i + 1]) . "," .
                get_rf_category($names1[$j + 1])}})
                unless $rfignore{$names1[$i + 1]} || $rfignore{$names1[$j + 1]};
            $ccy = substr($names1[$i + 1], 0, 3);
            $rfcccyidx{$ccy} ||= 0;
            ($rfcccyidx{$ccy}, @{$rfcccyrank{$ccy}}) = rankinsert($m3[$i][$j],
                get_rf_category($names1[$i + 1]) . ",$names1[$i + 1]," .
                get_rf_category($names1[$j + 1]) . ",$names1[$j + 1]," .
                $m1[$i][$j] . "," . $m2val,
                $devccy{$ccy}[0], $devccy{$ccy}[1], $t,
                $rfcccyidx{$ccy}, @{$rfcccyrank{$ccy}})
                unless $rfignore{$names1[$i + 1]} || $rfignore{$names1[$j + 1]};
            if ($ccy ne substr($names1[$j + 1], 0, 3)) {
                $ccy = substr($names1[$j + 1], 0, 3);
                $rfcccyidx{$ccy} ||= 0;
                ($rfcccyidx{$ccy}, @{$rfcccyrank{$ccy}}) = rankinsert($m3[$i][$j],
                    get_rf_category($names1[$i + 1]) . ",$names1[$i + 1]," .
                    get_rf_category($names1[$j + 1]) . ",$names1[$j + 1]," .
                    $m1[$i][$j] . "," . $m2val,
                    $devccy{$ccy}[0], $devccy{$ccy}[1], $t,
                    $rfcccyidx{$ccy}, @{$rfcccyrank{$ccy}})
                    unless $rfignore{$names1[$i + 1]} || $rfignore{$names1[$j + 1]};
            }
        }
        defined $opt_d && $opt_d > 2 && print "Row," . $i . ",Column," . $j . ",End\n";
    }
    defined $opt_d && $opt_d > 2 && print "Row," . $i . ",End\n";
}

##############################################################
#                                                            #
#                Now standard output to Stdout               #
#                                                            #
##############################################################

print "$0,$version,";
print "Ignored risk factors in " . $opt_i if defined $opt_i;
print "\nSlice,[Category/Currency],[Category],\n";
print "Min/max difference";
print " beyond tolerance" unless ! $opt_b;
print ",Category 1,Risk Factor 1,Category 2,Risk Factor2,Old value,New value\n\n";

$tdidx && printarr("GLOBAL,,", $tdidx, @totaldiff);
our $rfk;
foreach my $rfk (sort keys %rfcatidx) {
    $rfcatidx{$rfk} && printarr("CATEGORY,$rfk,", $rfcatidx{$rfk}, @{$rfcatrank{$rfk}});
}
foreach $rfk (sort keys %rfcccyidx) {
    $rfcccyidx{$rfk} && printarr("CURRENCY,$rfk,", $rfcccyidx{$rfk}, @{$rfcccyrank{$rfk}});
}
foreach $rfk (sort keys %rfcpairidx) {
    $rfcpairidx{$rfk} && printarr("CATEGORYPAIR,$rfk", $rfcpairidx{$rfk}, @{$rfcpairrank{$rfk}});
}

$opt_w && close $doutfile;

exit(0);

sub printarr {
# Print array. With option -w write min & max into deviation file.
#
# Version Date       Author         Comment
```

```
# 1.4     26/12/2018 Bernd Plumhoff Anonymized version
   my ($title, $idx, @arr) = @_;
   my $i;

   print "$title\n";
   for ($i=0; $i<$idx; $i++) {
      printf STDOUT "%7.4f,%s,\n", $arr[$i][0], $arr[$i][1];
   }
   print "\n";

   if (defined $opt_w && $idx > 0) {
      # If we have min and/or max write them into deviation file.
      # Please note that in this version of Perl (5.12.4) the standard
      # precision of the %f format is 6 digits - which coincides with
      # our definition of $epsilon above.
      $arr[$opt_m][0] ||= 0;
      $arr[$opt_m + 1][0] ||= 0;
      $arr[$opt_m + 2][0] ||= 0;
      printf $doutfile "%s,%.6f,%.6f,%u,%u,%u\n", $title, $arr[0][0],
         $arr[$idx - 1][0],$arr[$opt_m][0],$arr[$opt_m + 1][0],
         $arr[$opt_m + 2][0];
   }
}

sub init_rf_category {
# Initialize risk factor hashtable with recognition patterns
#
# Synopsis: init_rf_category(RF_Category_filename)
#
# Example:  init_rf_category("./RF_Categories.csv");
#
# Version Date       Author         Comment
# 1.3     26/12/2018 Bernd Plumhoff Anonymized version
   open (FILE, "<", $_[0]);
   while (my $rfline = <FILE>) {
      $rfline =~ s/[\r\n]+//g;
      if (substr($rfline,0,2) ne '//' and 4 < length($rfline)) {
         my @rffields = split(",", $rfline);
         my $matchpattern = $rffields[0];
         if (substr($matchpattern,0,1) eq '#') {
            $matchpattern = '^(.*)' . substr($matchpattern,1,length($matchpattern) - 1) . '\(T[0-9]+\)$';
         } else {
            $matchpattern = '^(.*)\.' . $matchpattern . '$';
         }
         $rfcatpat{$matchpattern} = $rffields[3];
      }
   }
   close (FILE);
}

sub get_rf_category {
# Return risk factor category (Super Category - SubCategory) for a recognized pattern
# or "NC-NC" if no pattern matched.
#
# Synopsis: get_rf_category(Risk_Factor)
#
# Example:  get_rf_category("USD.#SPREAD-CORP-BB(T1825)");
#           will result in "MR-CS-CORP".
#
# Version Date       Author         Comment
# 1.3     26/12/2018 Bernd Plumhoff Anonymized version
   defined $rfcat{$_[0]} && return $rfcat{$_[0]};
   my $searchpat = $_[0];
   my $success = "NC";
   $searchpat =~ s/\.\#/-/; # We have to substitute ".#" by "-" in order to match the patterns
   foreach my $pat (keys %rfcatpat) {
      # my $temp = $searchpat =~ /$pat/;
      # print $searchpat . " =~ \"" . $pat . "\" -> " . $temp . "\n";
      # Please note that if this approach is too slow we can try to apply
      # the map, study and eval commands to speed this up later.
      # See http://perldoc.perl.org/functions/study.html, for example. [Bernd 25/01/2012]
      # General hints to speed perl up: http://www.ccl4.org/~nick/P/Fast_Enough/
      # [Bernd 08/03/2012]
      $success = $rfcatpat{$pat}, last if $searchpat =~ /$pat/;
   }
   if ($success eq "NC") {
      # This is a trick because the RAI regex are sometimes comparing
      # against $ when there still is a bracket term.
      # For example: CZK.#SWAP(T30)
      # In this case we omit the bracket term and look up CZK.#SWAP.
      $searchpat = substr($searchpat, 0, index($searchpat, "("));
      foreach my $pat (keys %rfcatpat) {
         $success = $rfcatpat{$pat}, last if $searchpat =~ /$pat/;
      }
   }
   $rfcat{$_[0]} = $success;
   # next call will be quicker
   $rfcatidx{$success} = 0;
   return $success;
}
```

```
sub rankinsert {
# Ranks and inserts value into array and stores text with it.
#
# Synopsis: rankinsert(value, text, deviation_lower_bound, deviation_upper_bound,
#      tolerance, arrayindex, array[][])
#
# Example:  rankinsert($fields[$. - 1], "$ARGV[0], Row $.: $fields[0]", 2, -2, 0,
#                                       $dnolidx, @diagnotone1);
#
# Version Date       Author         Comment
# 1.1     26/12/2018 Bernd Plumhoff Anonymized version

my ($value, $text, $devlb, $devub, $tol, $idx, @arr) = @_;
my $i;
my $border = int($opt_m / 2) - 1;
if (defined $opt_b) {
    if ($value > $tol + $epsilon) {
        $value -= $tol;
        if (defined $devlb && defined $devub &&
            ($value < $devlb - $epsilon || $value > $devub + $epsilon)) {
            return ($idx, @arr);
        }
        $arr[$opt_m + 2][0]++;
    # Tolerance overshot
    } elsif ($value < -$tol - $epsilon) {
        $value += $tol;
        if (defined $devlb && defined $devub &&
            ($value < $devlb - $epsilon || $value > $devub + $epsilon)) {
            return ($idx, @arr);
        }
        $arr[$opt_m + 1][0]++;
    # Tolerance undershot
    } else {
        $arr[$opt_m][0]++;  # Total count for this slice
        return ($idx, @arr);
    }
} else {
    if ($value > $tol + $epsilon) {
        if (defined $devlb && defined $devub &&
            ($value < $devlb - $epsilon || $value > $devub + $epsilon)) {
            return ($idx, @arr);
        }
        $arr[$opt_m + 2][0]++;
    # Tolerance overshot
    } elsif ($value < -$tol - $epsilon) {
        if (defined $devlb && defined $devub &&
            ($value < $devlb - $epsilon || $value > $devub + $epsilon)) {
            return ($idx, @arr);
        }
        $arr[$opt_m + 1][0]++;
    # Tolerance undershot
    } else {
        $arr[$opt_m][0]++;  # Total count for this slice
        return ($idx, @arr);
    }
}
$arr[$opt_m][0]++;  # Total count for this slice

if ($idx >= $opt_m) {
    # @arr has been filled already. We potentially need to throw less extreme
    # values out and we might need to move (apply a different rank to) others.
    $i = $border;
    if ($value < $arr[$i][0]) {
        # We found a small outlier
        do {
            if ($i < $border) {
                $arr[$i + 1][0] = $arr[$i][0];
                $arr[$i + 1][1] = $arr[$i][1];
            }
        } while (--$i >= 0 && $value < $arr[$i][0]);
        $arr[$i + 1][0] = $value;
        $arr[$i + 1][1] = $text;
        return ($idx, @arr);
    } else {
        $i++;
        if ($value > $arr[$i][0]) {
            # We found a big outlier
            while ($i + 1 < $opt_m && $value > $arr[$i + 1][0]) {
                $arr[$i][0] = $arr[$i + 1][0];
                $arr[$i][1] = $arr[$i + 1][1];
                $i++;
            }
            $arr[$i][0] = $value;
            $arr[$i][1] = $text;
        }
    }
} else {
    # @arr is not full yet. We are sure that the new value will be inserted
    # but we might need to move (apply a different rank to) others.
```

```
    if ($idx == 0) {
        # Ok, it's the very first array member
        $arr[$idx][0] = $value;
        $arr[$idx++][1] = $text;
    } else {
        $i = $idx - 1;
        if ($value < $arr[$i][0]) {
            # We found a small outlier
            do {
                $arr[$i + 1][0] = $arr[$i][0];
                $arr[$i + 1][1] = $arr[$i][1];
            } while (--$i >= 0 && $value < $arr[$i][0]);
            $arr[$i + 1][0] = $value;
            $arr[$i + 1][1] = $text;
        } else {
            # A new max can simply be added at the end
            $i++;
            $arr[$i][0] = $value;
            $arr[$i][1] = $text;
        }
        $idx++;
    }
}
return ($idx, @arr);
}
```

Rounding Values Preserving Their Sum with *RoundToSum*

Abstract

Rounded values do not always sum up to their original total, as demonstrated in this article. How can you ensure that the sum of rounded percentages equals exactly 100%? Is it possible to guarantee that, for accounting purposes, the distribution of overhead costs precisely matches the original total? These challenges are well-known and have been studied extensively.

This article introduces a simple solution using Excel/VBA. The function presented here can round relative values (e.g., percentages) to ensure they sum to exactly 100%. It can also round absolute values (such as cost distributions) while preserving their original sum after rounding. A key parameter allows users to choose which type of error to minimize — absolute error or relative error — compared to the common half-up rounding method.

Rounding Values Preserving Their Sum

If you need to round values without changing their sum, you might need to round one or more summands to the more distant rounded value.

Percentage Example

For example, the values 11, 45, and 555, which sum to 611, do not yield a percentage total of 100.00 but rather 99.99 if rounded to two decimals places. The **bold** values in non-sum cells have been adjusted using the *RoundToSum* function:

	Values	**Percentage rounded to 2 decimals**	**Minimize absolute Error**	**Minimize relative Error**
	11	1.80	1.80	1.80
	45	7.36	**7.37**	7.36
	555	90.83	90.83	**90.84**
Sum	611	99.99	100.00	100.00

The Excel / VBA function call *RoundToSum({11,45,555},2,FALSE,1)* would result in *{1.80,7.37,90.83}*, though. Here, the percentage value *7.364975* is rounded differently to achieve a percentage sum of *100.00* and to minimize the absolute error compared to half-up rounding. By using *RoundToSum({11,45,555},2,FALSE,2)* we would have received *{1.80,7.36,90.84}*, as this would minimize the relative error.

Example with Absolute Values

The sum of the second column differs by *+2,000* from the rounded sum. The **bold** values in non-sum cells have been adjusted using the *RoundToSum* function:

	Values	Rounded to absolute 1,000	Minimize absolute Error	Minimize relative Error
	4.523	5.000	5.000	5.000
	456	0	0	0
	-78.845	-79.000	-79.000	-79.000
	-14.491	-14.000	**-15.000**	-14.000
	65.789	66.000	66.000	66.000
	129.512	130.000	**129.000**	**129.000**
	15.562	16.000	16.000	16.000
	548.555	549.000	549.000	**548.000**
	1.590	2.000	2.000	2.000
	-897	-1.000	-1.000	-1.000
	6.968	7.000	7.000	7.000
	2.987	3.000	3.000	3.000
Sum	681.709	684.000	682.000	682.000

The User-Defined VBA Function *RoundToSum*

Name

RoundToSum – Rounding values preserving their rounded sum

Synopsis

RoundToSum(vInput, [lDigits], [bAbsSum], [lErrorType])

Description

RoundToSum rounds values without altering their rounded sum. It uses the largest remainder method to minimize the error compared to the commonly used half-up rounding method.
If the error is identical for one or more values, the first value(s) encountered will be adjusted.

Note: This solution is limited to one-dimensional tables without subtotals. There is no general solution for higher-dimensional tables or tables with subtotals.

Parameters

vInput Range or array containing the unrounded input values.

lDigits Optional, default value is 2. The number of digits to round to. For example: 0 rounds to integers, 2 rounds to the nearest cent, -3 rounds to the nearest thousand.

bAbsSum Optional, default value is TRUE. TRUE rounds the values directly which you often need for accounting calculations. FALSE adjusts the percentages so they sum to exactly 100%. This is frequently used in presentations of percentage distributions.

lErrorType Optional, default value is 1. The type of error to minimize: 1 for absolute error, 2 for relative error. The absolute error you normally minimize for values you need to book in general ledgers. For statistical distributions you often minimize the relative error to avoid amendments in the tails of the distributions.

RoundToSum Program Code

```
Enum mc_Macro_Categories
  mcFinancial = 1
  mcDate_and_Time
  mcMath_and_Trig
  mcStatistical
  mcLookup_and_Reference
  mcDatabase
  mcText
  mcLogical
  mcInformation
  mcCommands
  mcCustomizing
  mcMacro_Control
  mcDDE_External
  mcUser_Defined
  mcFirst_custom_category
  mcSecond_custom_category 'and so on
End Enum 'mc_Macro_Categories

Function RoundToSum(vInput As Variant, Optional lDigits As Long = 2, Optional bAbsSum As Boolean = True, _
    Optional lErrorType As Long = 1) As Variant
'Calculate rounded summands which exactly add up to the rounded sum of unrounded summands.
'It uses the largest remainder method which minimizes the error to the original unrounded summands.
'V2.3 PB 27-Oct-2024 (C) (P) by Bernd Plumhoff
Dim b As Boolean, i As Long, j As Long, k As Long, n As Long, lCount As Long, lSgn As Long
Dim d As Double, dDiff As Double, dRoundedSum As Double, dSumAbs As Double: Dim vA As Variant
With Application.WorksheetFunction
vA = .Transpose(.Transpose(vInput)): On Error GoTo Errhdl: i = vA(1) 'Force error in case of vertical arrays
On Error GoTo 0: n = UBound(vA): ReDim vC(1 To n) As Variant, vD(1 To n) As Variant: dSumAbs = .Sum(vA)
For i = 1 To n
  d = IIf(bAbsSum, vA(i), vA(i) / dSumAbs * 100#): vC(i) = .Round(d, lDigits)
  If lErrorType = 1 Then 'Absolute error
    vD(i) = vC(i) - d
  ElseIf lErrorType = 2 Then 'Relative error
    vD(i) = (vC(i) - d) * d
  Else
    RoundToSum = CVErr(xlErrValue): Exit Function
  End If
Next i
dRoundedSum = .Round(IIf(bAbsSum, dSumAbs, 100#), lDigits)
dDiff = .Round(dRoundedSum - .Sum(vC), lDigits)
If dDiff <> 0# Then
  lSgn = Sgn(dDiff): lCount = .Round(Abs(dDiff) * 10 ^ lDigits, 0)
  'Now find highest (lowest) lCount indices in vD
  ReDim m(1 To lCount) As Long
  For i = 1 To lCount: m(i) = i: Next i
  For i = 1 To lCount - 1
    For j = i + 1 To lCount
      If lSgn * vD(m(i)) > lSgn * vD(m(j)) Then k = m(i): m(i) = m(j): m(j) = k
    Next j
  Next i
  For i = lCount + 1 To n
    If lSgn * vD(i) < lSgn * vD(m(lCount)) Then
      j = lCount - 1
      Do While j > 0
        If lSgn * vD(i) >= lSgn * vD(m(j)) Then Exit Do
        j = j - 1
      Loop
      For k = lCount To j + 2 Step -1: m(k) = m(k - 1): Next k: m(j + 1) = i
    End If
  Next i
  For i = 1 To lCount: vC(m(i)) = .Round(vC(m(i)) + dDiff / lCount, lDigits): Next i
End If
If b Then vC = .Transpose(vC)
RoundToSum = vC
Exit Function
Errhdl:
'Transpose variants to be able to address them with vA(i), not vA(i,1)
b = True: vA = .Transpose(vA): Resume Next
End With
End Function

Sub DescribeFunction_RoundToSum()
'Run this only once, then you will see this description in the function menu
Dim FuncName As String, FuncDesc As String, Category As String, ArgDesc(1 To 4) As String
FuncName = "RoundToSum"
FuncDesc = "Rounding values preserving their rounded sum"
Category = mcMath_and_Trig
```

```
ArgDesc(1) = "Range or array which contains unrounded values"
ArgDesc(2) = "[Optional = 2] Number of digits to round to. For example: 0 rounds to integers, 2 rounds to the cent, -3 will
use thousands"
ArgDesc(3) = "[Optional = True] True takes the summands as they are; False works on the summands' percentages to make all
percentages add up to 100% exactly"
ArgDesc(4) = "[Optional = 1] Error type: 1= absolute error, 2 = relative error"
Application.MacroOptions _
  Macro:=FuncName, _
  Description:=FuncDesc, _
  Category:=Category, _
  ArgumentDescriptions:=ArgDesc
End Sub
```

Round2Sum Lambda Expression

With three Lambda expressions we can replace the VBA function *RandToSum* by this *Round2Sum* Lambda expression:

```
=LAMBDA(vI,lD,bA,lE,
    LET(
        i,IF(bA,vI,vI/SUM(vI)%),
        r,ROUND(i,lD),
        _C,ROUND(SUM(i),lD)-SUM(r),
        _E,CHOOSE(lE,r-i,(r-i)*i),
        _R,UniqRank(_E,IF(_C>0,1,0)),
        _D,IF(_R<=ROUND(ABS(_C*10^lD),0),SGN(_C)*10^-lD,0),
        r+IF(ROWS(r)=1,TRANSPOSE(_D),_D)
    )
)
```

UniqRank is defined as:

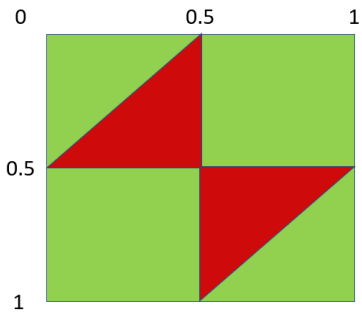
```
=LAMBDA(Ref,[Order],
    LET(
        _ord,IF(ISOMITTED(Order),-1,IF(Order=0,-1,1)),
        _r,INDEX(IF(ROWS(Ref)=1,TRANSPOSE(Ref),Ref),,1),
        _c,ROWS(_r),
        _i,SEQUENCE(ROWS(_r)),
        INDEX(SORT(HSTACK2(_i,INDEX(SORT(HSTACK2(_r,_i),,_ord),,2)),2,1),,1)
    )
 )
```

And – since Excel's worksheet function HSTACK only accepts ranges, not arrays – *HSTACK2* as:

```
=LAMBDA(a,b,
    MAKEARRAY(
        ROWS(a),
        2,
        LAMBDA(r,c,
            IF(c=1,INDEX(a,r),INDEX(b,r))
        )
    )
)
```

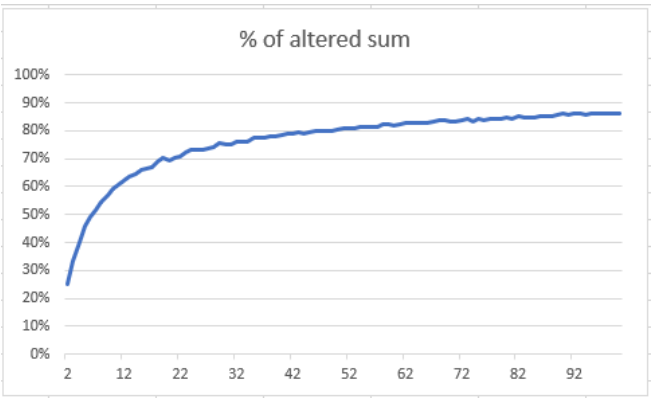
Rounding Values Alters Their Sum

How likely is it that a sum of rounded values is not identical to their rounded sum?



For two random floating point numbers this is obvious: The likelihood is around 25% - that is the percentage of **red** in this picture:

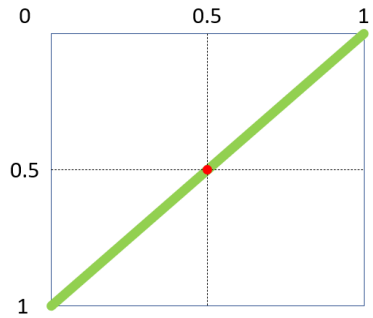
But it might be somewhat surprising that the likelihood approaches 90% if you round and add more and more numbers:



With seven floating point numbers the likelihood is already larger than 50% that the sum of rounded values is not equal to their rounded sum.

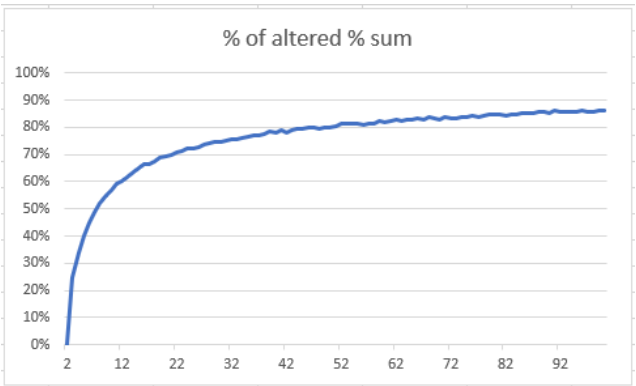
Rounded Percentages

Rounded percentages also often fail to add up to 100%.



With two random numbers the issue arises only if both numbers equal 0.5:

But with more random numbers it is similar to the problem stated initially, just with around one number more. Rounded percentages of three arbitrary numbers fail to add up to 1 with a chance of around 25%:



Monte Carlo Code

```
Const n = 100
Const runs = 20000
Const bOnlyPositive = True 'Without loss of generality

Sub monte_carlo_add_rounded_values()
'Calculates for 2 to n how likely it is
'that rounding would not alter their sum.
'Example: for 2 numbers there is a 25% chance
'that the sum of their rounded values is not
'equal to their rounded sum.
'(C) (P) by Bernd Plumhoff 16-Dec-2023 PB V0.3
Dim i                 As Long
Dim j                 As Long
Dim k                 As Long
Dim m                 As Long
Dim d                 As Double
Dim s1                As Double
Dim s2                As Double

With Application.WorksheetFunction
Randomize
For i = 2 To n
  m = 0
  For j = 1 To runs
    s1 = 0#
    s2 = 0#
    For k = 1 To i
      If bOnlyPositive Then
        d = Rnd()
      Else
        d = 2# * Rnd() - 1#
      End If
      s1 = s1 + d
      s2 = s2 + .Round(d, 0)
    Next k
    s1 = .Round(s1, 0)
    If s1 <> s2 Then
      m = m + 1
    End If
  Next j
  Cells(i, 1) = i
  Cells(i, 2) = m / runs
Next i
End With
End Sub

Sub monte_carlo_percentage_sum_of_rounded_values()
'Calculates for 2 to n how likely it is that
'rounding would not alter their percentage sum.
'Example: for 2 numbers there is a 25% chance
'that the sum of their rounded values is not
'equal to their rounded sum.
'(C) (P) by Bernd Plumhoff 16-Dec-2023 PB V0.2
Dim i                 As Long
Dim j                 As Long
Dim k                 As Long
Dim m                 As Long
Dim s1                As Double
Dim s2                As Double

With Application.WorksheetFunction
Randomize
For i = 2 To n
  m = 0
  ReDim e (1 To i) As Double
  For j = 1 To runs
    s1 = 0#
    For k = 1 To i
      If bOnlyPositive Then
        e(k) = Rnd()
      Else
        e(k) = 2# * Rnd() - 1#
      End If
      s1 = s1 + eFehler! Textmarke nicht definiert.(k)
    Next k
    s2 = 0#
    For k = 1 To i
      e(k) = .Round(1000# * e(k) / s1, 0)
      s2 = s2 + (k)
    Next k
    If s2 <> 1000# Then
      m = m + 1
    End If
  Next j
  Cells(i, 1) = i
  Cells(i, 2) = m / runs
Next i
End With
End Sub
```

Usage Examples of *RoundToSum*

Allocation of Overheads

When allocating overhead costs to products you often encounter the fact that the resulting sum of allocated overheads does not equal the original cost sum. Due to rounding differences you frequently face a little cent difference. In this case the user defined function *RoundToSum* can help.

A Real-Life Example

We present an allocation of overheads where all individual cent values accurately add up to their intermediate or final sums.

First you define how the overheads have to be allocated to support cost centres (sheet 'Keys'):

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	**Phase 1 - Allocation of overhead costs to all cost centers**															
2			Overhead Cost Centers								Secondary Cost Centers			Primary Cost Centers		
3	Key	Total	Management	Secretariat	Accounting	Controlling	HR	Marketing	Trainees	Workers Cou	Factory 1	Factory 2	Car Park	Production1	Production2	Production3
4	per Head	102	1	1	3	1	2	3	3	1	12	10		20	20	25
5	sqm	2685	50	40	100	30	50	50		15	250	350	100	500	550	600
6	uniform	14	1	1	1	1	1	1	1	1	1	1	1	1	1	1
7	Weighted	16	2	1	1	1	1	2	1	1	1	1	1	1	1	1

Read_me | Keys | 1_Allocation | 2_Allocation

The first allocation of overheads uses a rounding correction so that all summands accurately sum up on support cost centre level (sheet '1_Allocation'):

D13 =RoundToSum($C13/Keys!$B$8*Keys!C$8:P$8)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	**Allocation of overhead costs to all cost centers**																	
2																		
3				Overhead Cost Centers								Secondary Cost Centers			Primary Cost Centers			Total
4	Cost Category	Key	Overhead Costs	Management	Secretariat	Accounting	Controlling	HR	Marketing	Trainees	Workers Cou	Factory 1	Factory 2	Car Park	Production1	Production2	Production3	
5	Travel expenses	Weighted	10.000,00	1.250,00	625,00	625,00	625,00	625,00	1.250,00	625,00	625,00	625,00	625,00	625,00	625,00	625,00	625,00	10.000,00
6	Supervisory board	uniform	3.000,00	375,00	187,50	187,50	187,50	187,50	375,00	187,50	187,50	187,50	187,50	187,50	187,50	187,50	187,50	3.000,00
7	Hospitality	Weighted	2.000,00	250,00	125,00	125,00	125,00	125,00	250,00	125,00	125,00	125,00	125,00	125,00	125,00	125,00	125,00	2.000,00
8	Presents	Weighted	1.000,00	125,00	62,50	62,50	62,50	62,50	125,00	62,50	62,50	62,50	62,50	62,50	62,50	62,50	62,50	1.000,00
9	Fees, charges	uniform	500,00	62,50	31,25	31,25	31,25	31,25	62,50	31,25	31,25	31,25	31,25	31,25	31,25	31,25	31,25	500,00
10	Car costs	Weighted	4.500,00	562,50	281,25	281,25	281,25	281,25	562,50	281,25	281,25	281,25	281,25	281,25	281,25	281,25	281,25	4.500,00
11	Hardware, equipment	Weighted	200,00	25,00	12,50	12,50	12,50	12,50	25,00	12,50	12,50	12,50	12,50	12,50	12,50	12,50	12,50	200,00
12	External audit	uniform	10.000,00	1.250,00	625,00	625,00	625,00	625,00	1.250,00	625,00	625,00	625,00	625,00	625,00	625,00	625,00	625,00	10.000,00
13	Other operating expenses	sqm	5.500,00	687,50	343,75	343,75	343,75	343,75	687,50	343,75	343,75	343,75	343,75	343,75	343,75	343,75	343,75	5.500,00
14	Energy costs	sqm	6.000,00	750,00	375,00	375,00	375,00	375,00	750,00	375,00	375,00	375,00	375,00	375,00	375,00	375,00	375,00	6.000,00
15	Insurances	per Head	5.000,00	625,00	312,50	312,50	312,50	312,50	625,00	312,50	312,50	312,50	312,50	312,50	312,50	312,50	312,50	5.000,00
16	Legal costs	Weighted	5.000,00	625,00	312,50	312,50	312,50	312,50	625,00	312,50	312,50	312,50	312,50	312,50	312,50	312,50	312,50	5.000,00
17	Accounting costs	Weighted	1.000,00	125,00	62,50	62,50	62,50	62,50	125,00	62,50	62,50	62,50	62,50	62,50	62,50	62,50	62,50	1.000,00
18	Stationary	Weighted	2.000,00	250,00	125,00	125,00	125,00	125,00	250,00	125,00	125,00	125,00	125,00	125,00	125,00	125,00	125,00	2.000,00
19	Telecommunication	Weighted	2.500,00	312,50	156,25	156,25	156,25	156,25	312,50	156,25	156,25	156,25	156,25	156,25	156,25	156,25	156,25	2.500,00
20	Shipping and mailing costs	Weighted	2.000,00	250,00	125,00	125,00	125,00	125,00	250,00	125,00	125,00	125,00	125,00	125,00	125,00	125,00	125,00	2.000,00
21	Books, magazines	Weighted	1.000,00	125,00	62,50	62,50	62,50	62,50	125,00	62,50	62,50	62,50	62,50	62,50	62,50	62,50	62,50	1.000,00
22	Money transfer fees	Weighted	500,00	62,50	31,25	31,25	31,25	31,25	62,50	31,25	31,25	31,25	31,25	31,25	31,25	31,25	31,25	500,00
23	Damages, compensation	uniform	250,00	31,24	15,62	15,62	15,62	15,62	31,25	15,62	15,63	15,63	15,63	15,63	15,63	15,63	15,63	250,00
24	Working clothes	Weighted	1.500,00	187,50	93,75	93,75	93,75	93,75	187,50	93,75	93,75	93,75	93,75	93,75	93,75	93,75	93,75	1.500,00
25	Handicapped fee	uniform	2.000,00	250,00	125,00	125,00	125,00	125,00	250,00	125,00	125,00	125,00	125,00	125,00	125,00	125,00	125,00	2.000,00
26	Training and further educatior	Weighted	2.000,00	250,00	125,00	125,00	125,00	125,00	250,00	125,00	125,00	125,00	125,00	125,00	125,00	125,00	125,00	2.000,00
27	Other operating expenses	Weighted	1.500,00	187,50	93,75	93,75	93,75	93,75	187,50	93,75	93,75	93,75	93,75	93,75	93,75	93,75	93,75	1.500,00
28	Total		68.950,00	8.618,74	4.309,37	4.309,37	4.309,37	4.309,37	8.618,75	4.309,37	4.309,38	4.309,38	4.309,38	4.309,38	4.309,38	4.309,38	4.309,38	68.950,00

Worksheet Formulas		
Range	Formula	
D5:D27	**D5**	=RoundToSum($C5/Keys!$B$7*Keys!C$7:P$7)
C28:Q28	**C28**	=SUM(C5:C27)
R5:R28	**R5**	=SUM(D5:Q5)
R30	**R30**	=R28-C28

The second allocation of overheads (sheet 'Keys') also uses a rounding correction so that all support cost centres get accurately distributed to products:

	A	B	C	D	E	F	G	H
10	**Phase 2 - Allocation of Overhead Cost Centers and Secondary Cost Centers to Primary Cost Centers**							
11								
12								
13	Secondary Cost Center	Key	Production1	Production2	Production3	Total		
14								
15	Management	Weighted	30%	40%	30%	100%		
16	Secretariat	Weighted	40%	50%	10%	100%		
17	Accounting	Weighted	30%	13%	57%	100%		
18	Controlling	uniform	1	1	1	3		
19	HR	per Head	20	20	25	65		
20	Marketing	Weighted	30%	42%	28%	100%		
21	Trainees	uniform	1	1	1	3		
22	Workers Council	per Head	20	20	25	65		
23	Factory 1	Weighted	25%	20%	55%	100%		
24	Factory 2	Weighted	20%	20%	60%	100%		
25	Car Park	Weighted	40%	30%	30%	100%		

Read_me | **Keys** | 1_Allocation | 2_Allocation

The final result (sheet ‚2_Allocation'):

	A	B	C	D	E	F	G	H
1	**Allocation of Overhead Cost Centers and Secondary Cost Centers to Primary Cost Centers**							
2								
3	**Allocated Cost Centers**	**Direct Costs**	**Allocation Phase 1**	**Total**	**Production1**	**Production2**	**Production3**	**Total**
4								
5	Management	111.666,00	8.618,75	120.284,75	36.085,42	48.113,90	36.085,43	120.284,75
6	Secretariat	34.627,00	4.309,37	38.936,37	15.574,55	19.468,18	3.893,64	38.936,37
7	Accounting	96.834,00	4.309,37	101.143,37	30.343,01	13.148,64	57.651,72	101.143,37
8	Controlling	83.875,00	4.309,37	88.184,37	29.394,79	29.394,79	29.394,79	88.184,37
9	HR	53.765,00	4.309,37	58.074,37	17.869,04	17.869,04	22.336,29	58.074,37
10	Marketing	239.170,00	8.618,75	247.788,75	74.336,62	104.071,28	69.380,85	247.788,75
11	Trainees	147.397,00	4.309,37	151.706,37	50.568,79	50.568,79	50.568,79	151.706,37
12	Workers Council	471,00	4.309,37	4.780,37	1.470,88	1.470,88	1.838,61	4.780,37
13	Factory 1	125.225,00	4.309,38	129.534,38	32.383,59	25.906,88	71.243,91	129.534,38
14	Factory 2	2.398.512,00	4.309,38	2.402.821,38	480.564,27	480.564,28	1.441.692,83	2.402.821,38
15	Car Park	26.992,00	4.309,38	31.301,38	12.520,55	9.390,42	9.390,41	31.301,38
16	Phase 1 Allocation				4.309,38	4.309,38	4.309,38	12.928,14
17	Phase 2 Allocation	3.318.534,00	56.021,86	3.374.555,86	781.111,51	799.967,08	1.793.477,27	3.374.555,86
18	Directs Costs				738.060,00	854.000,00	650.360,00	2.242.420,00
19	Total Primary Cost Centers				1.523.480,89	1.658.276,46	2.448.146,65	5.629.904,00
20								
21	Overhead rate				106,4%	94,2%	276,4%	151,1%
22								
23							Check	0,00

Worksheet Formulas		
Range	Formula	
C5:C15	**C5**	=TRANSPOSE('1_Allocation'!D28:N28)
D5:D15	**D5**	=SUM(B5:C5)
E5:E15	**E5**	=RoundToSum($D5/Keys!$F15*Keys!C15:E15)
H5:H16	**H5**	=SUM(E5:G5)
E16	**E16**	='1_Allocation'!O28
B17:H17	**B17**	=SUM(B5:B15)
E19:H19	**E19**	=SUM(E16:E18)
E21:H21	**E21**	=(E17+E16)/E18
H23	**H23**	=H19-SUM(E18:G18)-SUM(B5:B15)-SUM('1_Allocation'!C5:C27)

This correct allocation of overheads you will be able to enter into a general ledger without any cent / penny difference.

Example of an Exact Relation of Random Numbers

It is fairly easy to create a loaded die, let us say on average the 6 should appear twice as often as all the other numbers 1 thru 5: Enter into A1: *=MIN(INT(RAND()*7+1),6)*

But what if you want to create 7 rolls of this die and all numbers between 1 and 5 should appear exactly once and 6 exactly twice?

Here is a general solution:

	A	B	C	D	E	F	G	H	I	J	K	L
1						**Just statistical likelihood**					**Total**	
2	**Color**	**Likelihood**	**Pos / Iteration**	**One**	**Two**	**Three**	**Four**	**Five**	**Six**	**Green**	**Yellow**	**Red**
3	**Green**	50,00%	**1**	Green	Yellow	Green	Red	Green	Yellow	3	2	1
4	**Yellow**	33,33%	**2**	Yellow	Yellow	Yellow	Green	Green	Red	2	3	1
5	**Red**	16,67%	**3**	Yellow	Green	Red	Red	Green	Yellow	2	2	2
6			**4**	Green	Green	Yellow	Green	Red	Green	4	1	1
7			**5**	Green	Green	Red	Yellow	Yellow	Yellow	2	3	1
8			**6**	Yellow	Yellow	Yellow	Yellow	Green	Yellow	1	5	0
9			**7**	Green	Red	Green	Green	Yellow	Yellow	3	2	1
10			**8**	Green	Green	Green	Yellow	Green	Red	4	1	1
11			**9**	Yellow	Green	Green	Yellow	Green	Green	4	2	0
12			**10**	Green	Red	Yellow	Green	Red	Green	3	1	2
13									**Total:**	28	22	10
14								**Should stochastically be:**		30	20	10
16						**Exact likelihood**					**Total**	
17			**Pos / Iteration**	**One**	**Two**	**Three**	**Four**	**Five**	**Six**	**Green**	**Yellow**	**Red**
18			**1**	Green	Green	Red	Green	Yellow	Yellow	3	2	1
19			**2**	Green	Yellow	Red	Yellow	Green	Green	3	2	1
20			**3**	Yellow	Green	Green	Red	Green	Yellow	3	2	1
21			**4**	Green	Yellow	Green	Green	Red	Yellow	3	2	1
22			**5**	Green	Yellow	Green	Red	Green	Yellow	3	2	1
23			**6**	Green	Green	Green	Yellow	Red	Yellow	3	2	1
24			**7**	Yellow	Green	Red	Yellow	Green	Green	3	2	1
25			**8**	Green	Yellow	Green	Yellow	Red	Green	3	2	1
26			**9**	Yellow	Yellow	Green	Green	Green	Red	3	2	1
27			**10**	Green	Yellow	Green	Yellow	Red	Green	3	2	1
28									**Total:**	30	20	10
29								**Should stochastically be:**		30	20	10

Worksheet Formulas		
Range	Formula	
D3:I12	**D3**	=INDEX($A$3:$A$5,INT(sbRandHistogrm(1,4,$B$3:$B$5)))
J3:L12;J18:L27	**J3**	=COUNTIF($D3:$I3,J$2)
J13:L13;J28:L28	**J13**	=SUM(J3:J12)
J14;J29	**J14**	=COUNTA($D$3:$I$12)*TRANSPOSE($B$3:$B$5)
D18:D27	**D18**	=INDEX($A$3:$A$5,INT(sbExactRandHistogrm(6,1,4,$B$3:$B$5)))

The User-Defined VBA Function sbExactRandHistogrm

Name

sbExactRandHistogrm – Create an exact double histogram distribution.

Synopsis

sbExactRandHistogrm(ldraw, dmin, dmax, vWeight)

Description

sbExactRandHistogrm creates an exact histogram distribution for ldraw draws of floating point numbers with double precision within range dmin:dmax. This range is divided into *vWeight.count* classes. Each class has weight *vWeight(i)*, reflecting the probability of occurrence of a value within the class. If weights can't be achieved exactly for *ldraw* draws the largest remainder method will be applied to minimize the absolute error. This function calls *RoundToSum*.

Parameters

ldraw Number of draws

dmin Minimum = lower boundary of range of numbers to draw

dmax Maximum = upper boundary of range of numbers to draw

vWeight Array of weights. Array size determines the number of different classes the range *dmin : dmax* is divided into. Values in this array specify likelihood of this class' numbers to appear (be drawn).

sbExactRandHistogrm Program Code

```
Function sbExactRandHistogrm(ldraw As Long, _
            dmin As Double, _
            dmax As Double, _
            vWeight As Variant) As Variant
'Creates an exact histogram distribution for ldraw draws within range dmin:dmax.
'This range is divided into vWeight.count classes. Each class has weight vWeight(i)
'reflecting the probability of occurrence of a value within the class.
'If weights can't be achieved exactly for ldraw draws the largest remainder method will
'be applied to minimize the absolute error. This function calls (needs) RoundToSum.
'(C) (P) by Bernd Plumhoff 01-May-2021 PB V0.9

Dim i As Long, j As Long, n As Long
Dim vW As Variant
Dim dSumWeight As Double, dR As Double

Randomize
With Application.WorksheetFunction
vW = .Transpose(vWeight)
On Error GoTo Errhdl
i = vW(1) 'Throw error in case of horizontal array
On Error GoTo 0

n = UBound(vW)
ReDim dWeight(1 To n) As Double
ReDim dSumWeightI(0 To n) As Double
ReDim vR(1 To ldraw) As Variant

For i = 1 To n
    If vW(i) < 0# Then 'A negative weight is an error
        sbExactRandHistogrm = CVErr(xlErrValue)
        Exit Function
    End If
    'Calculate sum of all weights
    dSumWeight = dSumWeight + vW(i)
Next i

If dSumWeight = 0# Then
    'Sum of weights has to be greater zero
    sbExactRandHistogrm = CVErr(xlErrValue)
    Exit Function
End If

For i = 1 To n
    'Align weights to number of draws
    dWeight(i) = CDbl(ldraw) * vW(i) / dSumWeight
Next i

vW = RoundToSum(dWeight, 0)
On Error GoTo Errhdl
i = vW(1) 'Throw error in case of horizontal array
On Error GoTo 0

For j = 1 To ldraw

    dSumWeight = 0#
    dSumWeightI(0) = 0#
    For i = 1 To n
        'Calculate sum of all weights
        dSumWeight = dSumWeight + vW(i)
        'Calculate sum of weights till i
        dSumWeightI(i) = dSumWeight
    Next i

    dR = dSumWeight * Rnd

    i = n
    Do While dR < dSumWeightI(i)
        i = i - 1
    Loop

    vR(j) = dmin + (dmax - dmin) * (CDbl(i) + _
            (dR - dSumWeightI(i)) / vW(i + 1)) / CDbl(n)
    vW(i + 1) = vW(i + 1) - 1#

Next j

sbExactRandHistogrm = vR

Exit Function

Errhdl:
'Transpose variants to be able to address
'them with vW(i), not vW(i,1)
vW = .Transpose(vW)
Resume Next
End With

End Function
```

Fair Staff Selection Based on Team Size – sbFairStaffSelection

Let us assume your company needs to get some special tasks done. All staff members can do the work. You want the teams to second their staff based on the size of each team. This selection can be done by the user-defined function *RoundToSum*.

Since we cannot guarantee that each team can provide staff exactly in relation to its staff number for each special task, we need to call *RoundToSum* including a lookback onto previous staff selections.

RoundToSum uses the largest remainder method (also called Hare-Niemeyer) which can suffer from the Alabama paradoxon. If the total number of staff to be selected increases it can happen that a team needs to provide less staff than before. Because we cannot account for this in hindsight, this paradoxon needs to be dealt with as soon as it occurs.

Example

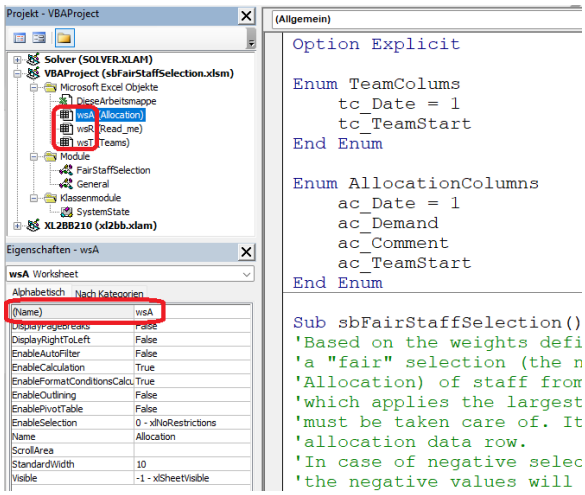
On 1-Jan-2023 these teams exist (sheet 'Teams', codename (its internal VBA name) 'wsT'):

	A	B	C	D	E
1	**Date**	**Team A**	**Team B**	**Team C**	**Team D**
2	01.01.2023	5670	3850	420	60

Over the following three months these staff numbers are required for special tasks and are selected (sheet 'Allocation', codename (its internal VBA name) 'wsA'):

	A	B	C	D	E	F	G
	Calculate Allocation						
1	**Date**	**Demand**	**Comment**	**Team A**	**Team B**	**Team C**	**Team D**
2	01.01.2023	323		183	124	14	2
3	01.02.2023	1	Recalc 11.03.2023 10:52:24. Allocation for 1 amended to 0. Allocation for 3 set to 0.	0	1	0	0
4	01.03.2023	9676	Recalc 11.03.2023 10:52:24.	5487	3725	406	58

On 1-Feb-2023 the largest remainder method would have selected a total number of 184, 125, 13, and 2 employees of teams A, B, C, and D ausgewählt. But on 1-Jan-2023 team C had already provided 14 members of staff which cannot be taken back. This means that team A or team B needs to provide one employee less. The implemented algorithm looks left to right to account for this, so in this case team A is impacted. On 1-Mar-2023 all remaining staff counts of all teams are requested. The algorithm selects for each team exactly its staff count in total because the lookback includes all request data records.



Note: The codename (the internal VBA name – here wsA.Codename) of a worksheet can be referred to directly from VBA. It might differ from the sheet name the Excel user sees and which can be read or written via Sheets("Your_Sheetname") (here wsA.Name). Unfortunately the codename of a worksheet is read-only and you can only manually change it, not via VBA.

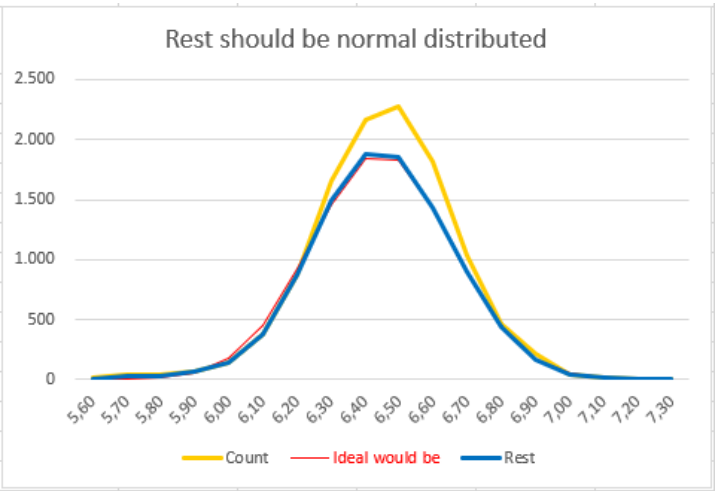
sbFairStaffSelection Program Code

```
Enum TeamColums
    tc_Date = 1
    tc_TeamStart
End Enum

Enum AllocationColumns
    ac_Date = 1
    ac_Demand
    ac_Comment
    ac_TeamStart
End Enum

Sub sbFairStaffSelection()
'Based on the weights defined in tab Teams this program allocates
'a "fair" selection (the number given in column Demand of tab
'Allocation) of staff from these teams. This program uses (calls) RoundToSum
'which applies the largest remainder method, so the Alabama paradoxon
'must be taken care of. It also applies a lookback up to the topmost
'allocation data row.
'In case of negative selection counts (i. e. the Alabama paradoxon)
'the negative values will be set to zero and the necessary amendments
'(reductions) will be applied from left to right. Please order your
'teams with ascending sizes or descending sizes to account for this.
'(C) (P) by Bernd Plumhoff 09-Mar-2023 PB V0.1

Dim bLookBack                As Boolean
Dim bReCalc                  As Boolean

Dim i                        As Long
Dim j                        As Long
Dim k                        As Long
Dim m                        As Long
Dim lAmend                   As Long
Dim lCellResult              As Long
Dim lDemand                  As Long
Dim lRowSum                  As Long
Dim lSum                     As Long
Dim lTotal                   As Long 'Most recent total number of staff in all teams

Dim sComment                 As String

Dim vAlloc                   As Variant
Dim vTeams                   As Variant

Dim state                    As SystemState

Set state = New SystemState

With Application.WorksheetFunction

vTeams = .Transpose(.Transpose(Range(wsT.Cells(1, 1).End(xlDown).Offset(0, tc_TeamStart - 1), _
          wsT.Cells(1, 1).End(xlDown).End(xlToRight))))
j = UBound(vTeams)
ReDim dAlloc(1 To j) As Double
lTotal = .Sum(vTeams)

bReCalc = False
i = 2
lDemand = wsA.Cells(i, ac_Demand)
Do While lDemand > 0

    lRowSum = .Sum(Range(wsA.Cells(i, ac_TeamStart), wsA.Cells(i, ac_TeamStart + j)))

    If lDemand <> lRowSum Then bReCalc = True

    If bReCalc Or wsA.Cells(i + 1, ac_Demand) = 0 Then

        sComment = "Recalc " & Format(Now(), "DD.MM.YYYY HH:nn:ss") & ". "
        bLookBack = False
        k = i - 1
        If k > 1 Then
            bLookBack = True
            lDemand = 0
            lSum = 0
            ReDim lTeamSum(1 To j) As Long
            Do While k > 1
                lSum = lSum + wsA.Cells(k, ac_Demand)
                lDemand = wsA.Cells(i, ac_Demand) + lSum
                For m = 1 To j
                    lTeamSum(m) = lTeamSum(m) + wsA.Cells(k, m + ac_TeamStart - 1)
                Next m
                'If lSum >= lTotal Then Exit Do 'Uncomment if lookback should be restricted
                                               'to total staff number
                k = k - 1
            Loop
        End If

        For m = 1 To j
            dAlloc(m) = lDemand * vTeams(m) / lTotal
        Next m

        vAlloc = RoundToSum(vInput:=dAlloc, lDigits:=0)

        If bLookBack Then
            For m = 1 To j
                lCellResult = vAlloc(m) - lTeamSum(m)
                If lCellResult < 0 Then
                    'The Alabama Paradoxon: we have to reduce other parties'
                    'allocations because we cannot have negative allocations
                    lAmend = lAmend - lCellResult
```

```
                End If
                vAlloc(m) = lCellResult
            Next m
            If lAmend > 0 Then
                For m = 1 To j
                    lCellResult = vAlloc(m)
                    If lCellResult < 0 Then
                        vAlloc(m) = 0
                        sComment = sComment & "Allocation for " & m & " set to 0. "
                    ElseIf lCellResult > 0 And lAmend > 0 Then
                        If lCellResult > lAmend Then
                            vAlloc(m) = lCellResult - lAmend
                            lAmend = 0
                        Else
                            vAlloc(m) = 0
                            lAmend = lAmend - lCellResult
                        End If
                        sComment = sComment & "Allocation for " & m & " amended to " & _
                                   vAlloc(m) & ". "
                    End If
                Next m
            End If
        End If
        wsA.Cells(i, ac_Comment) = sComment
        For m = 1 To j
            wsA.Cells(i, ac_TeamStart + m - 1) = vAlloc(m)
        Next m

    End If

    i = i + 1
    lDemand = wsA.Cells(i, ac_Demand)

Loop

Range(wsT.Cells(1, tc_TeamStart), wsT.Cells(1, 250)).Copy Destination:=wsA.Cells(1, ac_TeamStart)

End With

End Sub
```

Distribute a Sample Normally

You have 11,256 Christmas trees. A customer wants to buy 1,500 of them from you. The only condition: the average height should be 6.50 meters. You would now like the remaining quantity of your Christmas trees to be as normally distributed as possible:



How can you achieve this?

A Calculation Example

	A	B	C	D	E	F	G	H
1				Withdrawal:		1500		
2				Average Length:		6,5		
3	Length	Count	Ideal would be	Auto-Withdrawal	Temp Result	Withdrawal	Rest	Ideal would be
4	5,60	20	0	20	0	10	10	0
5	5,70	40	3	37	3	15	25	3
6	5,80	40	14	26	14	8	32	14
7	5,90	72	59	16	56	8	64	56
8	6,00	148	192	0	148	0	148	179
9	6,10	372	497	0	372	0	372	456
10	6,20	876	1.016	0	876	0	876	918
11	6,30	1.660	1.644	200	1.460	165	1.495	1.460
12	6,40	2.160	2.102	323	1.837	281	1.879	1.837
13	6,50	2.276	2.125	449	1.827	416	1.860	1.827
14	6,60	1.820	1.698	384	1.436	383	1.437	1.436
15	6,70	1.036	1.073	143	893	143	893	893
16	6,80	464	536	25	439	28	436	439
17	6,90	212	212	41	171	43	169	171
18	7,00	48	66	0	48	0	48	52
19	7,10	12	16	0	12	0	12	13
20	7,20	0	3	0	0	0	0	2
21	7,30	0	0	0	0	0	0	0
22	Total	11.256	11.256	1.664	9.592	1.500	9.756	9.756
23								
24	AVERAGE	6,45	6,45	6,47	6,45	6,50	6,45	6,45
25	STDEV.P	0,21	0,21		0,20		0,21	0,21
26	SKEW.P	-0,35	-0,00		-0,01		-0,20	-0,00
27	KURT	0,95	-0,02		0,03		0,53	-0,02

Worksheet Formulas		
Range	Formula	
C4	**C4**	=RoundToSUm(NORM.DIST(A4:A21,B24,B25,FALSE)*B22/10,0,,2)
D4	**D4**	=IF(B4:B21-H4:H21<0,0,B4:B21-H4:H21)
E4	**E4**	=B4:B21-D4:D21
F4:F21	**F8**	=D8
G4	**G4**	=B4:B21-F4:F21
H4	**H4**	=RoundToSUm(NORM.DIST(A4:A21,(B24*B22-F2*F1)/(B22-F1),B25,FALSE)*(B22-F1)/10,0,,2)
B22:H22	**B22**	=SUM(B4:B21)
B24:H24	**B24**	=sbSWV($A24,$A$4:$A$21,B$4:B$21)
B25:C27;E25:E27;G25:H27	**B25**	=sbSWV($A25,$A$4:$A$21,B$4:B$21)

Assume you have the quantity of trees shown above with the specified lengths.
An initial interesting calculation is certainly to determine how normally distributed your starting sample is. We calculate the skewness using the function sbSWV: it is approximately *=sbSWV("SKEW.P", $A$4:$A$21, B$4:B$21) = -0.35*. The excess (a measure of the kurtosis) is approximately *=sbSWV("KURT", $A$4:$A$21, B$4:B$21) = 0.95*. As we can see from the yellow-orange curve in the chart above, this initial sample is already "somewhat" normally distributed.

Ideally, however, this sample would be distributed as shown in column C with the formula =TRANSPOSE(RoundToSum(NORM.DIST(A4:A21,B24,B25,FALSE)*B22/10,0)): the skewness and excess would both be zero (the rounding performed leads to slight deviations).
Column H shows the ideal distribution of the remaining quantities after removal.

With the automatic sampling shown in column D, we try to match this ideal remaining quantity as closely as possible. Of course, this can only succeed if we have enough trees of the respective lengths available. Where this is not the case, we cannot add any trees and must enter 0 as the removal quantity – for example, in the chart, you can see that the ideal distribution at a length of 6.10 m is higher than the actual remaining distribution.
The original formulas in column F should be =D4 to =D21.

We now overwrite these values with manual inputs to:

- Achieve a total removal of exactly 1,500 trees,
- Achieve an average tree length of 6.5 meters,
- Achieve a standard deviation (spread) of the remaining quantity similar to that of the original quantity,
- Achieve a skewness smaller in absolute value than the original quantity,
- Achieve a kurtosis smaller in absolute value than the original quantity.

In the example file provided below, increased deviations are highlighted using conditional formatting.

Note: It is not always possible to obtain a "sufficiently" normally distributed remaining quantity. As can be seen easily, sometimes even the desired average length cannot be reached – for example, with the quantity shown above, try to get 21 trees with an average length of 5.60 meters.

Helper Functions

While Excel offers many basic statistical functions, they are not capable of processing weighted values. The custom function *sbSWV* (statistics for weighted values) used here provides an easy and quick way to display how well the samples are normally distributed.

To ensure that the sums of the integer ideal distributions of the samples match exactly with the sums of the original samples, the custom function *RoundToSum* was used. Note that the parameter 2 was chosen for the error type to minimize the relative error. This prevents artificial rounding to the "wrong" side at the outer ends of the distributions.

sbSWV Program Code

```
#Const SORTED = False

Function sbSWV(sStat As String, _
        ParamArray vInput() As Variant) As Variant
'Calculate some statistical measures of weighted values
'(C) (P) by Bernd Plumhoff 20-Aug-2024 PB V0.81
Dim d As Double, d2 As Double, dSum As Double
Dim i As Long, j As Long, k As Long, m As Long, n As Long
Dim vV, vV2, vV3, vW 'Variants

With Application.WorksheetFunction
vV = .Transpose(vInput(0))
Select Case sStat
Case "COVAR", "CORREL"
    vV2 = .Transpose(vInput(1))
    vW = .Transpose(vInput(2))
Case Else
    vW = .Transpose(vInput(1))
End Select
On Error GoTo errhdl
i = vV(1) 'Force error in case of vertical arrays
On Error GoTo 0
If UBound(vV) <> UBound(vW) Then
    'Arrays of values and of weights must have same dimension
    sbSWV = CVErr(xlErrNum)
    Exit Function
End If
Select Case UCase(sStat)
Case "AVERAGE"
    sbSWV = .SumProduct(vV, vW) / .Sum(vW)
Case "CORREL"
    vV3 = vV
    dSum = .Sum(vW)
    d = .SumProduct(vV, vW) / dSum
    d2 = .SumProduct(vV2, vW) / dSum
    For i = LBound(vV) To UBound(vV)
        vV3(i) = vW(i) * (vV(i) - d) * (vV2(i) - d2)
        vV(i) = vW(i) * (vV(i) - d) ^ 2#
        vV2(i) = vW(i) * (vV2(i) - d2) ^ 2#
    Next i
    sbSWV = .Sum(vV3) / Sqr(.Sum(vV) * .Sum(vV2))
Case "COVAR"
    dSum = .Sum(vW)
    d = .SumProduct(vV, vW) / dSum
    d2 = .SumProduct(vV2, vW) / dSum
    For i = LBound(vV) To UBound(vV)
        vV(i) = vW(i) * (vV(i) - d) * (vV2(i) - d2)
    Next i
    sbSWV = .Sum(vV) / dSum
Case "KURT"
    n = .Sum(vW)
    ReDim dV(1 To n) As Double
    k = 1
    For i = 1 To UBound(vW)
      For j = 1 To vW(i)
        dV(k) = vV(i)
        k = k + 1
      Next j
    Next i
    sbSWV = .Kurt(dV)
Case "MODE"
    k = .Max(vW)
    If k < 2 Then
        sbSWV = CVErr(xlErrNA)
        Exit Function
    End If
    sbSWV = vV(.Match(.Max(vW), vW, False))
Case "MEDIAN"
    If .Min(vW) < 1 Then
        sbSWV = CVErr(xlErrNA)
        Exit Function
    End If
    k = 0
    j = .Sum(vW)
    m = j Mod 2
    For i = LBound(vW) To UBound(vW)
        If vW(i) Mod 1 <> 0 Then
            sbSWV = CVErr(xlErrNum)
            Exit Function
        End If
        #If Not SORTED Then
            'Ensure ascending values in case input is unsorted.
            'This simple bubble sort leads to a quadratic runtime
            'but it's still quicker on 50 input values or more than
            'Lorimer Miller's nifty worksheet function approach
            '=LOOKUP(2,1/FREQUENCY(SUM(B1:B50)/2,SUMIF(A1:A50,"<="&A1:A50,B1:B50)),A1:A50)
            'BTW: Lorimer's approach is different from Excel's MEDIAN
            '(see below); and his other elegant array formula
            '=MEDIAN(IF(TRANSPOSE(ROW(A1:A1000))<=B1:B50,A1:A50))
            'calculates like Excel's MEDIAN but IMHO it's way too slow
            For n = i + 1 To UBound(vW)
                If vV(n) < vV(i) Then
                    d = vV(i)
                    vV(i) = vV(n)
                    vV(n) = d
                    d = vW(i)
                    vW(i) = vW(n)
                    vW(n) = d
                End If
            Next n
```

```
        #End If
        k = k + vW(i)
        Select Case 2 * k
        Case j + m
            If m = 0 Then
                #If Not SORTED Then
                    'Ensure vV(i + 1) is next greater value
                    For n = i + 2 To UBound(vW)
                        If vV(n) < vV(i + 1) Then
                            vV(i + 1) = vV(n)
                        End If
                    Next n
                #End If
                'Here Lorimer's function mentioned above would
                'return vV(i), the lower value
                sbSWV = (vV(i) + vV(i + 1)) / 2#
            Else
                sbSWV = vV(i)
            End If
            Exit Function
        Case Is > j + m
            sbSWV = vV(i)
            Exit Function
        End Select
    Next i
Case "SKEW.P"
    n = .Sum(vW)
    ReDim dV(1 To n) As Double
    k = 1
    For i = 1 To UBound(vW)
      For j = 1 To vW(i)
        dV(k) = vV(i)
        k = k + 1
      Next j
    Next i
    sbSWV = .Skew_p(dV)
Case "STDEV"
    dSum = .Sum(vW)
    d = .SumProduct(vV, vW) / dSum
    For i = LBound(vV) To UBound(vV)
        vV(i) = Abs(vV(i) - d) ^ 2#
    Next i
    sbSWV = Sqr(.SumProduct(vV, vW) / (dSum - 1#))
Case "STDEV.P"
    dSum = .Sum(vW)
    d = .SumProduct(vV, vW) / dSum
    For i = LBound(vV) To UBound(vV)
        vV(i) = Abs(vV(i) - d) ^ 2#
    Next i
    sbSWV = Sqr(.SumProduct(vV, vW) / dSum)
Case "VAR"
    dSum = .Sum(vW)
    d = .SumProduct(vV, vW) / dSum
    For i = LBound(vV) To UBound(vV)
        vV(i) = vW(i) * (vV(i) - d) ^ 2#
    Next i
    sbSWV = .Sum(vV) / (dSum - 1#)
Case Else
    sbSWV = CVErr(xlErrValue)
End Select
Exit Function
errhdl:
'Transpose variants to be able to address them
'with vV(i), not vV(i,1)
vV = .Transpose(vV)
vW = .Transpose(vW)
Select Case sStat
Case "COVAR", "CORREL"
    vV2 = .Transpose(vV2)
End Select
Resume Next
End With
End Function
```

Distribution of Budgets Among Remaining Staff

When staff members leave, their budgets can be redistributed among the remaining employees based on initial budget. But how can this redistribution be done accurately and fairly?

A Simple Approach

A simple formula which you can copy down from D3 to D12 is *=ROUND(C3*$B$2/$C$2,2).*

You can delete the budgets of leavers easily in column C. The order of deletions does not matter. The obvious disadvantage of this approach is a potential rounding difference, because the sum of rounded values is not necessary equal to the rounded sum of not-rounded summands. The example above shows a difference of 0.02.

	A	B	C	D
1	**Name**	**Amount**	**Deletion**	**New Amount**
2	**Sum**	**94.020,00**	**40.000,00**	**94.020,02**
3	Lehmann	49.000,00		-
4	Schulze	6.000,00	6.000,00	14.103,00
5	Schultze	5.750,00	5.750,00	13.515,38
6	Schmidt	5.500,00	5.500,00	12.927,75
7	Schmitt	5.270,00	5.250,00	12.340,13
8	Müller	5.000,00		-
9	Maier	4.750,00	4.750,00	11.164,88
10	Mayer	4.500,00	4.500,00	10.577,25
11	Meier	4.250,00	4.250,00	9.989,63
12	Meyer	4.000,00	4.000,00	9.402,00

Worksheet Formulas		
Range	Formula	
B2:D2	**B2**	=SUM(B3:B12)
D3	**D3**	=ROUND(C3:C12*$B$2/$C$2,2)

A Correct Calculation

With the user defined function RoundToSum you can use the spill formula *=RoundToSum(C4:C13*$B$3/$C$3,D1)*.

RoundToSum sometime needs to round to the 'wrong' side but then it ensures a minimal error.

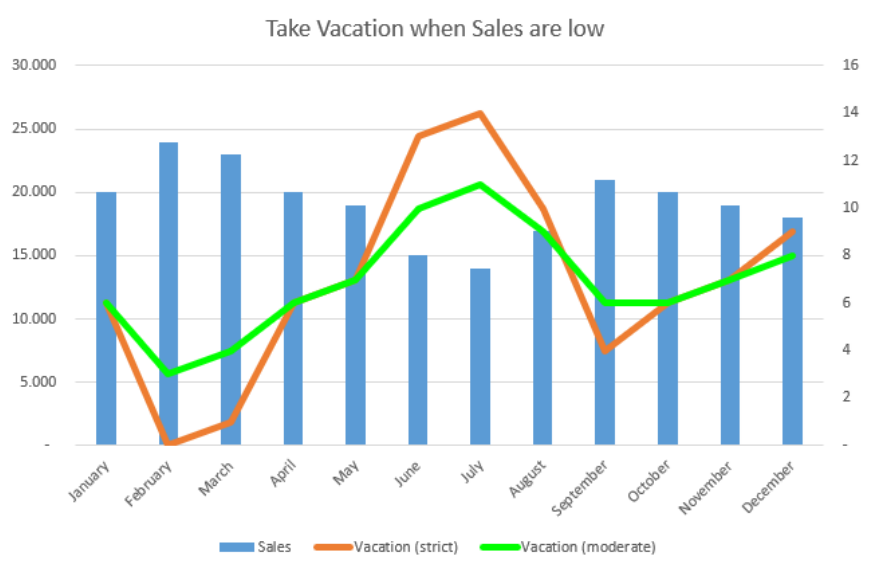
	A	B	C	D
1		Round to [digits]:		-1
2	**Name**	**Amount**	**Deletion**	**New Amount**
3	**Sum**	**94.020,00**	**40.000,00**	**94.020,00**
4	Lehmann	49.000,00		-
5	Schulze	6.000,00	6.000,00	14.100,00
6	Schultze	5.750,00	5.750,00	13.520,00
7	Schmidt	5.500,00	5.500,00	12.930,00
8	Schmitt	5.270,00	5.250,00	12.340,00
9	Müller	5.000,00		-
10	Maier	4.750,00	4.750,00	11.160,00
11	Mayer	4.500,00	4.500,00	10.580,00
12	Meier	4.250,00	4.250,00	9.990,00
13	Meyer	4.000,00	4.000,00	9.400,00

Worksheet Formulas		
Range	Formula	
B3:D3	**D3**	=SUM(D4:D13)
D4	**D4**	=RoundToSum(C4:C13*$B$3/$C$3,D1)

Take Vacation When Less is Going on

If your business fluctuates strongly seasonally, you can plan the vacation of your staff accordingly and consider hiring seasonal staff:

Note: Of course you cannot force anybody when to take a vacation and how many days are to be taken. These calculations are just meant to be suggestions of reasonable indicators.



Simple Example

If you like to take the maximum sales values (here: 24,000) as a basis, applying zero vacations to it, and scale the vacation days linearly to the other sales values:

	A	B	C	D	E	F	G
1			**Sales Limit for no vacation:**		**Higher Sales Limit for no vacation:**	Increased to allow for some vacation at sales maimum.	
2			24.000		28.000		
3		**Sales**	**Vacation (strict)**	**Integers**	**Vacation (moderate)**	**Integers**	
4	**Total**	**230.000**	83		83		
5	**January**	20.000	5,7	6	6,3	6	
6	**February**	24.000	-	-	3,1	3	
7	**March**	23.000	1,4	1	3,9	4	
8	**April**	20.000	5,7	6	6,3	6	
9	**May**	19.000	7,2	7	7,0	7	
10	**June**	15.000	12,9	13	10,2	10	
11	**July**	14.000	14,3	14	11,0	11	
12	**August**	17.000	10,0	10	8,6	9	
13	**September**	21.000	4,3	4	5,5	6	
14	**October**	20.000	5,7	6	6,3	6	
15	**November**	19.000	7,2	7	7,0	7	
16	**December**	18.000	8,6	9	7,8	8	
17		**Checksum Vacation**	**83,0**	**83**	**83,0**	**83**	

Worksheet Formulas		
Range	Formula	
C5	**C5**	=(C$2-$B5:$B16)/(C$2*12-$B$4)*C$4
D5	**D5**	=RoundToSum(C5:C16,0)
E5	**E5**	=(E$2-$B5:$B16)/(E$2*12-$B$4)*E$4
F5	**F5**	=RoundToSum(E5:E16,0)
C17:F17	**C17**	=SUM(C5:C16)

More Complex Example

If you got employees who are not present at specified months – *RoundToSum* rounds to whole vacation days in the last table:

	A	B	C	D	E	F	G	H
1			Sales Limit for no vacation:					
2			24.000					
3		Sales	Vacation days (fractional)	Vacation days (integer)	Vacation Claim			
4	Total	230.000			Andrew	Benjamin	Charlie	David
5	January	20.000	5,7	6	x	x		x
6	February	24.000	-	-	x	x		x
7	March	23.000	1,4	1	x	x	x	x
8	April	20.000	5,7	6	x	x	x	x
9	May	19.000	7,2	7	x	x	x	
10	June	15.000	12,9	13	x	x	x	
11	July	14.000	14,3	14	x	x	x	
12	August	17.000	10,0	10	x	x	x	
13	September	21.000	4,3	4	x	x	x	x
14	October	20.000	5,7	6	x	x	x	x
15	November	19.000	7,2	7	x		x	x
16	December	18.000	8,6	9	x		x	x
17		Total	83,0	83	25,0	21,0	21,0	16,0
18								
19				Vacation days (fractional)				
20				Total	Andrew	Benjamin	Charlie	David
21			January	6,1	1,8	1,9	-	2,5
22			February	-	-	-	-	-
23			March	1,3	0,3	0,3	0,3	0,4
24			April	7,8	1,8	1,9	1,6	2,5
25			May	6,2	2,1	2,2	1,9	-
26			June	11,5	3,9	4,1	3,5	-
27			July	12,4	4,2	4,4	3,8	-
28			August	8,9	3,0	3,1	2,7	-
29			September	5,2	1,2	1,3	1,1	1,6
30			October	7,8	1,8	1,9	1,6	2,5
31			November	6,9	2,1	-	1,9	2,9
32			December	8,9	2,7	-	2,5	3,7
33			Total	83,0	25,0	21,0	21,0	16,0
34								
35				Vacation days (integer)				
36				Total	Andrew	Benjamin	Charlie	David
37			January	7	2	2	-	3
38			February	-	-	-	-	-
39			March	-	-	-	-	-
40			April	8	2	2	2	2
41			May	6	2	2	2	-
42			June	11	4	4	3	-
43			July	13	4	5	4	-
44			August	9	3	3	3	-
45			September	5	1	1	1	2
46			October	8	2	2	2	2
47			November	7	2	-	2	3
48			December	9	3	-	2	4
49			Total	83	25	21	21	16

Overwrite with higher value to allow for vacation in month with max sales

Worksheet Formulas		
Range	Formula	
C5	**C5**	=($C$2-B5:B16)/($C$2*12-$B$4)*$C$17
D5	**D5**	=RoundToSum(C5:C16;0)
D21:D32,D37:D48	**D21**	=SUMME(E21:H21)
E21:H21	**E21**	=WENNFEHLER(($E$5:E$16="x")*E$17*$D$5:$D$16/SUMME(($E$5:E$16="x")*$D$5:$D$16);0)
D33:H33,D49:H49	**D33**	=SUMME(D21:D32)
E37:H37	**E37**	=WENNFEHLER(RoundToSum(E21:E32;0);0)

Assign Work Units Adjusted by Delivered Output

How can you fairly assign work units to your staff while considering the number of units they have already delivered?

Yellow cells show input values, green ones are intermediate or helper cells, and blue cells mark final output values. Note: You need to enter 'Units done' in descending order.

In this example *90.6* units have already been delivered, but *86* more units are to be assigned to *28* lecturers. A fair share for each lecturer would be *(90.6 + 86) / 28 = 6.3*, but *7* lecturers have already delivered more than that.

Column C shows the fractional results. In column D a simple worksheet function approach has been applied to round values of column C to integers, preserving their original sum.

As you can easily see, column E shows smoother results using the user defined function *RoundToSum*.

	A	B	C	D	E
1	Total units still to do	86			
2	Total	90,6	86	86	86
3	Lecturer	Units done	Helper	Units to do (Formulas)	Units to do (RoundToSum)
4	Fair share	6,307143			
5	Lecturer 1	12	0	0	0
6	Lecturer 2	11	0	0	0
7	Lecturer 3	9	0	0	0
8	Lecturer 4	8	0	0	0
9	Lecturer 5	8	0	0	0
10	Lecturer 6	7	0	0	0
11	Lecturer 7	7	0	0	0
12	Lecturer 8	6	0	0	0
13	Lecturer 9	5	0,43	0	1
14	Lecturer 10	3	2,43	3	3
15	Lecturer 11	3	2,43	2	3
16	Lecturer 12	2	3,43	4	4
17	Lecturer 13	2	3,43	3	4
18	Lecturer 14	2	3,43	4	4
19	Lecturer 15	2	3,43	3	4
20	Lecturer 16	2	3,43	3	4
21	Lecturer 17	1	4,43	5	4
22	Lecturer 18	0,6	4,83	5	5
23	Lecturer 19	0	5,43	5	5
24	Lecturer 20	0	5,43	6	5
25	Lecturer 21	0	5,43	5	5
26	Lecturer 22	0	5,43	5	5
27	Lecturer 23	0	5,43	6	5
28	Lecturer 24	0	5,43	5	5
29	Lecturer 25	0	5,43	6	5
30	Lecturer 26	0	5,43	5	5
31	Lecturer 27	0	5,43	6	5
32	Lecturer 28	0	5,43	5	5

Worksheet Formulas		
Range	Formula	
B2:E2	**B2**	=SUMME(B5:B32)
B4	**B4**	=(B2+B1)/ZEILEN(B5:B32)
C5:C32	**C5**	=WENN(B5>=B6;MAX(0;B$4-B5-SUMMENPRODUKT(--(C$4:C4=0);B$4:B4-B$4)/(ZEILEN(B$5:B$32)-SUMMENPRODUKT(--(C$4:C4=0))+1));"Values in column B are not decreasing!")
D5:D32	**D5**	=RUNDEN(SUMME(C$4:C5);0)-SUMME(D$4:D4)
E5	**E5**	=WENNFEHLER(RoundToSum(C5:C32;0);0)

RoundToSum Versus Other Methods

RoundToSum Versus Other "Simple" Methods

There are several different naïve approaches circulating around which try to round values preserving their rounded sum:

- (worst) Round all values but the last one and replace the last one by the rounded original sum minus the sum of the previously rounded values (i.e. aggregate all rounding errors in the last summand):

	A	B	C	
1		**Original Data**	**Aggregate Rounding Error**	**Formula in C**
2	**Total**	**2,594**	**2,59**	=SUM(C4:C8)
3				
4		0,875	0,88	=ROUND(B4,2)
5		0,865	0,87	=ROUND(B5,2)
6		0,344	0,34	=ROUND(B6,2)
7		0,455	0,46	=ROUND(B7,2)
8		0,055	**0,04**	=ROUND(B$2,2)-SUM(C$4:C7)

- (better, but still bad) Apply a cascading (sliding) round:

	A	B	C	
1		**Original Data**	**Cascading Round**	**Formula in C**
2	**Total**	2,593	2,59	=SUM(C4:C8)
3				
4		0,875	0,88	=ROUND(SUM($B$3:$B4),2)-SUM($C$3:$C3)
5		0,865	**0,86**	=ROUND(SUM($B$3:$B5),2)-SUM($C$3:$C4)
6		0,344	0,34	=ROUND(SUM($B$3:$B6),2)-SUM($C$3:$C5)
7		0,454	**0,46**	=ROUND(SUM($B$3:$B7),2)-SUM($C$3:$C6)
8		0,055	**0,05**	=ROUND(SUM($B$3:$B8),2)-SUM($C$3:$C7)

Let us compare these approaches to *RoundToSum*.

Calculation Example

We create 40 random numbers *RAND() * 1000* and compare as follows:

	A	B	C	D	E	F	G	H	I	J
1			I	II	III	IV	V	VI	VII	VIII
2			Original unrounded	RoundToSum	Cascading Round	Simple Round & Amend Last	Simple Round	Difference II - V	Difference III - V	Difference IV - V
3		Summands	948,5426666	948,54	948,54	948,54	948,54			
4			640,6107903	640,61	640,61	640,61	640,61			
5			604,8177225	604,82	604,82	604,82	604,82			
6			759,719267	759,72	759,72	759,72	759,72			
7			716,9320656	716,93	716,93	716,93	716,93			
8			263,431133	263,43	263,43	263,43	263,43			
9			726,0940269	726,09	726,10	726,09	726,09		0,01	
10			70,69027141	70,69	70,69	70,69	70,69			
11			468,6681995	468,67	468,67	468,67	468,67			
12			695,6816155	695,68	695,68	695,68	695,68			
13			68,51388814	68,51	68,51	68,51	68,51			
14			179,9413044	179,94	179,94	179,94	179,94			
15			994,1708842	994,17	994,17	994,17	994,17			
16			450,2225474	450,22	450,23	450,22	450,22		0,01	
17			875,4975592	875,50	875,49	875,5	875,5		-0,01	
18			217,4084507	217,41	217,41	217,41	217,41			
19			186,4643542	186,47	186,47	186,46	186,46	0,01	0,01	
20			428,5237989	428,52	428,52	428,52	428,52			
21			692,9424797	692,94	692,94	692,94	692,94			
22			460,6134853	460,61	460,62	460,61	460,61		0,01	
23			699,4999856	699,50	699,50	699,5	699,5			
24			512,7661261	512,77	512,76	512,77	512,77		-0,01	
25			173,039623	173,04	173,04	173,04	173,04			
26			385,9625179	385,96	385,96	385,96	385,96			
27			221,3543041	221,36	221,36	221,35	221,35	0,01	0,01	
28			945,2643498	945,27	945,26	945,26	945,26	0,01		
29			401,3771987	401,38	401,38	401,38	401,38			
30			666,2311689	666,23	666,23	666,23	666,23			
31			378,0140135	378,01	378,02	378,01	378,01		0,01	
32			446,3934267	446,39	446,39	446,39	446,39			
33			903,7448716	903,75	903,74	903,74	903,74	0,01		
34			987,4524282	987,45	987,46	987,45	987,45		0,01	
35			553,6299239	553,63	553,63	553,63	553,63			
36			349,8348857	349,84	349,83	349,83	349,83	0,01		
37			14,55826737	14,56	14,56	14,56	14,56			
38			152,9945856	153,00	152,99	152,99	152,99	0,01		
39			783,5934795	783,59	783,60	783,59	783,59		0,01	
40			178,9163192	178,92	178,91	178,92	178,92		-0,01	
41			922,6008936	922,60	922,60	922,6	922,6			
42			776,412911	776,41	776,42	776,47	776,41		0,01	0,06
43		Total	20903,12779	20.903,13	20.903,13	20.903,13	20.903,07	0,06	0,06	0,06
44	ABS Difference to Original			0,11	0,14	0,15	0,10			

Worksheet Formulas		
Range	Formula	
D3	**D3**	=RoundToSum(C3:C42)
E3	**E3**	=ROUND(SUM($C$3:$C3),2)-SUM(E$2:E2)
F3	**F3**	=ROUND(C3:C41,2)
F42	**F42**	=ROUND(C43,2)-SUM(F3:F41)
G3	**G3**	=ROUND(C3:C42,2)
H3:J3	**H3**	=IF(ABS(D3:D42-$G3:$G42)<0.000001,"",D3:D42-$G3:$G42)
C43:J43	**C43**	=SUM(C3:C42)
D44:G44	**D44**	=SUMPRODUCT(ABS(D3:D42-$C$3:$C$42))

As you can see, if we simply round each single number, the resulting sum would differ from the original rounded sum by 0.06. Column J (VIII) shows the difference of the aggregated rounding error -0.06 in the last summand. Column F (IV) shows the corresponding rounded numbers. Worst case would be here to come up with an aggregated rounding error of n * 0,005 with n being the count of your numbers. Example: Take 40 times the number 0.005 instead of the 40 random numbers.

Good practical examples, why you should not aggregate rounding errors in the last summand, are normally distributed samples of integers.

The cascading (sliding) round in column I (VII) shows 12 roundings to the wrong side. Column E (III) shows the corresponding rounded numbers. Worst case would be for the cascading round to round half of your numbers to the wrong side when all numbers could have been rounded correctly.

Example: Take 20 times the number -0.0049999 and then 20 times the number 0.0049999 instead of the 40 random numbers.

On the other hand, the optimal RoundToSum just rounds 6 values to the wrong side which result in the least number of changes which achieve the correct rounded sum. The worst case would now involve n/2 roundings to the wrong side with n being the count of your numbers. Example: Take 40 times the number 0.005 again instead of the 40 random numbers. This is the best solution with the smallest absolute rounding error for each number and then with the smallest number of roundings to the wrong side.

Conclusion

Use RoundToSum. It will apply the least number of changes and it will result in the correct sum with the smallest absolute (or relative) error.

A cascading round as shown above does not need any VBA nor does it apply any array formula, but it requires at least as many rounding differences as *RoundToSum* but can leave you with much more unnatural roundings which you can hardly explain to any senior manager.

But worst of all is the approach of aggregating all rounding differences in the last summand. Just imagine 1,000 people, each having 49 Cents, adding up to $490, which you should distribute fairly, but rounded to a whole Dollar. In this case you would end up with $490 at the last person, while *RoundToSum* would give the first 490 persons one Dollar each and all the others zero.

RoundToSum Compared to the D'Hondt Approach

RoundToSum implements the Hare-Niemeyer approach. In the context of distributing parliamentary seats, this method is superior to the D'Hondt approach. One key advantage is that the absolut value of the relative percentage difference from an ideal proportional distribution is generally lower, as illustrated by the following example:

	A	B	C	D	E	F
1		69	Seats		Rel. % diff from ideal distribution	
2	Party	Votes	D'Hondt	Hare-Niemeyer	D'Hondt	Hare-Niemeyer
3	A	576.100	30	29	3,175%	-0,265%
4	B	554.844	29	28	3,425%	-0,014%
5	C	94.920	4	5	-2,720%	0,719%
6	D	89.330	4	4	-1,749%	-1,749%
7	E	51.901	2	3	-2,131%	1,308%
8	Total	1.367.095	69	69		

Worksheet Formulas		
Range	Formula	
C3	C3	=sbdHondt(B1,B3:B7)
D3	D3	=RoundToSum(B1*B3:B7/B8,0)
E3:F3	E3	=(C3:C7/C$8-$B3:$B7/$B$8)/($B3/$B$8)
B8:D8	B8	=SUM(B3:B7)

sbDHondt Program Code

```
Function sbdHondt(lSeats As Long, vVotes As Variant) As Variant
'Implements the d'Hondt method for allocating seats in
'party-list proportional representation political election
'systems.
'(C) (P) by Bernd Plumhoff 01-Dec-2009 PB V0.10
Dim i As Long, k As Long, n As Long
Dim vA As Variant, vB As Variant, vR As Variant
Dim dMax As Double

With Application.WorksheetFunction
vA = .Transpose(.Transpose(vVotes))
vB = vA
n = UBound(vA, 1)
ReDim vR(1 To n, 1 To 1) As Variant
ReDim lDenom(1 To n) As Long

Do While i < lSeats
  'identify max
  dMax = .Max(vB)
  k = .Match(dMax, vB, 0)
  lDenom(k) = lDenom(k) + 1
  vB(k, 1) = vA(k, 1) / (lDenom(k) + 1#)
  vR(k, 1) = vR(k, 1) + 1
  i = i + 1
Loop
sbdHondt = vR
End With
End Function
```

Literature

Diaconis, P., & Freedman, D. (13. Juli 2007), On Rounding Percentages.

Sande, G. (2005, August 7), Guaranteed Controlled Rounding for Many Totals in Multi-way and Hierarchical Tables.

Random Number Generation (Excel / VBA)

Abstract

Random numbers you often need for simulations, for what-if-calculations, or to anonymize data. Here I present my collection of programs which generate random numbers: natural numbers, integers, or floating point numbers. I use Excel's built-in functions which means the generated numbers are pseudo random numbers.

Random Integers

Natural Random Numbers – *UniqRandInt*

Note: This function is shown here purely for historical reasons because it is efficiently executed and can be useful for learning about VBA compiler constants. The newer function *sbRandInt* (see Random Integers – *sbRandInt*) also allows negative lower bounds and determines the best execution method at runtime, not during compilation.

Sometimes, you need random integers that do not repeat, or only repeat a limited number of times. For example, if you need 20 positive random integers between 1 and 100, you would enter: *=UniqRandInt(20,100)*. If the range should be between 100 and 199, you would use *=UniqRandInt(20,100)+99*. In general:

=UniqRandInt(Count, End_Value – Start_Value + 1) + Start_Value - 1

If you need 10 positive random integers in the range 1 to 2, where each number may appear up to 10 times, use *=UniqRandInt(10,2,10)*. In this case, the compiler constant *ALLOW_REPETITION* must be set to *True*. And in case you want 3 numbers between 1 and 100 million which must not repeat, you should set the compiler constant *LATE_INITIALISATION* to *True*:

	A	B	C	D	E	F
1	**n**	6	12	10	6	3
2	**IRange**	6	6	2	6	100.000.000
3	**IMaxOccurrence**	1	2	10	1	1
4						
5	**Result**	5	1	2	1	84.876.019
6		6	5	2	2	39.223.814
7		4	1	2	3	51.271.980
8		3	2	1	6	
9		1	4	1	5	
10		2	6	1	4	
11			5	1		
12			2	1		
13			3	2		
14			6	2		
15			4			
16			3			

Worksheet Formulas		
Range	Formula	
B5:F5	**B5**	=TRANSPOSE(UniqRandInt(B1,B2,B3))

UniqRandInt Program Code

```
'If lRange >> n then set LATE_INITIALISATION to true. For example,
'if lRange=1,000,000 and if 1,000 cells are selected (n=1000).
#Const LATE_INITIALISATION = True
'If random integers may occur more than once, allow repetitions
#Const ALLOW_REPETITION = True

#If ALLOW_REPETITION Then
Function UniqRandInt(n As Long, ByVal lRange As Long, _
  Optional lMaxOccurence As Long = 1) As Variant
#Else
Function UniqRandInt(n As Long, ByVal lRange As Long) As Variant
#End If
'Returns n unique (=non-repeating) random integers within 1..lRange,
'lRange >= n. Set ALLOW_REPETITION = True and call with
'lMaxOccurences > 1 if random integers may occur more than once.
'(C) (P) by Bernd Plumhoff 30-Oct-2024 PB V1.04

Static bRandomized As Boolean
Dim vA             As Variant
Dim vR             As Variant
Dim i              As Long
Dim j              As Long
Dim lr             As Long

If Not bRandomized Then Randomize: bRandomized = True

#If ALLOW_REPETITION Then
  If lMaxOccurence < 1 Then
    UniqRandInt = CVErr(xlErrNum)
    Exit Function
  End If
  lRange = lRange * lMaxOccurence
#End If

If n > lRange Then UniqRandInt = CVErr(xlErrValue): Exit Function

ReDim vR(1 To n) As Variant

ReDim vA(1 To lRange)
#If Not LATE_INITIALISATION Then
  For i = 1 To lRange
    #If ALLOW_REPETITION Then
      vA(i) = Int((i - 1) / lMaxOccurence) + 1
    #Else
      vA(i) = i
    #End If
  Next i
#End If

i = 1
For j = 1 To UBound(vR, 1)
  lr = Int(((lRange - i + 1) * Rnd) + 1)
  #If LATE_INITIALISATION Then
    If vA(lr) = 0 Then
      #If ALLOW_REPETITION Then
        vR(j) = Int((lr - 1) / lMaxOccurence) + 1
      #Else
        vR(j) = lr
      #End If
    Else
  #End If
      vR(j) = vA(lr)
  #If LATE_INITIALISATION Then
    End If
    If vA(lRange - i + 1) = 0 Then
      #If ALLOW_REPETITION Then
        vA(lr) = Int((lRange - i + 1 - 1) / lMaxOccurence) + 1
      #Else
        vA(lr) = lRange - i + 1
      #End If
    Else
  #End If
      vA(lr) = vA(lRange - i + 1)
  #If LATE_INITIALISATION Then
    End If
  #End If
  i = i + 1
Next j

UniqRandInt = vR

End Function
```

Random Integers – *sbRandInt*

If you need random integers between two given values that do not repeat, or only repeat a limited number of times, I recommend using my user-defined function *sbRandInt*:

	A	B	C	D	E	F
1	**ICount**	7	14	10	12	3
2	**IMin**	-3	-3	1	1	-100.000.000
3	**IMax**	3	3	2	3	100.000.000
4	**IRept**	1	2	10	4	1
5						
6	**Result**	3	-1	1	3	27.752.674
7		-1	0	1	3	-9.543.539
8		0	3	2	2	-16.514.767
9		1	-3	1	3	
10		-2	-1	2	1	
11		-3	-3	2	3	
12		2	0	2	1	
13			2	2	1	
14			-2	2	2	
15			1	1	2	
16			1		1	
17			2		2	
18			3			
19			-2			

Worksheet Formulas		
Range	Formula	
B6:F6	**B6**	=TRANSPOSE(sbRandInt(B1,B2,B3,B4))

Note: If the possible range of random numbers is significantly larger than the number of random numbers to be generated, *sbRandInt* delays the initialization of its arrays at runtime, whereas with *UniqRandInt* (Natural Random Numbers – *UniqRandInt*), delayed initialization must be set using a compiler constant before the program runs.

sbRandInt – Program Code

```
Function sbRandInt(ByVal lCount As Long, _
  lMin As Long, _
  lMax As Long, _
  Optional lRept As Long = 1) As Variant
'Returns lCount random integers between lMin and lMax, each one
'occurring zero to lRept times. lMax - lMin + 1 must be greater
'or equal to lCount.
'Error values:
'#NUM!   - lRept is less than 1
'#REF!   - lCount is greater than (lMax - lMin + 1) * lRept
'#VALUE! - lCount is less than 1
'(C) (P) by Bernd Plumhoff  30-Dec-2024 PB V1.02
Static bRandomized As Boolean
Dim i As Long, j As Long, k As Long
Dim lRnd As Long, lRange As Long
Const CLateInitFactor = 50

If lCount < 1 Then sbRandInt = CVErr(xlErrValue): Exit Function
If lRept < 1 Then sbRandInt = CVErr(xlErrNum): Exit Function
If lCount > (lMax - lMin + 1) * lRept Then sbRandInt = CVErr(xlErrRef): Exit Function

lRange = (lMax - lMin + 1) * lRept

ReDim lr(1 To lCount) As Long

If Not bRandomized Then Randomize: bRandomized = True

ReDim lT(1 To lRange) As Long
'If we have a huge range of possible random integers and a comparably
'small number of draws, i.e. if (lMax - lMin) * lRept >> lCount
'then we can save some runtime with late initialization.
If lRange / lCount < CLateInitFactor Then
  For i = 1 To lRange
    lT(i) = Int((i - 1) / lRept) + lMin
  Next i
End If

i = 1
If lRange / lCount < CLateInitFactor Then
  For k = 1 To UBound(lr)
    lRnd = Int(((lRange - i + 1) * Rnd) + 1)
    lr(k) = lT(lRnd)
    lT(lRnd) = lT(lRange - i + 1)
    i = i + 1
  Next k
Else
  j = lMin: If lMin <= 0 And lMax >= 0 Then j = 1
  For k = 1 To UBound(lr)
    lRnd = Int(((lRange - i + 1) * Rnd) + 1)
    If lT(lRnd) = 0 Then
      lr(k) = Int((lRnd - 1) / lRept) + j
    Else
      lr(k) = lT(lRnd)
    End If
    If lT(lRange - i + 1) = 0 Then
      lT(lRnd) = Int((lRange - i) / lRept) + j
    Else
      lT(lRnd) = lT(lRange - i + 1)
    End If
    i = i + 1
  Next k
  'If lRange includes zero we need to shift result array
  If lMin <= 0 And lMax >= 0 Then
    For k = 1 To UBound(lr)
      lr(k) = lr(k) + lMin - 1
    Next k
  End If
End If

sbRandInt = lr

End Function
```

Random Numbers with a Specified Sum

Minimum of Random Numbers given – *sbLongRandSumN*

Do you need 20 natural random numbers with a sum of 100? Then I suggest using my custom function *sbLongRandSumN* shown here. You can generate any number of integers with a specified sum, while ensuring that the generated numbers do not fall below a specified minimum:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	lSum	lCount	lMin	Total	sbLongRandSumN									
2	92	7	6	92	8	6	9	7	17	13	32			
3	87	6	5	87	6	5	5	5	11	55				
4	58	4	7	58	12	8	8	30						
5	21	3	2	21	9	9	3							
6	65	10	4	65	5	4	5	5	4	4	5	4	4	25
7	64	3	12	64	35	16	13							
8	46	3	15	46	16	15	15							
9	83	3	16	83	23	16	44							
10	59	10	5	59	6	5	5	5	5	5	5	5	8	10

Worksheet Formulas		
Range	Formula	
D2:D10	**D2**	=SUM(E2:K2)
E2:E10	**E2**	=sbLongRandSumN(A2,B2,C2)

sbLongRandSumN Program Code

```
Function sbLongRandSumN(lSum As Long, _
    ByVal lCount As Long, _
    Optional ByVal lMin As Long = 0) As Variant
'Generates lCount random integers greater equal lMin
'which sum up to lSum.
'(C) (P) by Bernd Plumhoff 26-Apr-2013 PB V0.1
Dim i As Long
Dim lSumRest As Long

If lCount * lMin > lSum Then
    sbLongRandSumN = CVErr(xlErrNum)
    Exit Function
End If
If lCount < 1 Then
    sbLongRandSumN = CVErr(xlErrValue)
    Exit Function
End If
Randomize
ReDim vR(1 To lCount) As Variant
lSumRest = lSum
For i = lCount To 2 Step -1
    vR(i) = lMin + Int(Rnd * (lSumRest - lMin * i))
    lSumRest = lSumRest - vR(i)
Next i
vR(1) = lSumRest
sbLongRandSumN = vR
End Function
```

Minimum and Maximum of Random Numbers given – *sbRandIntFixSum*

With *sbRandIntFixSum* you can generate *lCount* random integers between *lMin* and *lMax* with the sum *lSum*.

	A	B	C	D	E	F
1	**Sum**	1000			**Check**	1000
2	**Lower Bound**	30				
3	**Upper Bound**	110				
4	**Count**	10				
5				**Lower**	**Upper**	
6				**Border**	**Border**	**Random**
7	**Run**	**Rest**	**Count**	**Rest**	**Rest**	**Draw**
8	0	1000	10	30	110	
9	1	1000	10	30	110	68
10	2	932	9	52	110	92
11	3	840	8	70	110	87
12	4	753	7	93	110	94
13	5	659	6	109	110	110
14	6	549	5	109	110	110
15	7	439	4	109	110	109
16	8	330	3	110	110	110
17	9	220	2	110	110	110
18	10	110	1	110	110	110

Worksheet Formulas		
Range	Formula	
F1	**F1**	=SUM(IFERROR(F9:F29,0))
A8	**A8**	=SEQUENCE(B4+1,,0)
B8	**B8**	=$B$1
B9:B29	**B9**	=B8-F8
C8	**C8**	=$B$4
C9:C29	**C9**	=C8-1*(A9<>1)
D8	**D8**	=$B$2
D9:D29	**D9**	=ROUNDUP(MAX(D8,MIN(B9/C9,B9/C9-(C9-1)*(E8-B9/C9))),0)
E8	**E8**	=$B$3
E9:E29	**E9**	=ROUNDDOWN(MIN(E8,MAX(B9/C9,B9/C9+(C9-1)*(B9/C9-D8))),0)
F9:F29	**F9**	=INT(RAND()*(E9-D9+1)+D9)

sbRandIntFixSum Program Code

```
Function sbRandIntFixSum(lSum As Long, lMin As Long, _
    lMax As Long, Optional lCount As Long = 0, _
    Optional bUseRandTriang As Boolean = True, _
    Optional bVolatile As Boolean = False) As Variant
'Returns lCount (or selected cell count in case a range is select when
'called as a matrix formula) random integers between lMin and lMax
'which sum up to lSum. If bUseRandTriang the sbRandTriang distribution
'is used to "bias" the randomness to be "less extreme".

'Error values:
'#NUM!   - No solution exists
'#VALUE! - lCount is less than 1
'(C) (P) by Bernd Plumhoff 05-Aug-2020 PB V0.3

Dim i As Long
Dim lRnd As Long, lMinPrev As Long
Dim lRow As Long, lCol As Long

With Application

If TypeName(.Caller) = "Range" And lCount = 0 Then
    lCount = .Caller.Count
    ReDim lR(1 To .Caller.Rows.Count, 1 To .Caller.Columns.Count) As Long
ElseIf lCount < 1 Then
    sbRandIntFixSum = CVErr(xlErrValue)
    Exit Function
Else
    ReDim lR(1 To lCount, 1 To 1) As Long
End If

Randomize
If bVolatile Then .Volatile

For lRow = 1 To UBound(lR, 1)
    For lCol = 1 To UBound(lR, 2)
        lMinPrev = lMin
        lMin = .RoundUp(.Max(lMin, .Min(lSum / lCount, lSum / lCount _
               - (lCount - 1) * (lMax - lSum / lCount))), 0)
        lMax = .RoundDown(.Min(lMax, .Max(lSum / lCount, lSum / lCount _
               + (lCount - 1) * (lSum / lCount - lMinPrev))), 0)
        If lMin > lMax Or lSum / lCount <> .Median(lMin, lMax, lSum / _
            lCount) Then
            'No solution exists
            sbRandIntFixSum = CVErr(xlErrNum)
            Exit Function
        End If
        If bUseRandTriang Then
            If lMin = lMax Then
                lRnd = lMin
            Else
                lRnd = Int(sbRandTriang(CDbl(lMin), _
                       lSum / lCount, CDbl(lMax)) + 0.5)
            End If
        Else
            lRnd = Int(Rnd() * (lMax - lMin + 1) + lMin)
        End If
        lR(lRow, lCol) = lRnd
        lSum = lSum - lRnd
        lCount = lCount - 1
    Next lCol
Next lRow

sbRandIntFixSum = lR
End With

End Function
```

Usage Examples for Random Integers

Krabat, the Satanic Mill – How old can the apprentices become?

Krabat, the Satanic Mill is a youth novel by Otfried Preußler. I found the story fascinating, but somewhat illogical: 12 apprentices work in the mill. Every year, one dies, and every year a new apprentice, aged 14, is taken in. All of them age three years within one year.

After 30 years, there could be an apprentice who is 101 years old:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	**Start age**				14										
2	**Maximum age**				101										
3	**Max during first 20 years**				71										
4															
5				**Apprentices and their age over time**											
6	**Mill Year**	**Real Year**	**Who dies?**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
7	1	1		14	14	14	14	14	14	14	14	14	14	14	14
8	2	4	6	17	17	17	17	17	14	17	17	17	17	17	17
9	3	7	10	20	20	20	20	20	17	20	20	20	14	20	20
10	4	10	11	23	23	23	23	23	20	23	23	23	17	14	23
11	5	13	11	26	26	26	26	26	23	26	26	26	20	14	26
12	6	16	12	29	29	29	29	29	26	29	29	29	23	17	14
13	7	19	6	32	32	32	32	32	14	32	32	32	26	20	17
14	8	22	10	35	35	35	35	35	17	35	35	35	14	23	20
15	9	25	10	38	38	38	38	38	20	38	38	38	14	26	23
16	10	28	3	41	41	14	41	41	23	41	41	41	17	29	26
17	11	31	5	44	44	17	44	14	26	44	44	44	20	32	29
18	12	34	12	47	47	20	47	17	29	47	47	47	23	35	14
19	13	37	6	50	50	23	50	20	14	50	50	50	26	38	17
20	14	40	8	53	53	26	53	23	17	53	14	53	29	41	20
21	15	43	7	56	56	29	56	26	20	14	17	56	32	44	23
22	16	46	2	59	14	32	59	29	23	17	20	59	35	47	26
23	17	49	7	62	17	35	62	32	26	14	23	62	38	50	29
24	18	52	6	65	20	38	65	35	14	17	26	65	41	53	32
25	19	55	9	68	23	41	68	38	17	20	29	14	44	56	35
26	20	58	7	71	26	44	71	41	20	14	32	17	47	59	38
27	21	61	7	74	29	47	74	44	23	14	35	20	50	62	41
28	22	64	10	77	32	50	77	47	26	17	38	23	14	65	44
29	23	67	11	80	35	53	80	50	29	20	41	26	17	14	47
30	24	70	11	83	38	56	83	53	32	23	44	29	20	14	50
31	25	73	7	86	41	59	86	56	35	14	47	32	23	17	53
32	26	76	9	89	44	62	89	59	38	17	50	14	26	20	56
33	27	79	10	92	47	65	92	62	41	20	53	17	14	23	59
34	28	82	6	95	50	68	95	65	14	23	56	20	17	26	62
35	29	85	9	98	53	71	98	68	17	26	59	14	20	29	65
36	30	88	6	101	56	74	101	71	14	29	62	17	23	32	68

Worksheet Formulas		
Range	Formula	
E2	**E2**	=MAX(D8:O36)
E3	**E3**	=MAX(D8:O26)
B7:B36	**B7**	=A7*3-2
C8:C36	**C8**	=ROUNDDOWN(RAND()*12,0)+1
D8:O36	**D8**	=IF(COLUMN()-3=$C8,$E$1,D7+3)

Generate a Math Test with Random Integer Inputs – Generate_Math_Test

You like to generate a math test for your kid or for your class?

The input in sheet Main:

	A	B	C	D	E	F	G	H	I	J
1	**Min**	**Max**		**Sample Size**	15					
2	(									
3	1	5		Generate Random Sample						
4	+									
5	5	9		15 Expressions and Results, generated Thursday 09-Jun-2022 16:15:50						
6	)									
7	*									
8	2	4								

The solution output in sheet Sample_Q_and_A:

	A	B	C	D	E	F	G
1	15 Expressions and Results, generated Thursday 09-Jun-2022 16:15						
2	Expressio	equals	Result				
3	(4+7)*2	=	22				
4	(2+5)*4	=	28				
5	(2+9)*3	=	33				
6	(5+8)*2	=	26				
7	(5+9)*2	=	28				
8	(5+9)*2	=	28				
9	(3+7)*2	=	20				
10	(3+7)*3	=	30				
11	(2+6)*4	=	32				
12	(3+9)*2	=	24				
13	(2+7)*3	=	27				
14	(4+7)*2	=	22				
15	(4+8)*2	=	24				
16	(3+6)*4	=	36				
17	(2+6)*2	=	16				

The test output in sheet Sample_Q:

	A	B	C	D	E
1	15 Questions, generated Thursday 09-Jun-2022 16:15				
2	Expressio	equals	?		
3	(4+7)*2	=	?		
4	(2+5)*4	=	?		
5	(2+9)*3	=	?		
6	(5+8)*2	=	?		
7	(5+9)*2	=	?		
8	(5+9)*2	=	?		
9	(3+7)*2	=	?		
10	(3+7)*3	=	?		
11	(2+6)*4	=	?		
12	(3+9)*2	=	?		
13	(2+7)*3	=	?		
14	(4+7)*2	=	?		
15	(4+8)*2	=	?		
16	(3+6)*4	=	?		
17	(2+6)*2	=	?		

Generate_Math_Test Program Code

```
Sub Generate_Math_Test()
'(C) (P) by Bernd Plumhoff  09-Jun-2022 PB V1.0
Dim vi, vTime
Dim lInputRow As Long
Dim lOutputRow As Long
Dim lNum As Long
Dim sExpr As String

Dim state As SystemState
Application.StatusBar = False
Set state = New SystemState

vTime = Now
Randomize
wsQA.Range("1:1048576").Delete
wsQ.Range("1:1048576").Delete
wsQA.[a1] = Range("Sample_Size") & " Expressions and Results, generated " & _
            Format(vTime, "dddd dd-mmm-yyyy hh:mm")
wsQA.[a2] = "Expression"
wsQA.[b2] = "equals"
wsQA.[c2] = "Result"
wsQ.[a1] = Range("Sample_Size") & " Questions, generated " & Format(vTime, "dddd dd-mmm-yyyy hh:mm")
wsQ.[a2] = "Expression"
wsQ.[b2] = "equals"
wsQ.[c2] = "?"

lInputRow = 2
sExpr = ""
Do While Not IsEmpty(wsMain.Cells(lInputRow, 1))
  If IsNumeric(wsMain.Cells(lInputRow, 1)) And IsNumeric(wsMain.Cells(lInputRow, 2)) Then
    lNum = wsMain.Cells(lInputRow, 1) + Int(Rnd() * (1 + wsMain.Cells(lInputRow, 2) - _
           wsMain.Cells(lInputRow, 1)))
    sExpr = sExpr & lNum
  Else
    sExpr = sExpr & wsMain.Cells(lInputRow, 1).Text
  End If
  lInputRow = lInputRow + 1
Loop
If IsError(Evaluate(sExpr)) Then
  wsMain.[d5] = "Expression """ & sExpr & """ evaluates to error!"
  Exit Sub
End If

For lOutputRow = 2 To 1 + Range("Sample_Size")
  lInputRow = 2
  sExpr = ""
  Do While Not IsEmpty(wsMain.Cells(lInputRow, 1))
    If IsNumeric(wsMain.Cells(lInputRow, 1)) And IsNumeric(wsMain.Cells(lInputRow, 2)) Then
      lNum = wsMain.Cells(lInputRow, 1) + Int(Rnd() * (1 + wsMain.Cells(lInputRow, 2) - _
             wsMain.Cells(lInputRow, 1)))
      sExpr = sExpr & lNum
    Else
      sExpr = sExpr & wsMain.Cells(lInputRow, 1).Text
    End If
    lInputRow = lInputRow + 1
  Loop
  wsQA.Cells(lOutputRow + 1, 1) = sExpr
  wsQA.Cells(lOutputRow + 1, 2) = "="
  wsQ.Cells(lOutputRow + 1, 1) = sExpr
  wsQ.Cells(lOutputRow + 1, 2) = "="
  If IsError(Evaluate(sExpr)) Then
    wsQA.Cells(lOutputRow + 1, 3) = "Expression """ & sExpr & """ evaluates to error!"
    wsQ.Cells(lOutputRow + 1, 3) = "Expression """ & sExpr & """ evaluates to error!"
  Else
    wsQA.Cells(lOutputRow + 1, 3) = Evaluate(sExpr)
    wsQ.Cells(lOutputRow + 1, 3) = "?"
  End If
Next lOutputRow

wsMain.[d5] = Range("Sample_Size") & " Expressions and Results, generated " & _
              Format(vTime, "dddd dd-mmm-yyyy hh:mm:ss")

End Sub
```

Monte Carlo Simulation to Generate Teams Fairly – *sbGenerateTeams*

Do you and your 15 fiends like to play in 4 teams with 4 players each and you ask yourselves how to create these teams randomly but with similar strengths?

You can achieve this with *sbGenerateTeams*:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Player #	Player Name	Player Skill		# of Teams		# of Monte Carlo simulations		Team #	Player Name	Player Skill		Team #	Sum of Skills
2	1	Andrew	27		4		20000		1	David	47		1	141
3	2	Benjamin	38						1	Andrew	27		2	140
4	3	Charlie	31		# of players per team		Start Team Generation		1	Peter	22		3	140
5	4	David	47		4				1	Lucy	45		4	140
6	5	Edward	35						2	George	41		StDev	0,5
7	6	Frederick	26						2	King	25			
8	7	George	41						2	Isaac	39			
9	8	Harry	43						2	Edward	35			
10	9	Isaac	39						3	Harry	43			
11	10	Jack	44						3	Jack	44			
12	11	King	25						3	Oliver	22			
13	12	Lucy	45						3	Charlie	31			
14	13	Mary	26						4	Frederick	26			
15	14	Nellie	50						4	Benjamin	38			
16	15	Oliver	22						4	Nellie	50			
17	16	Peter	22						4	Mary	26			

This program combines several functionalities that I frequently use:

- The *SystemState* class reduces runtime.
- With *enumerations*, I organize access to columns flexibly – for additional or removed columns, I simply modify the enumeration, and the program automatically adjusts the column numbers.
- New shuffling of a set of elements using *UniqRandInt* (Natural Random Numbers – UniqRandInt).
- Test data (names) were generated using *sbGenerateTestData* (Generate Test Data – sbGenerateTestData).

A More Complex Example

If you want to generate random teams of equal strength that have subgroups of different player types, you can assign skill values with different powers of ten (or other powers) to each subgroup. You just need to ensure that all subgroups in all teams have the same number of players – except for the subgroup with the lowest skill values, which can have a different number of players in each team:

Player #	Player Name	Player Skill
1	Goalkeeper 1	50000
2	Goalkeeper 2	50000
3	Defender 1	5000
4	Defender 2	5000
5	Defender 3	5000
6	Defender 4	5000
7	Defender 5	5000
8	Defender 6	5000
9	Defender 7	5000
10	Defender 8	500
11	Midfielder 1	500
12	Midfielder 2	500
13	Midfielder 3	500
14	Midfielder 4	500
15	Midfielder 5	500
16	Midfielder 6	500
17	Striker 1	50
18	Striker 2	50
19	Striker 3	50
20	Striker 4	50
21	Striker 5	50
22	Striker 6	50
23	Striker 7	50

# of Teams
2

# of players per team
12

# of Monte Carlo simulations
20000

Start Team Generation

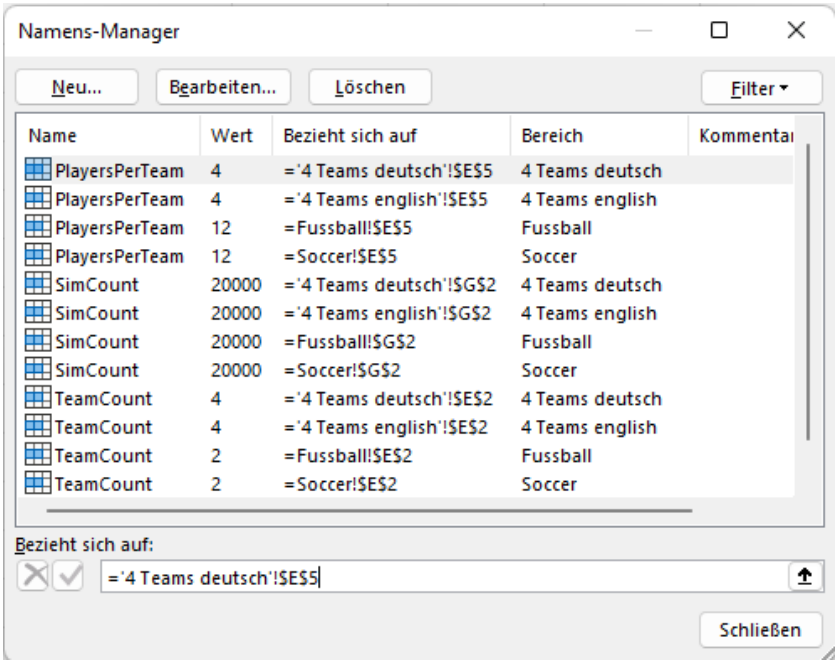
Team #	Player Name	Player Skill
1	Goalkeeper 2	50000
1	Striker 2	50
1	Defender 8	500
1	Midfielder 5	500
1	Midfielder 4	500
1	Defender 6	5000
1	Midfielder 1	500
1	Defender 4	5000
1	Midfielder 3	500
1	Defender 1	5000
1	Midfielder 2	500
1	Midfielder 6	500
2	Striker 6	50
2	Defender 3	5000
2	Striker 5	50
2	Defender 2	5000
2	Striker 3	50
2	Defender 5	5000
2	Striker 7	50
2	Goalkeeper 1	50000
2	Striker 1	50
2	[Empty]	0
2	Defender 7	5000
2	Striker 4	50

Team #	Sum of Skills
1	68550
2	70300
StDev	1237,43687

You can adjust the skill values after a game. For example, you could increase the values of the winners by 1, up to a maximum value for each subgroup. Or, you could decrease the values of the losers by 1, until a minimum value for each subgroup is reached. This ensures that changes in skill levels are represented fairly and transparently.

sbGenerateTeams Program Code

Note: This program requires (uses) the class *SystemState* and the user-defined function *UniqRandInt*, and it is aligned to these named ranges:



```
Option Explicit

#Const I_Want_Colors = True

#If I_Want_Colors Then
Private Enum xlCI 'Excel Color Index
: xlCIBlack = 1: xlCIWhite: xlCIRed: xlCIBrightGreen: xlCIBlue '1 - 5
: xlCIYellow: xlCIPink: xlCITurquoise: xlCIDarkRed: xlCIGreen '6 - 10
: xlCIDarkBlue: xlCIDarkYellow: xlCIViolet: xlCITeal: xlCIGray25 '11 - 15
: xlCIGray50: xlCIPeriwinkle: xlCIPlum: xlCIIvory: xlCILightTurquoise '16 - 20
: xlCIDarkPurple: xlCICoral: xlCIOceanBlue: xlCIIceBlue: xlCILightBrown '21 - 25
: xlCIMagenta2: xlCIYellow2: xlCICyan2: xlCIDarkPink: xlCIDarkBrown '26 - 30
: xlCIDarkTurquoise: xlCISeaBlue: xlCISkyBlue: xlCILightTurquoise2: xlCILightGreen '31 - 35
: xlCILightYellow: xlCIPaleBlue: xlCIRose: xlCILavender: xlCITan '36 - 40
: xlCILightBlue: xlCIAqua: xlCILime: xlCIGold: xlCILightOrange '41 - 45
: xlCIOrange: xlCIBlueGray: xlCIGray40: xlCIDarkTeal: xlCISeaGreen '46 - 50
: xlCIDarkGreen: xlCIGreenBrown: xlCIBrown: xlCIDarkPink2: xlCIIndigo '51 - 55
: xlCIGray80 '56
End Enum
#End If

Enum col_worksheet
    col_LBound = 0 'To be able to iterate from here + 1
    col_in_player_no
    col_in_player_name
    col_in_player_skill
    col_blank_1
    col_in_team_stats
    col_blank_2
    col_in_sim_stats
    col_blank_3
    col_out_team_no
    col_out_player_name
    col_out_player_skill
    col_blank_4
    col_stat_team_no
    col_stat_sum_skills
    col_Ubound 'To be able to iterate until here - 1
End Enum 'col_worksheet

Sub sbGenerateTeams()
'Implements a simple Monte Carlo simulation to randomly generate
'teams fairly, keeping track of the teams with the lowest standard
'deviation of skill sums.
'This sub needs UniqRandInt - google for sulprobil and uniqrandint.
'and the SystemState class - google for sulprobil and systemstate.
'(C) (P) by Bernd Plumhoff 07-Nov-2024 PB V0.5

Dim i                   As Long
Dim j                   As Long
Dim k                   As Long
Dim n                   As Long
Dim teamcount           As Long
Dim playersperteam      As Long
Dim stdev_hc_sum        As Double
Dim min_stdev           As Double
Dim s                   As Double
Dim v                   As Variant
Dim wsI                 As Worksheet
Dim state               As SystemState

'Initialize
Set state = New SystemState
Set wsI = ThisWorkbook.ActiveSheet
teamcount = wsI.Range("TeamCount")
wsI.Range("PlayersPerTeam").Calculate
playersperteam = wsI.Range("PlayersPerTeam")
n = teamcount * playersperteam
ReDim hc(1 To n) As Double
ReDim mina(1 To n) As Double
ReDim hc_sum(1 To teamcount) As Double
wsI.Cells.Interior.ColorIndex = False
#If I_Want_Colors Then
wsI.Range("A1:C1").Interior.ColorIndex = xlCIYellow
wsI.Range("E1").Interior.ColorIndex = xlCIYellow
wsI.Range("G1").Interior.ColorIndex = xlCIYellow
wsI.Range("E4").Interior.ColorIndex = xlCIYellow
wsI.Range("E2").Interior.ColorIndex = xlCILightYellow
wsI.Range("G2").Interior.ColorIndex = xlCILightYellow
wsI.Range("E5").Interior.ColorIndex = xlCILightYellow
wsI.Range("I1:K1").Interior.ColorIndex = xlCIBrightGreen
wsI.Range("M1:N1").Interior.ColorIndex = xlCIBrightGreen
wsI.Range("M" & teamcount + 2 & ":N" & teamcount + 2).Interior.ColorIndex = xlCILightGreen
#End If
For j = 1 To n
  hc(j) = wsI.Cells(j + 1, col_in_player_skill)
  #If I_Want_Colors Then
  wsI.Range("A" & j + 1 & ":C" & j + 1).Interior.ColorIndex = xlCILightYellow
  #End If
Next j
```

```
min_stdev = 1E+308

k = 1
Do
  v = UniqRandInt(n, n)
  For i = 1 To teamcount
    hc_sum(i) = 0
    For j = 1 To playersperteam
      hc_sum(i) = hc_sum(i) + hc(v((i - 1) * playersperteam + j))
    Next j
  Next i
  stdev_hc_sum = WorksheetFunction.StDev(hc_sum)
  If stdev_hc_sum < min_stdev Then
    For i = 1 To n
      mina(i) = v(i)
    Next i
    min_stdev = stdev_hc_sum
    Application.StatusBar = "Iteration " & k & ", new min stdev = " & min_stdev
  End If
  k = k + 1
Loop Until k > wsI.Range("SimCount")

wsI.Range(wsI.Cells(2, col_out_team_no), _
  wsI.Cells(1000, col_stat_sum_skills)).ClearContents

For i = 1 To teamcount
  s = 0#
  For j = 1 To playersperteam
    wsI.Cells(1 + (i - 1) * playersperteam + j, col_out_team_no) = i
    wsI.Cells(1 + (i - 1) * playersperteam + j, col_out_player_name) = _
      IIf("" = wsI.Cells(1 + mina((i - 1) * playersperteam + j), col_in_player_name), _
        "[Empty]", wsI.Cells(1 + mina((i - 1) * playersperteam + j), col_in_player_name))
    wsI.Cells(1 + (i - 1) * playersperteam + j, col_out_player_skill) = _
      CDbl(wsI.Cells(1 + mina((i - 1) * playersperteam + j), col_in_player_skill))
    s = s + wsI.Cells(1 + mina((i - 1) * playersperteam + j), col_in_player_skill)
    #If I_Want_Colors Then
    wsI.Range("I" & 1 + (i - 1) * playersperteam + j & ":K" & 1 + (i - 1) * _
      playersperteam + j).Interior.ColorIndex = xlCILightGreen
    #End If
  Next j
  wsI.Cells(1 + i, col_stat_team_no) = i
  wsI.Cells(1 + i, col_stat_sum_skills) = s
  #If I_Want_Colors Then
  wsI.Range("M" & i + 1 & ":N" & i + 1).Interior.ColorIndex = xlCILightGreen
  #End If
Next i
wsI.Cells(2 + teamcount, col_stat_team_no) = "StDev"
wsI.Cells(2 + teamcount, col_stat_sum_skills) = min_stdev
End Sub
```

Monte Carlo Simulation for a Regatta Flight Plan – *sbRegattaFlightPlan*

If you want to generate a Regatta Flight Plan for 12 flights (sailing rounds) for 12 individual sailors with 4 available boats, where:

- No sailor competes against another too frequently
- No sailor is assigned a boat too often
- Ideally, no sailor has to sail in consecutive flights

then hopefully this program, which uses *UniqRandInt* the class *SystemState*, will help you.

Input:

	A	B
1		
2	Start Simulation	
3		
4		
5	**Number of Simulations**	20.000
6	**Number of Flights**	12
7	**Number of Boats**	4
8	**Sailors**	
9	Abe	
10	Ben	
11	Carl	
12	Dan	
13	Eve	
14	Fred	
15	Giles	
16	Hanna	
17	Ida	
18	Joe	
19	Kim	
20	Luke	

Output:

	D	E	F	G	H	I	J	K	L	M	N	O	P
1		Flight 1	Flight 2	Flight 3	Flight 4	Flight 5	Flight 6	Flight 7	Flight 8	Flight 9	Flight 10	Flight 11	Flight 12
2	Boat 1	Abe	Fred	Ida	Hanna	Eve	Giles	Ida	Luke	Eve	Dan	Joe	Fred
3	Boat 2	Hanna	Luke	Eve	Luke	Abe	Joe	Kim	Hanna	Carl	Ben	Giles	Abe
4	Boat 3	Giles	Carl	Dan	Carl	Ida	Ben	Dan	Joe	Giles	Kim	Ida	Eve
5	Boat 4	Kim	Joe	Ben	Kim	Dan	Fred	Abe	Ben	Fred	Hanna	Carl	Luke
6	Maximal meet of sailor pairs	3											
7	Maximal repetition of boat per sailor	2											
8	Number of sailors with adjacent flights	0											

sbRegattaFlightPlan Program Ccode

This program requires *UniqRandInt* and uses the class *SystemState*.

```
#Const I_Want_Colors = True
#If I_Want_Colors Then
Private Enum xlCI 'Excel Color Index
: xlCIBlack = 1: xlCIWhite: xlCIRed: xlCIBrightGreen: xlCIBlue '1 - 5
: xlCIYellow: xlCIPink: xlCITurquoise: xlCIDarkRed: xlCIGreen '6 - 10
: xlCIDarkBlue: xlCIDarkYellow: xlCIViolet: xlCITeal: xlCIGray25 '11 - 15
: xlCIGray50: xlCIPeriwinkle: xlCIPlum: xlCIIvory: xlCILightTurquoise '16 - 20
: xlCIDarkPurple: xlCICoral: xlCIOceanBlue: xlCIIceBlue: xlCILightBrown '21 - 25
: xlCIMagenta2: xlCIYellow2: xlCICyan2: xlCIDarkPink: xlCIDarkBrown '26 - 30
: xlCIDarkTurquoise: xlCISeaBlue: xlCISkyBlue: xlCILightTurquoise2: xlCILightGreen '31 - 35
: xlCILightYellow: xlCIPaleBlue: xlCIRose: xlCILavender: xlCITan '36 - 40
: xlCILightBlue: xlCIAqua: xlCILime: xlCIGold: xlCILightOrange '41 - 45
: xlCIOrange: xlCIBlueGray: xlCIGray40: xlCIDarkTeal: xlCISeaGreen '46 - 50
: xlCIDarkGreen: xlCIGreenBrown: xlCIBrown: xlCIDarkPink2: xlCIIndigo '51 - 55
: xlCIGray80 '56
End Enum
Private xlFC(1 To 56) As Boolean 'Font color: True is black, False is white
#End If

Sub sbRegattaFlightPlan()
'Performs a simple Monte Carlo simulation to create a regatta flight plan.
'(C) (P) by Bernd Plumhoff 07-Jan-2023 PB V0.3
Dim i As Long, j As Long, k As Long, m As Long
Dim lAdjacentFlights        As Long
Dim lBestSailorInBoat       As Long
Dim lBestSailorMeetSailor   As Long
Dim lBoatCount              As Long
Dim lFlightCount            As Long
Dim lLowestAdjacentFlights  As Long
Dim lMaxSailorInBoat        As Long
Dim lMaxSailorMeetSailor    As Long
Dim lSailorIndex            As Long
Dim lSailorCount            As Long
Dim lSimulationCount        As Long
Dim ws                      As Worksheet
Dim state                   As SystemState

With Application.WorksheetFunction

'Initialize
Set ws = ThisWorkbook.ActiveSheet
Set state = New SystemState
ws.Cells.Interior.Pattern = xlNone
ws.Cells.Interior.TintAndShade = 0
ws.Cells.Interior.PatternTintAndShade = 0
ws.Cells.Font.ColorIndex = xlAutomatic
ws.Cells.Font.TintAndShade = 0

#If I_Want_Colors Then
For i = 1 To 56: xlFC(i) = True: Next i
xlFC(xlCIBlack) = False: xlFC(xlCIRed) = False: xlFC(xlCIBlue) = False
xlFC(xlCIDarkRed) = False: xlFC(xlCIGreen) = False: xlFC(xlCIDarkBlue) = False
xlFC(xlCIDarkYellow) = False: xlFC(xlCIViolet) = False: xlFC(xlCIDarkPurple) = False
xlFC(xlCILightBrown) = False: xlFC(xlCIDarkPink) = False: xlFC(xlCIDarkBrown) = False
xlFC(xlCISeaBlue) = False: xlFC(xlCIBlueGray) = False: xlFC(xlCIDarkTeal) = False
xlFC(xlCIDarkGreen) = False: xlFC(xlCIGreenBrown) = False: xlFC(xlCIIndigo) = False
xlFC(xlCIGray80) = False
#End If

Randomize
i = Range("Sailors").Row + 1
Do While Not IsEmpty(ws.Cells(i + lSailorCount, 1))
  lSailorCount = lSailorCount + 1
Loop
ReDim sSailor(1 To lSailorCount) As String
i = Range("Sailors").Row
j = 1
Do While Not IsEmpty(ws.Cells(i + j, 1))
  sSailor(j) = ws.Cells(i + j, 1)
  #If I_Want_Colors Then
  k = (j Mod 56) + 1
  ws.Cells(i + j, 1).Interior.ColorIndex = k
  If xlFC(k) Then
    ws.Cells(i + j, 1).Font.ColorIndex = xlCIBlack
  Else
    ws.Cells(i + j, 1).Font.ColorIndex = xlCIWhite
  End If
  #End If
  j = j + 1
Loop

lBoatCount = Range("Boats")
```

```
lFlightCount = Range("Flights")
lSimulationCount = Range("Simulations")
lBestSailorMeetSailor = lSailorCount
lBestSailorInBoat = lBoatCount
lLowestAdjacentFlights = lFlightCount * lSailorCount

If lFlightCount * lBoatCount Mod lSailorCount <> 0 Then
  Call MsgBox("Number of flights" & vbCrLf & "times number of boats" & vbCrLf & _
    "needs to be divisible" & vbCrLf & "by number of sailors!", vbOKOnly, "Error")
  Exit Sub
End If
If lBoatCount > lSailorCount Then
  Call MsgBox("Number of boats" & vbCrLf & "needs to be less or equal" & _
    vbCrLf & "to number of sailors!", vbOKOnly, "Error")
  Exit Sub
End If

Range("D:XFD").EntireColumn.Delete

ReDim lBestBoatInFlight(1 To lBoatCount, 1 To lFlightCount) As Long
For i = 1 To lSimulationCount
  ReDim lSailorInBoat(1 To lSailorCount, 1 To lBoatCount) As Long
  ReDim lSailorMeetSailor(1 To lSailorCount, 1 To lSailorCount) As Long
  ReDim lBoatInFlight(1 To lBoatCount, 1 To lFlightCount) As Long
  lAdjacentFlights = 0
  For j = 1 To lFlightCount
    ReDim lBoat(1 To lBoatCount) As Long
    For k = 1 To lBoatCount
      If lSailorIndex = 0 Then
        ReDim vSailor(1 To lSailorCount) As Variant
        vSailor = UniqRandInt(lSailorCount, lSailorCount)
        lSailorIndex = 1
      End If
      lBoat(k) = vSailor(lSailorIndex)
      lBoatInFlight(k, j) = vSailor(lSailorIndex)
      For m = 1 To k - 1
        lSailorMeetSailor(lBoat(k), lBoat(m)) = _
          lSailorMeetSailor(lBoat(k), lBoat(m)) + 1
        lSailorMeetSailor(lBoat(m), lBoat(k)) = _
          lSailorMeetSailor(lBoat(m), lBoat(k)) + 1
      Next m
      If j > 1 Then
        For m = 1 To lBoatCount
          If lBoatInFlight(k, j) = lBoatInFlight(m, j - 1) Then
            lAdjacentFlights = lAdjacentFlights + 1
          End If
        Next m
      End If
      lSailorInBoat(vSailor(lSailorIndex), k) = _
        lSailorInBoat(vSailor(lSailorIndex), k) + 1
      lSailorIndex = (lSailorIndex + 1) Mod (lSailorCount + 1)
    Next k
  Next j
  lMaxSailorMeetSailor = 0
  For j = 1 To lSailorCount - 1
    For m = j + 1 To lSailorCount
      If lMaxSailorMeetSailor < lSailorMeetSailor(j, m) Then
        lMaxSailorMeetSailor = lSailorMeetSailor(j, m)
      End If
    Next m
  Next j
  lMaxSailorInBoat = 0
  For j = 1 To lSailorCount
    For m = 1 To lBoatCount
      If lMaxSailorInBoat < lSailorInBoat(j, m) Then
        lMaxSailorInBoat = lSailorInBoat(j, m)
      End If
    Next m
  Next j
  If lBestSailorMeetSailor + lBestSailorInBoat + lLowestAdjacentFlights > _
    lMaxSailorMeetSailor + lMaxSailorInBoat + lAdjacentFlights Then
    For j = 1 To lBoatCount
      For m = 1 To lFlightCount
        lBestBoatInFlight(j, m) = lBoatInFlight(j, m)
      Next m
    Next j
    lBestSailorMeetSailor = lMaxSailorMeetSailor
    lBestSailorInBoat = lMaxSailorInBoat
    lLowestAdjacentFlights = lAdjacentFlights
  End If
Next i

For m = 1 To lFlightCount: ws.Cells(1, 4 + m) = "Flight " & m: Next m
For j = 1 To lBoatCount
  ws.Cells(1 + j, 4) = "Boat " & j
  For m = 1 To lFlightCount
    ws.Cells(1 + j, 4 + m) = sSailor(lBestBoatInFlight(j, m))
    #If I_Want_Colors Then
    k = (lBestBoatInFlight(j, m) Mod 56) + 1
    ws.Cells(1 + j, 4 + m).Interior.ColorIndex = k
```

```
    If xlFC(k) Then
      ws.Cells(1 + j, 4 + m).Font.ColorIndex = xlCIBlack
    Else
      ws.Cells(1 + j, 4 + m).Font.ColorIndex = xlCIWhite
    End If
    #End If
  Next m
Next j

ws.Cells(j + 1, 4) = "Maximal meet of sailor pairs"
ws.Cells(j + 1, 5) = lBestSailorMeetSailor
ws.Cells(j + 2, 4) = "Maximal repetition of boat per sailor"
ws.Cells(j + 2, 5) = lBestSailorInBoat
ws.Cells(j + 3, 4) = "Number of sailors with adjacent flights"
ws.Cells(j + 3, 5) = lLowestAdjacentFlights
Range("D:XFD").EntireColumn.AutoFit

End With
End Sub
```

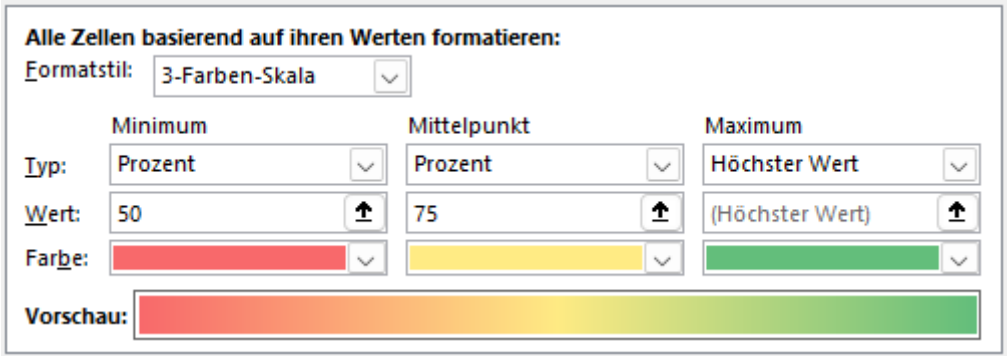
Chances at Board Game Risk

Do you know your chance of winning an attack with 15 armies on one of your countries against a neighboring defender with 11 armies? According to the old, original rules, you could attack with 14 of your 15 armies and would have about a 32% chance of winning, but under the new rules, the chance would be 79%: (see blue circles)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	Old Version: Both Attacker and defender roll up to 3 dice.																				
2	Attacker armies \ Defender armies	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
3	2	0,41	0,11	0,02	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
4	3	0,76	0,36	0,12	0,05	0,02	0,01	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
5	4	0,92	0,66	0,32	0,22	0,12	0,06	0,03	0,02	0,01	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
6	5	0,97	0,79	0,45	0,31	0,20	0,11	0,07	0,04	0,02	0,01	0,01	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
7	6	0,99	0,89	0,57	0,41	0,28	0,17	0,11	0,07	0,04	0,03	0,02	0,01	0,01	0,00	0,00	0,00	0,00	0,00	0,00	0,00
8	7	1,00	0,94	0,69	0,52	0,37	0,26	0,17	0,12	0,07	0,05	0,03	0,02	0,01	0,01	0,00	0,00	0,00	0,00	0,00	0,00
9	8	1,00	0,97	0,76	0,61	0,46	0,33	0,24	0,16	0,11	0,07	0,05	0,03	0,02	0,01	0,01	0,00	0,00	0,00	0,00	0,00
10	9	1,00	0,98	0,82	0,69	0,54	0,41	0,31	0,22	0,15	0,11	0,07	0,05	0,03	0,02	0,01	0,01	0,00	0,00	0,00	0,00
11	10	1,00	0,99	0,87	0,75	0,62	0,49	0,36	0,28	0,20	0,15	0,11	0,07	0,05	0,04	0,02	0,02	0,01	0,01	0,00	0,00
12	11	1,00	0,99	0,90	0,80	0,68	0,55	0,45	0,34	0,25	0,19	0,14	0,10	0,07	0,05	0,03	0,02	0,01	0,01	0,01	0,00
13	12	1,00	1,00	0,93	0,84	0,73	0,62	0,51	0,40	0,32	0,24	0,17	0,13	0,09	0,07	0,05	0,03	0,02	0,02	0,01	0,01
14	13	1,00	1,00	0,95	0,88	0,78	0,68	0,57	0,47	0,36	0,28	0,22	0,16	0,12	0,09	0,06	0,04	0,03	0,02	0,01	0,01
15	14	1,00	1,00	0,96	0,90	0,83	0,73	0,62	0,52	0,43	0,34	0,26	0,20	0,16	0,11	0,08	0,06	0,04	0,03	0,02	0,01
16	15	1,00	1,00	0,97	0,93	0,86	0,77	0,68	0,58	0,48	0,39	0,32	0,25	0,19	0,15	0,11	0,08	0,05	0,04	0,03	0,02
17	16	1,00	1,00	0,98	0,94	0,89	0,81	0,72	0,63	0,54	0,44	0,37	0,29	0,24	0,18	0,13	0,10	0,07	0,06	0,04	0,03
18	17	1,00	1,00	0,98	0,96	0,90	0,85	0,78	0,68	0,58	0,50	0,42	0,34	0,27	0,22	0,17	0,13	0,10	0,07	0,05	0,04
19	18	1,00	1,00	0,99	0,97	0,93	0,88	0,80	0,73	0,64	0,55	0,47	0,39	0,31	0,26	0,21	0,15	0,12	0,09	0,07	0,05
20	19	1,00	1,00	0,99	0,98	0,94	0,89	0,83	0,76	0,68	0,60	0,52	0,44	0,35	0,30	0,23	0,19	0,15	0,12	0,08	0,06
21	20	1,00	1,00	1,00	0,98	0,96	0,91	0,86	0,80	0,71	0,64	0,56	0,50	0,41	0,34	0,28	0,22	0,17	0,14	0,11	0,08
22																					
23	New Version: Attacker rolls up to 3 dice, defender only up to 2.																				
24	Attacker armies \ Defender armies	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
25	2	0,41	0,10	0,03	0,01	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
26	3	0,75	0,36	0,21	0,09	0,05	0,02	0,01	0,01	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
27	4	0,92	0,67	0,47	0,32	0,21	0,13	0,08	0,05	0,03	0,02	0,01	0,01	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
28	5	0,97	0,78	0,65	0,46	0,36	0,26	0,18	0,13	0,09	0,06	0,04	0,03	0,02	0,01	0,01	0,00	0,00	0,00	0,00	0,00
29	6	0,99	0,88	0,78	0,64	0,51	0,39	0,29	0,23	0,17	0,11	0,08	0,06	0,04	0,03	0,02	0,01	0,01	0,01	0,00	0,00
30	7	1,00	0,94	0,85	0,74	0,64	0,52	0,43	0,32	0,26	0,18	0,15	0,11	0,08	0,06	0,04	0,03	0,02	0,02	0,01	0,01
31	8	1,00	0,97	0,91	0,83	0,74	0,64	0,54	0,45	0,35	0,28	0,22	0,17	0,13	0,10	0,07	0,05	0,04	0,03	0,02	0,02
32	9	1,00	0,98	0,95	0,89	0,82	0,73	0,65	0,55	0,45	0,38	0,31	0,24	0,19	0,15	0,12	0,09	0,07	0,06	0,04	0,03
33	10	1,00	0,99	0,97	0,93	0,87	0,81	0,73	0,65	0,55	0,48	0,40	0,33	0,27	0,22	0,17	0,14	0,10	0,08	0,06	0,05
34	11	1,00	0,99	0,98	0,95	0,92	0,86	0,80	0,73	0,64	0,56	0,50	0,42	0,35	0,29	0,24	0,19	0,15	0,11	0,10	0,08
35	12	1,00	1,00	0,99	0,97	0,94	0,91	0,85	0,80	0,73	0,64	0,58	0,50	0,43	0,38	0,31	0,25	0,21	0,17	0,14	0,11
36	13	1,00	1,00	0,99	0,98	0,96	0,93	0,90	0,85	0,79	0,72	0,65	0,59	0,51	0,45	0,38	0,33	0,28	0,23	0,19	0,15
37	14	1,00	1,00	1,00	0,99	0,98	0,95	0,92	0,89	0,84	0,79	0,72	0,66	0,59	0,53	0,47	0,40	0,34	0,29	0,25	0,21
38	15	1,00	1,00	1,00	0,99	0,99	0,97	0,95	0,91	0,88	0,83	0,79	0,72	0,66	0,60	0,54	0,47	0,41	0,37	0,31	0,25
39	16	1,00	1,00	1,00	1,00	0,99	0,98	0,97	0,94	0,91	0,88	0,83	0,78	0,72	0,67	0,60	0,55	0,48	0,43	0,37	0,32
40	17	1,00	1,00	1,00	1,00	0,99	0,99	0,98	0,96	0,93	0,90	0,87	0,83	0,78	0,73	0,67	0,62	0,56	0,49	0,44	0,39
41	18	1,00	1,00	1,00	1,00	1,00	0,99	0,98	0,97	0,95	0,93	0,90	0,87	0,82	0,78	0,73	0,67	0,61	0,57	0,50	0,45
42	19	1,00	1,00	1,00	1,00	1,00	0,99	0,99	0,98	0,97	0,95	0,92	0,90	0,87	0,82	0,78	0,73	0,68	0,62	0,57	0,51
43	20	1,00	1,00	1,00	1,00	1,00	1,00	0,99	0,99	0,98	0,97	0,94	0,92	0,89	0,86	0,82	0,78	0,73	0,68	0,62	0,57

Run Simulations

A conditional format colors the cells red for chances of 50% or less, a yellow background indicates chances between 50% and 75%, and green cell colors show chances of 75% or higher:



Theoretically, both tables should be identical for the first 2 columns. Small differences are caused by the "incomplete" randomness of the finite Monte Carlo simulation with 10,000 runs.

Game of Risk Program Code

```
Const GCMonteCarloRuns = 10000

Sub Schedule()
'Calculate chances for an attacker at the game of risk for both the original
'version (both attacker and defender roll up to 3 dice) and the new version
'(attacker rolls up to 3 dice, defender only up to 2).
'Calls parametrized sub Calculate_Chances twice.
'(C) (P) by Bernd Plumhoff 30-Sep-2012 PB V0.1
Dim ws As Worksheet
Dim cPerf As clsPerf 'See: https://jkp-ads.com/Articles/performanceclass.asp

'Include SystemState class from https://sulprobil.com/html/systemstate.html
Dim state As SystemState
If gbDebug Then
  Set cPerf = New clsPerf
  cPerf.SetRoutine "Schedule"
End If
Application.StatusBar = False
Set state = New SystemState

'Preparation
Set ws = Sheets("Chances")
ws.Cells.ClearContents

Call Calculate_Chances("Old Version: Both Attacker and defender roll up to" & _
  " 3 dice.", 1, 3)
Call Calculate_Chances("New Version: Attacker rolls up to 3 dice, defender" & _
  " only up to 2.", 23, 2)

Call ReportPerformance

End Sub

Sub Calculate_Chances(sTitle As String, _
    lOutputRow As Long, _
    lMaxDefenderArmies As Long)
'Calculate chances for an attacker at the game of risk.
'This sub calculates the chances for a matrix of 2 to 20 attacking armies
'against 1 to 20 defending armies.
'Reverse(moc.liborplus.www) V0.1 30-Sep-2012
Dim i As Long
Dim j As Long
Dim k As Long
Dim m As Long
Dim lAttackerDice As Long
Dim lAttackerThrow As Long
Dim lAttackerResult(1 To 3) As Long
Dim lAttackerWins As Long
Dim lDefenderDice As Long
Dim lDefenderThrow As Long
Dim lDefenderResult(1 To 3) As Long
Dim ws As Worksheet
Dim cPerf As clsPerf 'See: https://jkp-ads.com/Articles/performanceclass.asp

'Include SystemState class from https://sulprobil.com/html/systemstate.html
Dim state As SystemState
If gbDebug Then
  Set cPerf = New clsPerf
  cPerf.SetRoutine "Calculate_Chances"
End If
```

```
Application.StatusBar = False
Set state = New SystemState

With Application.WorksheetFunction
'Preparation
Set ws = Sheets("Chances")
ws.Cells(lOutputRow, 1) = sTitle
ws.Cells(lOutputRow + 1, 1) = "Attacker armies \ Defender armies"
For i = 2 To 20
  Application.StatusBar = "Calculating " & i & " attackers for " & sTitle
  For j = 1 To 20
    ws.Cells(i + lOutputRow, 1) = i
    ws.Cells(1 + lOutputRow, j + 1) = j
    lAttackerWins = 0
    For k = 1 To GCMonteCarloRuns
      lAttackerDice = i - 1 'One army needs to occupy the land and
                            'cannot be used to attack
      lDefenderDice = j
      Do While lAttackerDice > 0 And lDefenderDice > 0
        lAttackerThrow = lAttackerDice
        If lAttackerThrow > 3 Then lAttackerThrow = 3
        lDefenderThrow = lDefenderDice
        If lDefenderThrow > lMaxDefenderArmies Then
          lDefenderThrow = lMaxDefenderArmies
        End If
        'Roll the dice
        For m = 2 To 3
          lAttackerResult(m) = 0
          lDefenderResult(m) = 0
        Next m
        For m = 1 To lAttackerThrow
          lAttackerResult(m) = Int(1 + Rnd * 6)
        Next m
        For m = 1 To lDefenderThrow
          lDefenderResult(m) = Int(1 + Rnd * 6)
        Next m
        'Sort results
        If lAttackerResult(1) < lAttackerResult(2) Then
          If lAttackerResult(1) < lAttackerResult(3) Then
              If lAttackerResult(2) < lAttackerResult(3) Then
                  '3-2-1
                  m = lAttackerResult(1)
                  lAttackerResult(1) = lAttackerResult(3)
                  lAttackerResult(3) = m
              Else
                  '2-3-1
                   m = lAttackerResult(1)
                  lAttackerResult(1) = lAttackerResult(2)
                  lAttackerResult(2) = lAttackerResult(3)
                  lAttackerResult(3) = m
              End If
          Else
              '2-1-3
              m = lAttackerResult(1)
              lAttackerResult(1) = lAttackerResult(2)
              lAttackerResult(2) = m
          End If
        Else
          If lAttackerResult(1) < lAttackerResult(3) Then
              If lAttackerResult(2) < lAttackerResult(3) Then
                  '3-1-2
                  m = lAttackerResult(1)
                  lAttackerResult(1) = lAttackerResult(3)
                  lAttackerResult(3) = lAttackerResult(2)
                  lAttackerResult(2) = m
              End If
          Else
              If lAttackerResult(2) < lAttackerResult(3) Then
                  '1-3-2
                  m = lAttackerResult(2)
                  lAttackerResult(2) = lAttackerResult(3)
                  lAttackerResult(3) = m
              End If
          End If
        End If
        If lDefenderResult(1) < lDefenderResult(2) Then
          If lDefenderResult(1) < lDefenderResult(3) Then
            If lDefenderResult(2) < lDefenderResult(3) Then
              '3-2-1
              m = lDefenderResult(1)
              lDefenderResult(1) = lDefenderResult(3)
              lDefenderResult(3) = m
            Else
              '2-3-1
               m = lDefenderResult(1)
              lDefenderResult(1) = lDefenderResult(2)
              lDefenderResult(2) = lDefenderResult(3)
              lDefenderResult(3) = m
            End If
```

```
          Else
            '2-1-3
            m = lDefenderResult(1)
            lDefenderResult(1) = lDefenderResult(2)
            lDefenderResult(2) = m
          End If
        Else
          If lDefenderResult(1) < lDefenderResult(3) Then
            If lDefenderResult(2) < lDefenderResult(3) Then
              '3-1-2
              m = lDefenderResult(1)
              lDefenderResult(1) = lDefenderResult(3)
              lDefenderResult(3) = lDefenderResult(2)
              lDefenderResult(2) = m
            End If
          Else
            If lDefenderResult(2) < lDefenderResult(3) Then
              '1-3-2
              m = lDefenderResult(2)
              lDefenderResult(2) = lDefenderResult(3)
              lDefenderResult(3) = m
            End If
          End If
        End If
        'Analyze result and reduce armies
        For m = 1 To 3
          If lAttackerResult(m) > 0 And lDefenderResult(m) > 0 Then
            If lAttackerResult(m) > lDefenderResult(m) Then
              lDefenderDice = lDefenderDice - 1
            Else
              lAttackerDice = lAttackerDice - 1
            End If
          Else
            Exit For
          End If
        Next m
      Loop
      If lAttackerDice > 0 Then
        lAttackerWins = lAttackerWins + 1
      End If
    Next k
    ws.Cells(i + lOutputRow, j + 1) = lAttackerWins / GCMonteCarloRuns
  Next j
Next i
End With
End Sub
```

A Simple Monte Carlo Simulation

With Excel/VBA, it's fairly easy to create a Monte Carlo simulation, but you should avoid some potential pitfalls:

- Use the SystemState class to speed up the program by turning off ScreenUpdating and setting Calculation to xlCalculationManual.
- Keep the user informed about the progress of the simulation.
- Handle unknown dimensional growth efficiently during the program.
- Be aware that Excel is generally not the best (or the fastest) simulation tool.

	A	B	C	D	E	F	G
1	Number of Simulations	2.000.000		3 showed up after this many throws	How often	Theoretical Value (rounded)	
2				1	199029	200000	
3	Start Simulation			2	180354	180000	
4				3	162605	162000	
5				4	145455	145800	
6				5	131762	131220	
7				6	118324	118098	
8				7	106100	106288	
9				8	95572	95659	
10				9	85749	86094	
11				10	77749	77484	
12				11	69962	69736	
13				12	62811	62762	

Literature

If Excel is not too slow for your purposes it seems to be usable since Excel 2010:

Alexei Botchkarev, Assessing Excel VBA Suitability for Monte Carlo Simulation, https://arxiv.org/ftp/arxiv/papers/1503/1503.08376.pdf

sbMontaCarloSimulation Program Code

```
Sub Simulate()
'Creates a simple Monte Carlo simulation by counting how long
'it takes to throw a 3 with a die with 10 surfaces (likelihood
'for each to show is 1/10).
'(C) (P) by Bernd Plumhoff  23-Nov-2022 PB V0.2
Dim i                     As Long
Dim lSimulations          As Long
Dim lTries                As Long

Dim state                 As SystemState

With Application.WorksheetFunction
Set state = New SystemState
Randomize
lSimulations = Range("Simulations")
ReDim lResult(1 To 1) As Long 'Error Handler will increase as needed
On Error GoTo ErrHdl
For i = 1 To lSimulations
    If i Mod 10000 = 1 Then Application.StatusBar = "Simulation " & _
        Format(i, "#,##0") 'Inform the user that program is still alive
    lTries = 0
    Do
        lTries = lTries + 1
    Loop Until .RandBetween(1, 10) = 3 'This is the simulation
    lResult(lTries) = lResult(lTries) + 1
Next i
On Error GoTo 0
Range("D:F").ClearContents
Range("D1:F1").FormulaArray = Array("3 showed up after this many throws", _
    "How often", "Theoretical Value (rounded)")
For i = 1 To UBound(lResult)
    Cells(i + 1, 4) = i
    Cells(i + 1, 5) = lResult(i)
    lTries = .Round(lSimulations * 0.1, 0)
    Cells(i + 1, 6) = lTries
    lSimulations = lSimulations - lTries
Next i
End With
Exit Sub

ErrHdl:
If Err.Number = 9 Then
   'Here we normally get if we breach Ubound(lResult)
   If lTries > UBound(lResult) Then
       'So we need to increase dimension
       ReDim Preserve lResult(1 To lTries)
       Resume 'Back to statement which caused error
   End If
End If
'Other error - terminate
On Error GoTo 0
Resume
End Sub
```

Random Floating Point Numbers

Generate an Ideal Normal Distribution – *sbGenNormDist*

To generate a standard normal distribution, you typically use *=NORM.S.INV(RAND())*. So, if you need a standard normal distribution with a mean of 7 and a standard deviation of 10, you would use *=NORM.INV(RAND(),7,10)*.

However, your normal distribution will never have exactly a mean of 7 (and never exactly a standard deviation of 10) unless the number of random numbers approaches infinity. If you want to achieve exactly a mean of 7 and exactly a standard deviation of 10, you need to shift the generated set of random numbers to the mean and then scale them to the desired standard deviation. You can accomplish this in at least three different ways:

	A	B	C	D	E	F
1	**How to create a series of random numbers with given mean M and standard deviation S**					
3				**Approach with Worksheet Functions**		
5				**Formulae in column C**		**Formulae in column E**
6	**Mean M**	7	8,428432	=AVERAGE(C8:C17)	7	=AVERAGE(E8:E17)
7	**StDev S**	10	11,96582	=STDEV.S(C8:C17)	10	=STDEV.S(E8:E17)
8			21,94291	=NORM.INV(RAND(),$B$6,$B$7)	18,29423795	=$B$6+(C8-$C$6)*$B$7/$C$7
9			24,43755		20,3790441	
10			7,353518		6,101679564	
11			13,54982	1. Step of 1. Solution	11,28001202	1. Solution
12		Data	5,04178		4,169728014	
13			-19,81613		-16,6043719	
14			5,360509		4,436093864	
15			9,224908		7,665625876	
16			7,373342		6,118246906	
17			9,816112		8,159703605	
19				**Approach with VBA**		
21				**Formulae in column C**		**Formulae in column E**
22			7	=AVERAGE(C24:C33)	7	=AVERAGE(E24:E33)
23			10	=STDEV.S(C24:C33)	10	=STDEV.S(E24:E33)
24			9,625732	=sbGenNormDist(10,B6,B7)	20,39189782	20.3918978179763
25			9,425687		24,68205067	
26			2,464031		9,953757058	Generate Random Number **CONSTANTS**
27			21,88859		8,801365433	
28			13,49523		10,42167346	
29			-7,123743	2. Solution	-2,08459217	3. Solution
30			5,267867		1,329862782	
31			20,45142		3,820594747	
32			-6,577798		-7,67930564	
33			1,082993		0,362695834	

sbGenNormDist Program Code

```
Function sbGenNormDist(lCount As Long, _
    dMean As Double, _
    dStDev As Double) As Variant
'Generates lCount normally distributed random values
'with mean dMean and standard deviation dStDev.
'(C) (P) by Bernd Plumhoff 30-May-2024 V0.3
Dim i As Long
Dim dSampleMean As Double, dSampleStDev As Double

If lCount < 2 Then
  sbGenNormDist = CVErr(xlErrValue)
  Exit Function
End If
ReDim vR(1 To lCount) As Variant
With Application
For i = 1 To lCount
  vR(i) = .Norm_Inv(Rnd(), dMean, dStDev)
Next i
dSampleMean = .Average(vR)
dSampleStDev = .StDev_S(vR)
For i = 1 To lCount
  vR(i) = dMean + (vR(i) - dSampleMean) * dStDev / dSampleStDev
Next i
sbGenNormDist = .Transpose(vR)
End With
End Function
```

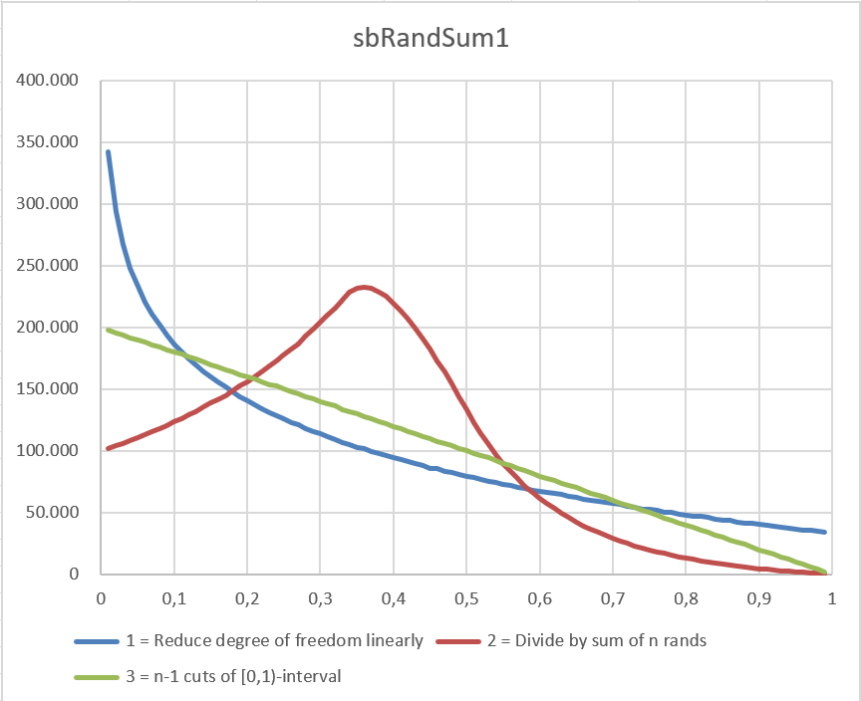
Generate Random Numbers with a Sum of 1 – *sbRandSum1*

We generate *n* random numbers with one condition: The sum of all the generated numbers must equal 1. This can be achieved using several different approaches.

Three possible approaches are:

- Gradually reduce the degrees of freedom: Generate the first random number, then the second within the range [0, 1 - First_Number), then the third within [0, 1 - First_Number - Second_Number), …, and the last number must eventually be equal to 1 - Sum_of_all_other_numbers.
- Generate n random numbers and divide them by their sum.
- Simulate dividing a cake: wherever you cut, you can't distribute more or less than the whole cake.

The resulting distributions look like this:



You can easily see that the commonly used approach of dividing n random numbers by their sum is a poor choice: you typically get numbers between 0.2 and 0.5 (see the red curve).

Note: A general approach would be the Dirichlet distribution. For an implementation in Python, see numpy — for our above output, the *size* parameter should be set to 1: https://numpy.org/doc/stable/reference/random/generated/numpy.random.dirichlet.html?highlight=dirichlet#numpy.random.dirichlet

sbRandSum1 Program Code

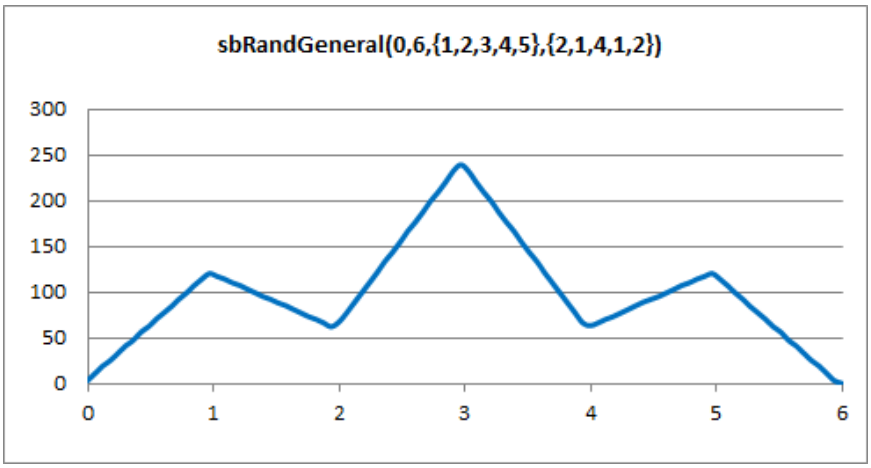
```
Function sbRandSum1(ByVal lDist As Long, Optional ByVal lCount As Long, _
    Optional bVolatile As Boolean = False) As Variant
'sbRandSum1 produces lCount (or the number of selected cells if
'called from a worksheet range) random numbers which sum up to 1.
'Possible values of lDist to specify desired distribution:
'        1 = reduce degree of freedom linearly
'        2 = divide lCount random numbers by their sum
'        3 = lCount-1 random cuts of (0,1)-interval
'If TypeName(Application.Caller) <> "Range" Then lCount has to be set.
'It specifies the count of summands which have to have the sum of 1.
'(C) (P) by Bernd Plumhoff 02-Aug-2020 PB V0.4
Static bRandomized As Boolean
Dim bRowWise As Boolean, vA As Variant, vT  As Variant
Dim i As Long, j As Long, dSum As Double

If bVolatile Then Application.Volatile
If Not bRandomized Then Randomize: bRandomized = True
If TypeName(Application.Caller) <> "Range" Then
  If lCount < 1 Then
    sbRandSum1 = CVErr(xlErrRef)
    Exit Function
  End If
  bRowWise = False
Else
  With Application.Caller
    lCount = .Rows.Count
    bRowWise = True
    If lCount < .Columns.Count Then
      lCount = .Columns.Count
      bRowWise = False
    End If
    If lCount = 1 Then
      sbRandSum1 = 1
      Exit Function
    End If
  End With
End If
ReDim vA(1 To lCount) As Variant
Select Case lDist
  Case 1
    ReDim nRand(1 To lCount) As Long
    For i = 1 To lCount
      nRand(i) = i
    Next i
    For i = 1 To lCount - 1
      j = Int(Rnd * (lCount - i + 1)) + i
      vA(nRand(j)) = Rnd * (1# - dSum)
      dSum = dSum + vA(nRand(j))
      nRand(j) = nRand(i)
    Next i
    vA(nRand(lCount)) = 1# - dSum
  Case 2
    For i = 1 To lCount
      vA(i) = Rnd
      dSum = dSum + vA(i)
    Next i
    For i = 1 To lCount
      vA(i) = vA(i) / dSum
    Next i
  Case 3
    For i = 1 To lCount - 1
      vA(i) = Rnd
      j = i - 1
      Do While j > 0
        If vA(j) > vA(j + 1) Then
          vT = vA(j + 1)
          vA(j + 1) = vA(j)
          vA(j) = vT
        End If
        j = j - 1
      Loop
    Next i
    vA(lCount) = 1# - vA(lCount - 1)
    i = lCount - 1
    Do While i > 1
      vA(i) = vA(i) - vA(i - 1)
      i = i - 1
    Loop
  Case Else
    sbRandSum1 = CVErr(xlErrValue)
    Exit Function
End Select
If bRowWise Then vA = Application.WorksheetFunction.Transpose(vA)
sbRandSum1 = vA
End Function
```

Distributions of Random Floating Point Numbers

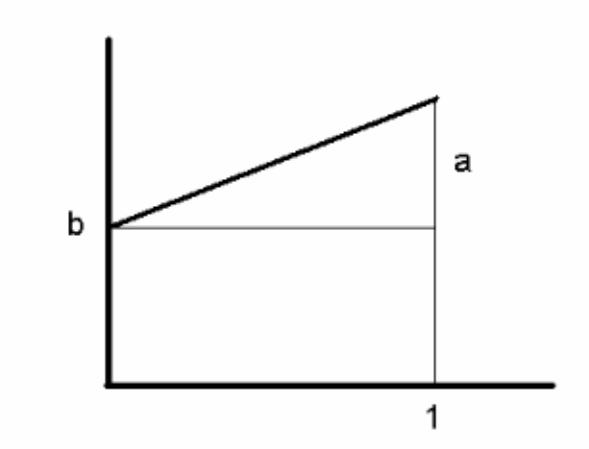
sbRandGeneral

If you need a stepwise linear distribution of random numbers, I recommend my custom function *sbRandGeneral*.

Note: Any distribution can be approximated with a stepwise linear distribution, like the one offered here, to a specified minimum accuracy.



Derivation of the Algorithm



Given: Values a and b.

Needed: Inverse function of area below $f(x) = ax + b$.

Function F for area below f(x): $F(x) = \frac{a}{2}x^2 + bx$

$\Leftrightarrow$ [a≠0]

$$\frac{2}{a}F(x) = x^2 + \frac{2b}{a}x + \frac{b^2}{a^2} - \frac{b^2}{a^2} = \left(x + \frac{b}{a}\right)^2 - \frac{b^2}{a^2}$$

$\Leftrightarrow$

$$x = -\frac{b}{a} \pm \sqrt{\frac{2}{a}F(x) + \frac{b^2}{a^2}}$$

$\Leftrightarrow$

$$G(x) = -\frac{b}{a} \pm \sqrt{\frac{2}{a}x + \frac{b^2}{a^2}}$$

With $b = w_i$ and $a = \frac{w_{i+1} - w_i}{x_{i+1} - x_i}$ we get

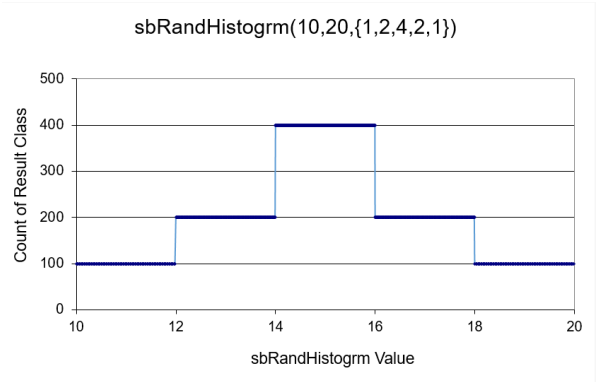
$$G(x) = -\frac{w_i(x_{i+1} - x_i)}{w_{i+1} - w_i} \pm \sqrt{\frac{2(x_{i+1} - x_i)}{w_{i+1} - w_i}x + \left(\frac{w_i(x_{i+1} - x_i)}{w_{i+1} - w_i}\right)^2}$$

sbRandGeneral Program Code

```
Function sbRandGeneral(dMin As Double, dMax As Double, vXi As Variant, _
    vWi As Variant, Optional dRandom As Double = 1#) As Double
'Generates a random number, General distributed.
'[see Vose: Risk Analysis, 2nd ed., p. 116]
'(C) (P) by Bernd Plumhoff  26-Jul-2020 PB V1.01
'Similar to @RISK's (C) RiskGeneral function.
Static bRandomized As Boolean
Dim i As Long, lWiCount As Long, lXiCount As Long
Dim dA As Double, dRand As Double, dSgn As Double

On Error GoTo ErrorLabelIsVariant
lXiCount = vXi.Count
lWiCount = vWi.Count
ErrorLabelWasVariant:
On Error GoTo 0
If lWiCount <> lXiCount Then
    sbRandGeneral = CVErr(xlErrValue)
    Exit Function
End If
If Not bRandomized Then Randomize: bRandomized = True
ReDim dX(0 To lXiCount + 1) As Double
ReDim dW(0 To lWiCount + 1) As Double

dX(0) = dMin
dX(UBound(dX)) = dMax
dW(0) = 0#
dW(UBound(dW)) = 0#
For i = 1 To lXiCount
    dX(i) = vXi(i)
    dW(i) = vWi(i)
Next i

'Calculate area
dA = 0#
For i = 0 To UBound(dX) - 1
    If dX(i) >= dX(i + 1) Or dW(i) < 0# Then
        sbRandGeneral = CVErr(xlErrValue)
        Exit Function
    End If
    dA = dA + (dX(i + 1) - dX(i)) * (dW(i + 1) + dW(i)) / 2#
Next i

'Normalise weights to set area to 1
For i = 1 To UBound(dW) - 1
    dW(i) = dW(i) / dA
Next i

ReDim dF(0 To UBound(dX)) As Double
'Calculate border points of value ranges for
'cumulative inverse function
dF(0) = 0#
dA = 0#
For i = 0 To UBound(dX) - 1
    dA = dA + (dX(i + 1) - dX(i)) * (dW(i + 1) + dW(i)) / 2#
    dF(i + 1) = dA
Next i
If dRandom = 1# Then
    dRand = Rnd()
Else
    dRand = dRandom
End If
i = 1
Do While dF(i) <= dRand
    i = i + 1
Loop
dSgn = Sgn(dW(i) - dW(i - 1))
If dSgn = 0# Then
    sbRandGeneral = dX(i - 1) + (dRand - dF(i - 1)) / _
                    (dF(i) - dF(i - 1)) * (dX(i) - dX(i - 1))
Else
    sbRandGeneral = dX(i - 1) + _
                    dSgn * Sqr((dRand - dF(i - 1)) * _
                    2# * (dX(i) - dX(i - 1)) / (dW(i) - dW(i - 1)) + _
                    (dW(i - 1) * (dX(i) - dX(i - 1)) / _
                    (dW(i) - dW(i - 1))) ^ 2#) - _
                    dW(i - 1) * (dX(i) - dX(i - 1)) / (dW(i) - dW(i - 1))
End If
Exit Function

ErrorLabelIsVariant:
lXiCount = UBound(vXi) - 1
lWiCount = UBound(vWi) - 1
Resume ErrorLabelWasVariant

End Function
```

sbRandHistogrm

A stepwise constant distribution is the histogram distribution:



sbRandHistogrm Program Code

```
Function sbRandHistogrm(dmin As Double, dMax As Double, _
            vWeight As Variant, Optional dRandom = 1#) As Double
'Specifies a histogram distribution with range dmin:dmax.
'This range is divided into vWeight.count classes. Each
'class has weight vWeight(i) reflecting the probability
'of occurrence of a value within the class.
'Similar to @Risk's function RiskHistogrm.
'(C) (P) by Bernd Plumhoff 18-Oct-2020 PB V1.01

Dim i As Long, n As Long, vW As Variant
Dim dRand As Double, dR As Double, dSumWeight As Double

With Application.WorksheetFunction
vW = .Transpose(.Transpose(vWeight))
End With

n = UBound(vW)
ReDim dSumWeightI(0 To n) As Double

dSumWeight = 0#
dSumWeightI(0) = 0#
For i = 1 To n
    If vW(i) < 0# Then 'A negative weight is an error
        sbRandHistogrm = CVErr(xlErrValue)
        Exit Function
    End If
    dSumWeight = dSumWeight + vW(i) 'Calculate sum of all weights
    dSumWeightI(i) = dSumWeight     'Calculate sum of weights till i
Next i

If dSumWeight = 0# Then  'Sum of weights has to be greater than zero
    sbRandHistogrm = CVErr(xlErrValue)
    Exit Function
End If

If dRandom = 1# Then
    dRand = Rnd()
Else
    dRand = dRandom
End If
dR = dSumWeight * dRand

i = n
Do While dR < dSumWeightI(i)
    i = i - 1
Loop

sbRandHistogrm = dmin + (dMax - dmin) * _
     (CDbl(i) + (dR - dSumWeightI(i)) / vW(i + 1)) / CDbl(n)

End Function
```

Similar distributions can be created by *rww* und *redw*:

rww Program Code

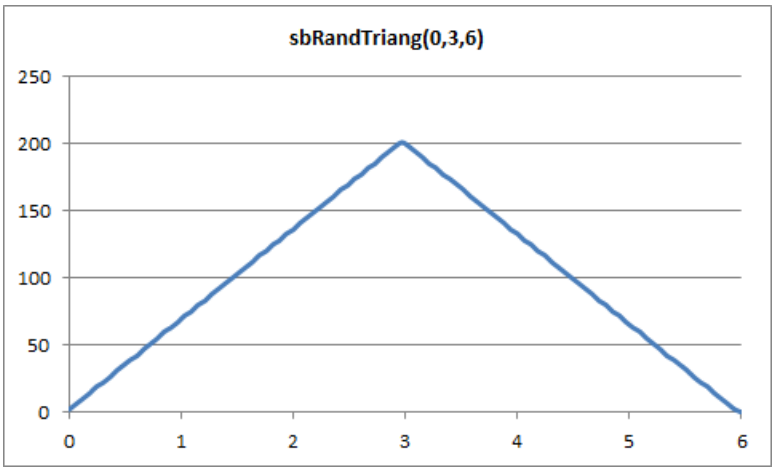
```
Function rww(ParamArray w() As Variant) As Double
'Produces random numbers with defined widths & weights
'Rww expects a vector of n random widths and weightings
'of type double and returns a random number of type double.
'This random number will lie in the given n-width-range of the
'(0,1)-interval with the given likelihood of the n weightings.
' Call Randomize before calling rww!
'(C) (P) by Bernd Plumhoff 06-Aug-2004 PB V0.50
'Examples:
'a) rww(1,0,1,1,8,0) will return a random number between 0.1 and 0.2
'b) rww(5,2,5,1) will return a random number between 0 and 0.5 twice as
'   often as a random number between 0.5 and 1.
'c) rww(1/3,0,1/3,1,1/3,0) will return a random number between
'   0.33333333333333 and 0.66666666666666.
'd) rww(5,15.4,3,7.7,2,0) would return a random value between
'   0 and 0.8, first 5 deciles with double likelihood than decile 6-8.

Dim i As Long
Dim swidths As Double
Dim sw As Double

If (UBound(w) + 1) Mod 2 <> 0 Then
  rww = -2     'No even number of arguments: Error
  Exit Function
End If

ReDim swidthsi(0 To (UBound(w) + 1) / 2 + 1) As Double
ReDim swi(0 To (UBound(w) + 1) / 2 + 1) As Double
ReDim weights(0 To (UBound(w) + 1) / 2) As Double
ReDim widths(0 To (UBound(w) + 1) / 2) As Double

swidths = 0#
sw = 0#
swi(0) = 0#
swidthsi(0) = 0#
For i = 0 To (UBound(w) - 1) / 2
  If w(2 * i) < 0# Then     'A negative width is an error
    rww = -3#
    Exit Function
  End If
  widths(i) = w(2 * i)
  swidths = swidths + widths(i)
  swidthsi(i + 1) = swidths
  If w(2 * i + 1) < 0# Then 'A negative weight is an error
    rww = -1#
    Exit Function
  End If
  weights(i) = w(2 * i + 1)
  If widths(i) > 0# Then
    sw = sw + weights(i)
  End If
  swi(i + 1) = sw
Next i
rww = sw * Rnd
'i = (UBound(w) - 1) / 2 + 1 'i already equals (UBound(w) - 1)/2 + 1, you may omit this statement.
Do While rww < swi(i)
  i = i - 1
Loop

rww = (swidthsi(i) + (rww - swi(i)) / weights(i) * widths(i)) / swidths

End Function
```

redw Program Code

```
Function redw(ParamArray vWeights() As Variant) As Double
'Produces random numbers with equidistant weights. Redw expects a vector of n random
'weights of type double and returns a random number of type double. This random
'number will lie in the given equidistant n-split-range of the [0,1)-interval with
'the given likelihood of weightings. Call Randomize before calling redw! '(C) (P) by Bernd Plumhoff 09-Dec-
2009 PB V0.50
Examples:
'a) redw(0,1,0,0,0,0,0,0,0,0) will return a random number d, 0.1 <= d < 0.2
'b) redw(2,1) will return a random number between 0 and 0.5 twice as
'   often as a random number between 0.5 and 1.
'c) redw(0,1,0) will return a random number d, 0.333333333333333 <= d < 0.666666666666666.
'd) redw(15.4,15.4,15.4,15.4,15.4,7.7,7.7,7.7,0,0) would return a random value between
'   0 and 0.8, first 5 deciles with double likelihood than decile 6-8.

Dim i As Long
Dim dw As Double
ReDim dwi(0 To UBound(vWeights) + 2) As Double

dw = 0#
dwi(0) = 0#
For i = 0 To UBound(vWeights)
  If vWeights(i) < 0# Then  'A negative weight is an error
    redw = CVErr(xlErrValue)
    Exit Function
  End If
  dw = dw + vWeights(i)      'Calculate sum of all weights
  dwi(i + 1) = dw            'Calculate sum of weights till i
Next i

redw = dw * Rnd

'i = UBound(vWeights) + 1 'i already equals UBound(vWeights) + 1, you may omit this statement.
Do While redw < dwi(i)
  i = i - 1
Loop

redw = (CDbl(i) + (redw - dwi(i)) / vWeights(i)) / (CDbl(UBound(vWeights) + 1))

End Function
```

sbRandTriang

The triangular distribution is a continuous probability distribution whose probability density function resembles a triangle. It is a simple distribution because you only need to know its minimum, median, and maximum:



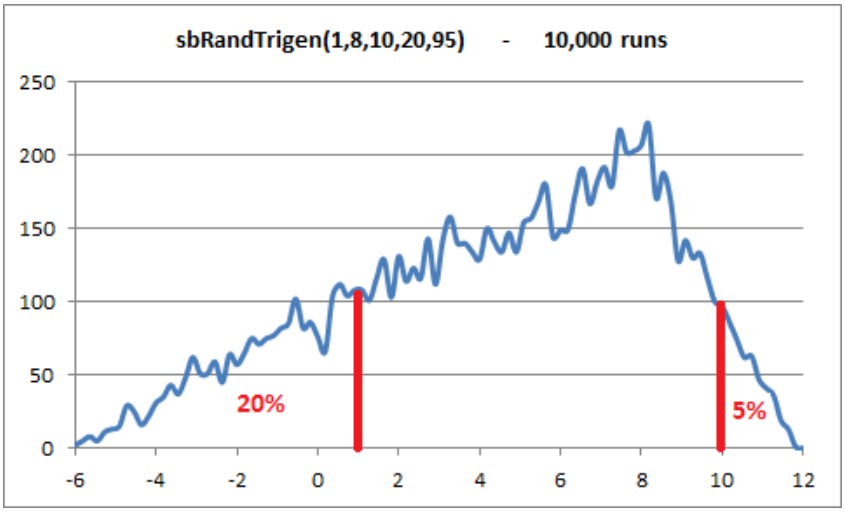
In the English-speaking world, this distribution is also referred to as the Distribution of Missing Data, because determining it requires only a minimal amount of information. This distribution is often used to simulate expert knowledge or when more accurate data collection is deemed too difficult or too expensive.

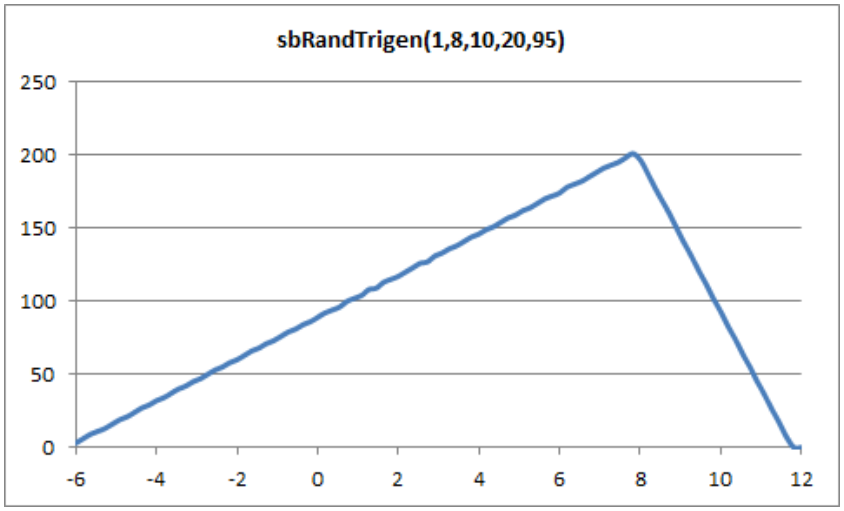
sbRandTriang Program Code

```
Function sbRandTriang(dMin As Double, dMode As Double, _
    dMax As Double, Optional dRandom = 1#) As Double
'Generates a random number, Triang distributed
'[see Vose: Risk Analysis, 2nd ed., p. 128]
'(C) (P) by Bernd Plumhoff 30-Aug-2024 PB V0.32
'Similar to @RISK's (C) RiskTriang function.
'sbRandTriang(minimum,mode,maximum) specifies a triangular
'distribution with three points – a minimum, a mode and
'a maximum. The skew of the triangular distribution is
'driven by the distance of the mode from the minimum and
'from the maximum. Reducing the distance from mode to
'minimum will increase the skew.
'Please ensure that you execute Randomize before you call
'this function for the first time.
Dim dRand As Double, dc_a As Double, db_a As Double
Dim dc_b As Double

If dMode < dMin Or dMax < dMode Then
  sbRandTriang = CVErr(xlErrValue)
  Exit Function
End If
If dMin = dMax Then
  sbRandTriang = dMin 'Triangle is just one point
  Exit Function
End If
dc_a = dMax - dMin
db_a = dMode - dMin
dc_b = dMax - dMode
If dRandom = 1# Then
  dRand = Rnd()
Else
  dRand = dRandom
End If
If dRand < db_a / dc_a Then
  sbRandTriang = dMin + Sqr(dRand * db_a * dc_a)
Else
  sbRandTriang = dMax - Sqr((1# - dRand) * dc_a * dc_b)
End If
End Function
```

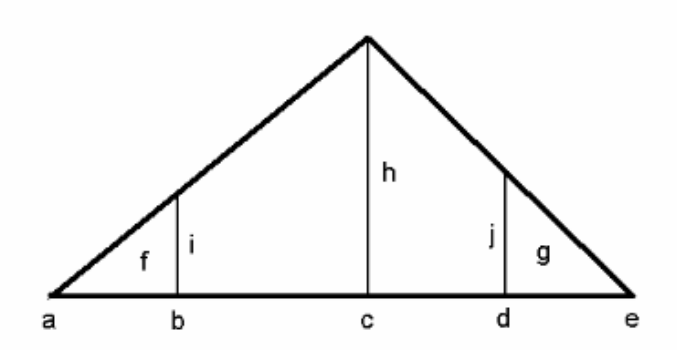
sbRandTrigen

sbRandTrigen defines a triangular distribution with three points: one with the highest probability and two others at the specified lower and upper percentiles. These percentile parameters have values between 0 and 100 and represent the cumulative probabilities for these lower and upper values. This is useful when the absolute smallest and largest values of the distribution are unknown or cannot be easily determined.





Derivation of the Algorithm



Given: Points b, c and d; Areas f and g. Area of the triangle is 1.

To be computed: Points a and e.

Triangle area is 1: [i] $h(e-a)=2 \Leftrightarrow h=\frac{2}{e-a}$

[ii] $\frac{h}{c-a}=\frac{i}{b-a} \Leftrightarrow i=h\frac{b-a}{c-a}=\frac{2(b-a)}{(e-a)(c-a)}$

[iii] $\frac{h}{e-c}=\frac{j}{e-d} \Leftrightarrow j=h\frac{e-d}{e-c}=\frac{2(e-d)}{(e-a)(e-c)}$

[iv] $2g=j(e-d)=\frac{2(e-d)^2}{(e-a)(e-c)} \Rightarrow a=e-\frac{(e-d)^2}{g(e-c)}$

[v] $2f=i(b-a)=\frac{2(b-a)^2}{(e-a)(c-a)}$

Substituting a in [v] and putting everything on one side gives:

$$\left\{b-e+\frac{(e-d)^2}{g(e-c)}\right\}^2-f\left\{e-e+\frac{(e-d)^2}{g(e-c)}\right\}\left\{c-e+\frac{(e-d)^2}{g(e-c)}\right\}=0$$

$\Leftrightarrow$

$$\frac{\left((b-e)g(e-c)+(e-d)^2\right)^2}{\left(g(e-c)\right)^2}-f\frac{(e-d)^2}{g(e-c)}\frac{(c-e)(g(e-c)+(e-d)^2}{g(e-c)}=0$$

$\Leftrightarrow$ [g≠0 and e≠c]

$$\left(g(b-c)(e-c)+(e-d)^2\right)^2 - f(e-d)^2\left((e-d)^2 - g(e-c)^2\right) = 0$$

$$\Leftrightarrow$$

$$g^2(b-e)^2(e-c)^2 + 2g(b-e)(e-c)(e-d)^2 + (e-d)^4 - f(e-d)^4 - fg(e-d)^2(e-c)^2 = 0$$

$$\Leftrightarrow$$

$$\begin{aligned}
&e^4(fg - f + 1 - 2g + g^2) \\
&+ e^3(-2fgd - 2fgc - 4d + 4fd + 2gc + 4gd + 2gb - 2g^2c - 2g^2b) \\
&+ e^2(fgd^2 + 4f\,\mathrm{gcd} + fgc^2 - 6fd^2 + 6d^2 - 4\mathrm{gcd} - 2gd^2 - 2gbc - 4gbd + g^2c^2 + 4g^2bc + g^2b^2) \\
&+ e(-2f\,\mathrm{gcd}^2 - 2fgc^2d + 4d^3f - 4d^3 + 2\mathrm{gcd}^2 + 4gbcd + 2gbd^2 - 2g^2bc^2 - 2g^2b^2c) \\
&+ (fgc^2d^2 - fd^4 + d^4 - 2gbcd^2 + g^2b^2c^2 = 0
\end{aligned}$$

The correct solution for e can now be calculated with a Newton iteration with a starting value > e, for example with the start value

$$d + \frac{d-c}{(1-g)^2}$$

If g=0 and f>0 then switch f and g, b and d and later the solution a and e.

sbRandTrigen Program Code

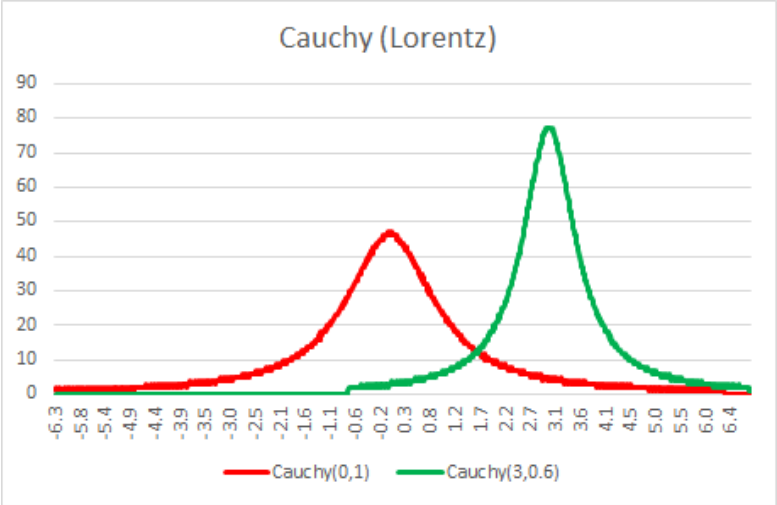
Please notice that *sbRandTrigen* requires (calls) *sbRandTriang*.

```
Function sbRandTrigen(dBottom As Double, dMode As Double, _
    dTop As Double, dBottomPerc As Double, _
    dTopPerc As Double, Optional dRandom = 1#) As Double
'Generates dMin random number, Triang distributed
'with given first and last decile
'[see Vose: Risk Analysis, 2nd ed., p. 129]
'(C) (P) by Bernd Plumhoff 19-Nov-2011 PB V0.32
'Similar to @RISK's (C) RiskTrigen function.
'sbRandTrigen(bottom, mode, top, bottom percentile, top percentile)
'specifies a triangular distribution with three points – one
'at the mode and two at the specified bottom and top percentiles.
'The bottom percentile and top percentile are values between
'0 and 100. Each percentile value gives the percentile of the
'total area under the triangle that is on the left side of the
'given point.
'Example:
'sbRandTrigen(1,8,10,20,95) will call
'sbRandTriang(-6.13212712795534, 8, 11.8648937411641).
'Please ensure that you execute Randomize before you call
'this function for the first time.

Static dBottomLast As Double
Static dModeLast As Double
Static dTopLast As Double
Static dBottomPercLast As Double
Static dTopPercLast As Double
Static dMin As Double
Static dMax As Double
Dim dMaxNew As Double
Dim da0 As Double, da1 As Double, da2 As Double
Dim da3 As Double, da4 As Double
```

```
Dim dfe As Double, df1e As Double
Dim dBottomPerc2 As Double, dTopPerc2 As Double
Dim i As Long

If dBottom = dBottomLast And dMode = dModeLast And dTop = dTopLast _
   And dBottomPerc = dBottomPercLast And dTopPerc = dTopPercLast _
   And Not IsError(dMin) Then
    sbRandTrigen = sbRandTriang(dMin, dMode, dMax, dRandom)
    Exit Function
End If

dBottomLast = dBottom
dModeLast = dMode
dTopLast = dTop
dBottomPercLast = dBottomPerc
dTopPercLast = dTopPerc

dBottomPerc2 = dBottomPerc / 100#
dTopPerc2 = 1# - dTopPerc / 100#
If dMode <= dBottom Or dTop <= dMode Then
    dMin = CVErr(xlErrValue) 'Trigger rerun next time
    sbRandTrigen = CVErr(xlErrValue)
    Exit Function
End If
If dBottomPerc2 < 0# Or dTopPerc2 < 0# Then
    dMin = CVErr(xlErrDiv0) 'Trigger rerun next time
    sbRandTrigen = CVErr(xlErrValue)
    Exit Function
End If

If dTopPerc2 = 0# Then
    If dBottomPerc2 = 0# Then
        sbRandTrigen = sbRandTriang(dBottom, dMode, dTop, dRandom)
        Exit Function
    End If
    sbRandTrigen = sbRandTrigen(dBottom, dMode, dTop, dBottomPerc2, dTopPerc2)
    Exit Function
End If

da4 = dBottomPerc2 * dTopPerc2 - dBottomPerc2 + 1# - 2# * dTopPerc2 + dTopPerc2 ^ 2#
da3 = -2# * dBottomPerc2 * dTopPerc2 * dTop - 2# * dBottomPerc2 * dTopPerc2 * dMode - _
      4# * dTop + 4# * dBottomPerc2 * dTop + 2# * dTopPerc2 * dMode + 4# * dTopPerc2 * _
      dTop + 2# * dTopPerc2 * dBottom - 2# * dTopPerc2 ^ 2# * dMode - _
      2# * dTopPerc2 ^ 2# * dBottom
da2 = dBottomPerc2 * dTopPerc2 * dTop ^ 2# + 4# * dBottomPerc2 * dTopPerc2 * dMode * _
      dTop + dBottomPerc2 * dTopPerc2 * dMode ^ 2# - 6# * dBottomPerc2 * dTop ^ 2# + _
      6# * dTop ^ 2# - 4# * dTopPerc2 * dMode * dTop - 2# * dTopPerc2 * dTop ^ 2# - 2# * _
      dTopPerc2 * dBottom * dMode - 4# * dTopPerc2 * dBottom * dTop + dTopPerc2 ^ 2# * _
      dMode ^ 2# + 4# * dTopPerc2 ^ 2# * dBottom * dMode + dTopPerc2 ^ 2# * dBottom ^ 2#
da1 = -2# * dBottomPerc2 * dTopPerc2 * dMode * dTop ^ 2# - 2# * dBottomPerc2 * dTopPerc2 * _
      dMode ^ 2# * dTop + 4# * dTop ^ 3# * dBottomPerc2 - 4# * dTop ^ 3# + 2# * dTopPerc2 * _
      dMode * dTop ^ 2# + 4# * dTopPerc2 * dBottom * dMode * dTop + 2# * dTopPerc2 * _
      dBottom * dTop ^ 2# - 2# * dTopPerc2 ^ 2# * dBottom * dMode ^ 2# - 2# * _
      dTopPerc2 ^ 2# * dBottom ^ 2# * dMode
da0 = dBottomPerc2 * dTopPerc2 * dMode ^ 2# * dTop ^ 2# - dBottomPerc2 * dTop ^ 4# + dTop ^ 4# - _
      2# * dTopPerc2 * dBottom * dMode * dTop ^ 2# + dTopPerc2 ^ 2# * dBottom ^ 2# * dMode ^ 2#

dMax = dTop + (dTop - dMode) / (1# - dTopPerc2) ^ 2#

'Newton iteration
Do While Abs(dMaxNew - dMax) > 0.000000000001

    i = i + 1
    If i > 30 Then
        If Abs(dfe) > 0.000000000001 Then
            dMin = CVErr(xlErrDiv0) 'Trigger rerun next time
            sbRandTrigen = CVErr(xlErrValue)
            Exit Function
        Else
            Exit Do
        End If
    End If
    dMaxNew = dMax
    dfe = da4 * dMaxNew ^ 4# + da3 * dMaxNew ^ 3# + da2 * dMaxNew ^ 2# + da1 * dMaxNew + da0
    df1e = 4# * da4 * dMaxNew ^ 3# + 3# * da3# * dMaxNew ^ 2# + 2# * da2 * dMaxNew + da1
    dMax = dMax - dfe / df1e

Loop

dMin = dMax - (dMax - dTop) ^ 2# / dTopPerc2 / (dMax - dMode)
sbRandTrigen = sbRandTriang(dMin, dMode, dMax, dRandom)
End Function
```

sbRandCauchy

If you want to simulate how particles hit a line starting from a fixed point, you can use a Cauchy distribution. This distribution is sometimes also called the Lorentz distribution. The quotient of two (0,1) normal distributions also results in a Cauchy distribution.



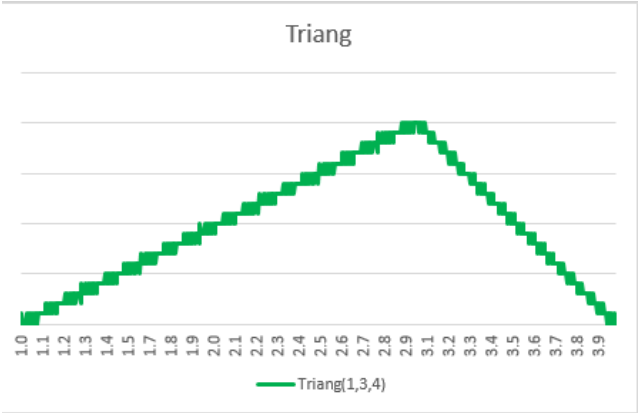
sbRandCauchy Program Code

```
Const GCPi = 3.14159265358979

Function sbRandCauchy(dLocation As Double, dScale As Double, _
  Optional dRandom = 1#) As Double
'(C) (P) by Bernd Plumhoff 03-Nov-2020 PB V0.2
Static bRandomized As Boolean
Dim dRand As Double
If dRandom < 0# Or dRandom > 1# Or dScale <= 0# Then
  sbRandCauchy = CVErr(xlErrValue)
  Exit Function
End If
If Not bRandomized Then
  Randomize
  bRandomized = True
End If
If dRandom = 1# Then
  dRand = Rnd()
Else
  dRand = dRandom
End If
sbRandCauchy = dLocation + dScale * Tan((dRand - 0.5) * GCPi)
End Function
```

sbRandCDFInv

You can easily generate random numbers with a desired distribution if the inverse distribution function is explicitly available.

An example of a stratified sample:



Without the explicitly available inverse distribution function, you can apply a linear approximation using the function *sbRandPDF*.

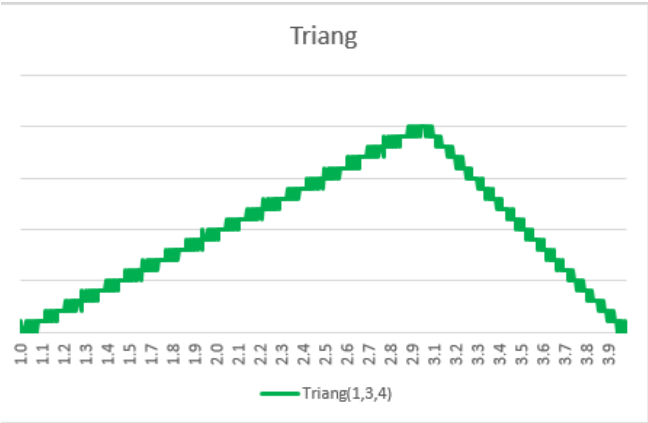
sbRandCFDInv Program Code

```
Function sbRandCDFInv(dParam1 As Double, dParam2 As Double, _
    dParam3 As Double, Optional dRandom = 1#) As Double
'(C) (P) by Bernd Plumhoff  03-Nov-2020 PB V0.2
Static bRandomized As Boolean
Dim dRand As Double
If dRandom < 0# Or dRandom > 1# Then
  sbRandCDFInv = CVErr(xlErrValue)
  Exit Function
End If
If Not bRandomized Then
  Randomize
  bRandomized = True
End If
If dRandom = 1# Then
    dRand = Rnd()
Else
    dRand = dRandom
End If
'Here you need to define the inverse of the cumulative distribution function
sbRandCDFInv = sbRandTriang(dParam1, dParam2, dParam3, dRand)
End Function
```

sbRandPDF

If an explicit form of the inverse distribution function is available, you should use *sbRandCDFInv*. However, if such a form is not available, you can use sbRandPDF with a linear approximation. Unfortunately, this approach is computationally intensive, even if you can reduce the number of points with identical or nearly identical slopes.

An example of a stratified sample:

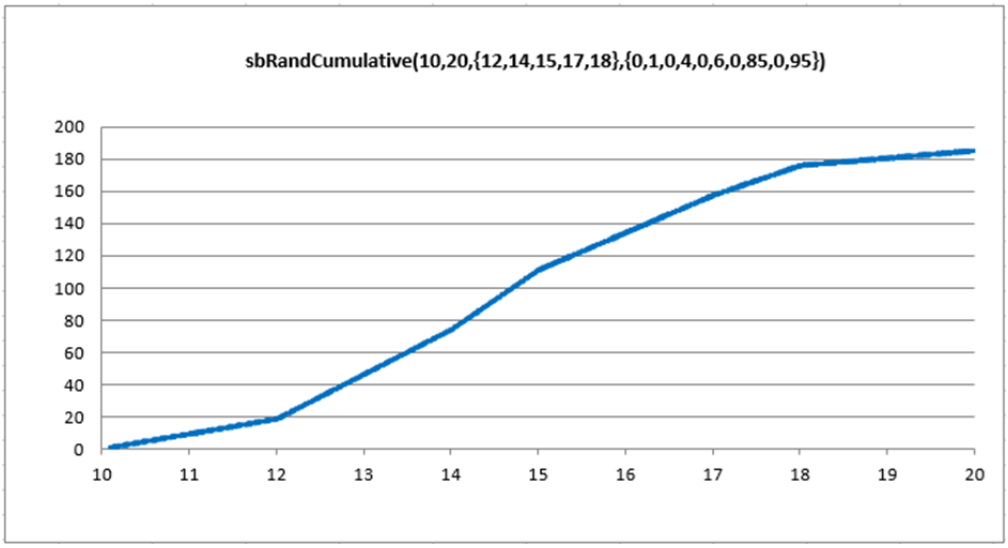


sbRandPDF Program Code

```
Function sbRandPDF(Optional dParam1, Optional dParam2, _
    Optional dParam3, Optional dRandom = 1#) As Double
'(C) (P) by Bernd Plumhoff  12-Sep-2014 PB V0.15
Dim dRand As Double
Dim i As Long
Static dPar1 As Double
Static dPar2 As Double
Static dPar3 As Double
Static vX(0 To 1000) As Variant
Static vY(0 To 1000) As Variant
If dRandom < 0# Or dRandom > 1# Then
  sbRandPDF = CVErr(xlErrValue)
  Exit Function
End If
If dRandom = 1# Then
  dRand = Rnd()
Else
  dRand = dRandom
End If
If dParam1 <> dPar1 Or dParam2 <> dPar2 Or dParam3 <> dPar3 Then
  dPar1 = dParam1
  dPar2 = dParam2
  dPar3 = dParam3
  'Initialize RandGeneral call parameters
  For i = 0 To 1000
    vX(i) = dPar1 + i * (dPar3 - dPar1) / 1000#
    'Now we can insert an arbitrary PDF function
    If vX(i) < dPar2 Then
      vY(i) = (vX(i) - dPar1) / ((dPar3 - dPar1) * (dPar2 - dPar1))
      If vY(i) < 0# Then vY(i) = 0#
    Else
      vY(i) = (dPar3 - vX(i)) / ((dPar3 - dPar1) * (dPar3 - dPar2))
      If vY(i) < 0# Then vY(i) = 0#
    End If
  Next i
End If
'Depending on the PDF input range you need to feed start
'and end values to sbRandGeneral
sbRandPDF = sbRandGeneral(dPar1, dPar3, vX, vY, dRand)
End Function
```

sbRandCumulative

If you need to generate a stepwise cumulative distribution function of random numbers, you can use the custom function sbRandCumulative. The derivation of the algorithm is analogous to *sbRandGeneral*.



sbRandCumulative Program Code

```
Function sbRandCumulative(dMin As Double, dMax As Double, _
    vXi As Variant, vWi As Variant, Optional dRandom = 1#) As Double
'Generates a random number, Cumulative distributed. [see Vose: Risk Analysis, 2nd ed., p. 109]
'(C) (P) by Bernd Plumhoff 23-Dec-2020 PB V0.50
'Similar to @RISK's (C) RiskCumulative function.
Static bRandomized As Boolean, i As Long
Dim dA As Double, dRand As Double, dSgn As Double

If vWi.Count <> vXi.Count Then sbRandCumulative = CVErr(xlErrValue): Exit Function
ReDim dX(0 To vXi.Count + 1) As Double
ReDim dW(0 To vWi.Count + 1) As Double

dX(0) = dMin
dX(UBound(dX)) = dMax
dW(0) = 0#
dW(UBound(dW)) = 1#
For i = 1 To vXi.Count
  dX(i) = vXi(i)
  dW(i) = vWi(i)
  If dW(i) < dW(i - 1) Then
    'Weights need to be monotonously increasing
    sbRandCumulative = CVErr(xlErrValue)
    Exit Function
  End If
Next i
If dW(UBound(dW)) < dW(UBound(dW) - 1) Then
  'Weights need to be monotonously increasing
  sbRandCumulative = CVErr(xlErrValue)
  Exit Function
End If

'Calculate area
dA = 0#
For i = 0 To UBound(dX) - 1
  If dX(i) >= dX(i + 1) Or dW(i) < 0# Then
    sbRandCumulative = CVErr(xlErrValue)
    Exit Function
  End If
  dA = dA + (dX(i + 1) - dX(i)) * (dW(i + 1) + dW(i)) / 2#
Next i

'Normalise weights to set area to 1
For i = 1 To UBound(dW)
  dW(i) = dW(i) / dA
Next i

ReDim dF(0 To UBound(dX)) As Double
'Calculate border points of value ranges for
'cumulative inverse function
dF(0) = 0#
dA = 0#
For i = 0 To UBound(dX) - 1
  dA = dA + (dX(i + 1) - dX(i)) * (dW(i + 1) + dW(i)) / 2#
  dF(i + 1) = dA
Next i

If dRandom = 1# Then
  If Not bRandomized Then
    Randomize
    bRandomized = True
  End If
  dRand = Rnd()
Else
  dRand = dRandom
End If

i = 1
Do While dF(i) <= dRand
  i = i + 1
Loop
dSgn = Sgn(dW(i) - dW(i - 1))
If dSgn = 0# Then
  sbRandCumulative = dX(i - 1) + (dRand - dF(i - 1)) / _
                 (dF(i) - dF(i - 1)) * (dX(i) - dX(i - 1))
Else
  sbRandCumulative = dX(i - 1) + _
                 dSgn * Sqr((dRand - dF(i - 1)) * _
                 2# * (dX(i) - dX(i - 1)) / (dW(i) - dW(i - 1)) + _
                 (dW(i - 1) * (dX(i) - dX(i - 1)) / _
                 (dW(i) - dW(i - 1))) ^ 2#) - _
                 dW(i - 1) * (dX(i) - dX(i - 1)) / (dW(i) - dW(i - 1))
End If

End Function
```

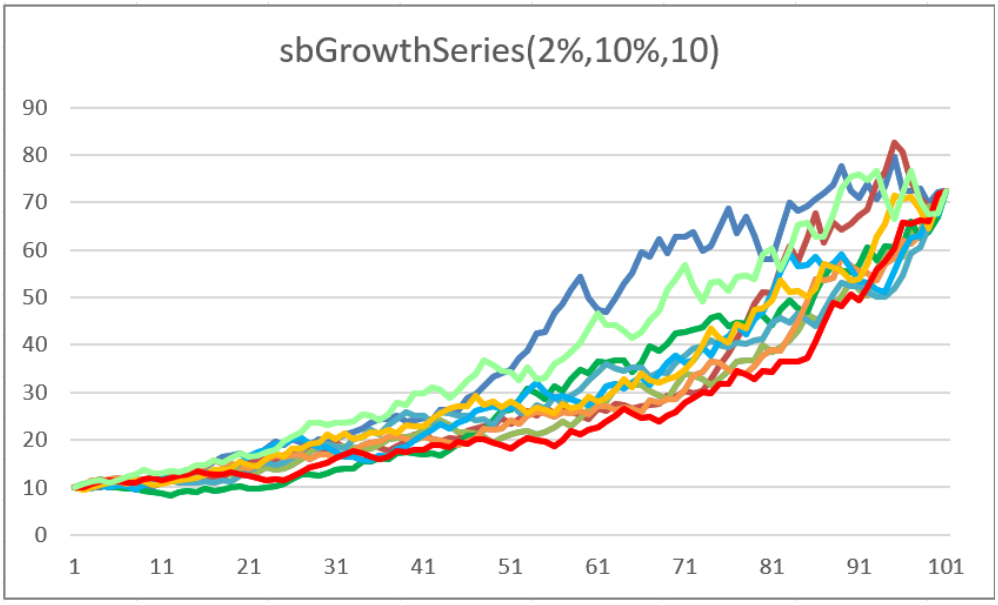
Brownian Bridges

A Brownian bridge is a Brownian (random) motion with a specified start and end value. It is used to model random developments of time series where the value is known at two points in time.

In addition to the examples presented here, *sbRandIntFixSum* can also be considered a Brownian bridge.

sbGrowthSeries

You can generate random numbers with a cumulative growth rate *dblRate*, a maximum relative change rate per time step *dblMaxRatePerStep*, and an optional starting value *dblStartVal*. The number of time steps (periods) is implicitly chosen by the number of selected cells, where the function call is entered as an array formula with CTRL + SHIFT + ENTER. This is a special type of Brownian bridge.



sbGrowthSeries Program Code

```
Function sbGrowthSeries(dblRate As Double, _
       dblMaxRatePerStep As Double, _
       Optional dblStartVal As Double = 1#) As Variant
'Returns random data with a compound growth rate dblRate, with
'a maximal relative change rate per step of dblMaxRatePerStep
'and with a start value dblStartVal. The number of periods
'is implicitly chosen by the number of selected cells which
'call this function as an array formula (entered with
'CTRL + SHIFT + ENTER). This is sort of a brownian bridge.
'(C) (P) by Bernd Plumhoff 20-Mar-2011 PB V0.91

Dim vR As Variant
Dim lP As Long 'Periods
Dim lrow As Long
Dim lcol As Long
Dim dblCurrVal As Double
Dim dblCurrRate As Double
Dim dblCurrMin As Double
Dim dblCurrMax As Double
Dim dblRelMin As Double
Dim dblRelMax As Double
Dim dblEndVal As Double

If TypeName(Application.Caller) <> "Range" Then
  sbGrowthSeries = CVErr(xlErrRef)
  Exit Function
End If

If Application.Caller.Rows.Count <> 1 And _
  Application.Caller.Columns.Count <> 1 Then
  sbGrowthSeries = CVErr(xlErrValue)
  Exit Function
End If

If Abs(dblRate) > dblMaxRatePerStep Then
  sbGrowthSeries = CVErr(xlErrNum)
  Exit Function
End If

lP = Application.Caller.Count

ReDim vR(1 To Application.Caller.Rows.Count, 1 To Application.Caller.Columns.Count)

dblCurrVal = dblStartVal
dblEndVal = dblStartVal * (1# + dblRate) ^ CDbl(lP)
dblCurrMin = dblEndVal / (1# + dblMaxRatePerStep) ^ CDbl(lP)
dblCurrMax = dblEndVal / (1# - dblMaxRatePerStep) ^ CDbl(lP)
For lrow = 1 To UBound(vR, 1)
  For lcol = 1 To UBound(vR, 2)
    dblCurrRate = (dblEndVal / dblCurrVal) ^ (1# / CDbl(lP - lcol * lrow + 1)) - 1#
    dblCurrMin = dblCurrMin * (1# + dblMaxRatePerStep)
    dblCurrMax = dblCurrMax * (1# - dblMaxRatePerStep)
    dblRelMin = (dblCurrMin - dblCurrVal) / dblCurrVal
    If dblRelMin < -dblMaxRatePerStep Then
      dblRelMin = -dblMaxRatePerStep
    End If
    dblRelMax = (dblCurrMax - dblCurrVal) / dblCurrVal
    If dblRelMax > dblMaxRatePerStep Then
      dblRelMax = dblMaxRatePerStep
    End If
    If dblCurrRate - dblRelMin < dblRelMax - dblCurrRate Then
      dblRelMax = 2# * dblCurrRate - dblRelMin
    Else
      dblRelMin = 2# * dblCurrRate - dblRelMax
    End If
    dblCurrVal = dblCurrVal * (1# + (dblRelMin + dblRelMax) / 2# + (Rnd() - 0.5) * (dblRelMax - dblRelMin))
    vR(lrow, lcol) = dblCurrVal
  Next lcol
Next lrow

sbGrowthSeries = vR

End Function
```

Fix Sum from Random Corridors

You need seven random numbers with different border values to add up to 100 exactly?

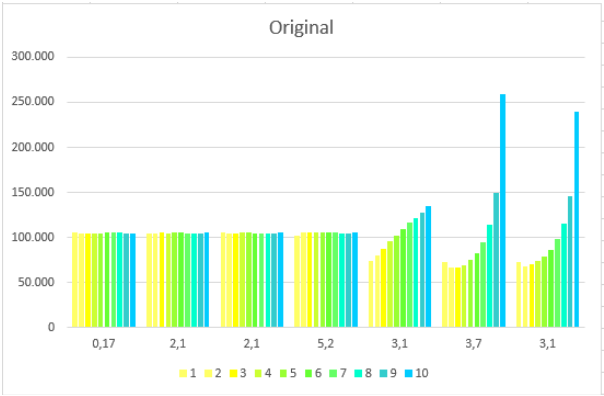
	A	B	C	D	E	F	G	H	I	J	K
1	**Target**	100								**Sum**	**Check**
2	**Lower Border**	0,27	2,8	15,3	43,3	15,1	9,8	2,7		89,27	
3	**Upper Border**	0,44	4,9	17,4	48,5	18,2	13,5	5,8		108,74	
4	**Formula Solutions:**										
5		0,40379	3,53798	15,4072	43,5003	18,1815	13,2235	5,74576		100	
6		0,27948	4,6466	16,2125	44,8473	15,4181	13,1083	5,48766		100	
7		0,34553	4,28888	17,014	48,303	15,4074	11,5856	3,05562		100	
8		0,29241	4,35964	16,7399	46,8495	15,5817	12,8168	3,36006		100	
9		0,28419	3,39074	16,4554	45,0109	15,7339	13,4882	5,63652		100	
10		0,42028	3,25492	16,5114	47,1417	16,2433	11,7482	4,68022		100	
11		0,2781	2,85025	17,396	48,1453	15,8712	12,7012	2,75799		100	
12		0,42567	4,38734	15,7825	47,3632	17,408	11,2474	3,386		100	
13		0,30344	4,15284	17,1746	45,0116	15,5844	12,6617	5,11157		100	
14		0,28599	3,86926	17,133	46,682	15,4564	10,8255	5,74789		100	

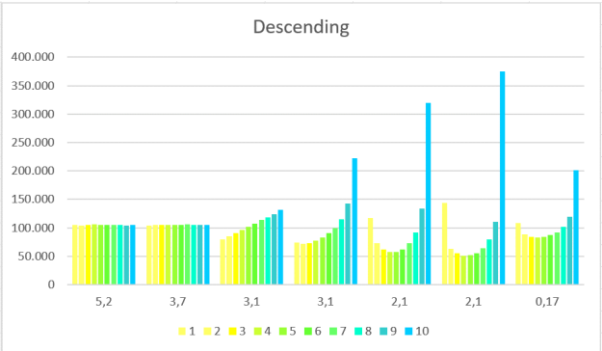
Worksheet Formulas		
Range	Formula	
J2:J3;J5:J14	**J2**	=SUM(B2:H2)
K2	**K2**	=IF(J2>$B$1,"No solution because sum of lower borders exceed " & $B$1 & ".","")
K3	**K3**	=IF(J3<$B$1,"No solution because sum of upper borders is less than " & $B$1 & ".","")
B5:H14	**B5**	=MAX(B$2,$B$1-SUM($A5:A5)-SUM(C$3:$I$3))+RAND()*(MIN(B$3,$B$1-SUM($A5:A5)-SUM(C$2:$I$2))-MAX(B$2,$B$1-SUM($A5:A5)-SUM(C$3:$I$3)))

Important note: There is not solution if the sum of lower borders exceeds 100 or if the sum of upper borders is less than 100! This will be checked in cells K2:K3.

The Distribution of the Random Numbers

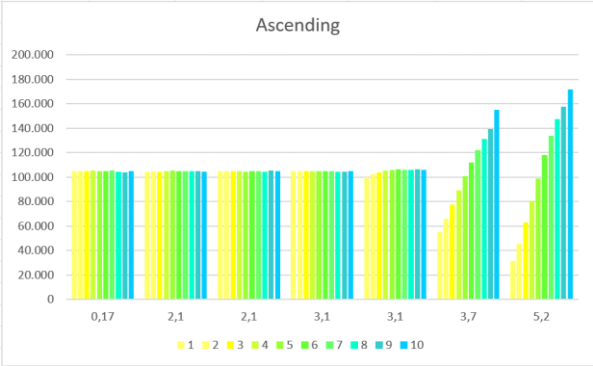
The generated random numbers in the example above are distributed fairly equally. With 1.048.572 generated rows of 7 numbers each you get for the original corridor width sort order (see on the right):





With descending corridor width sort order you get (see left):

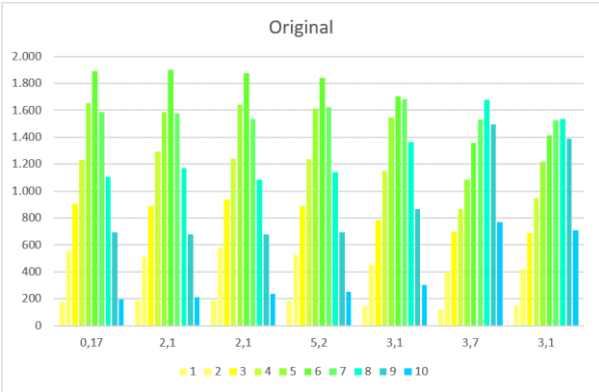
When the corridor widths are sorted ascending:



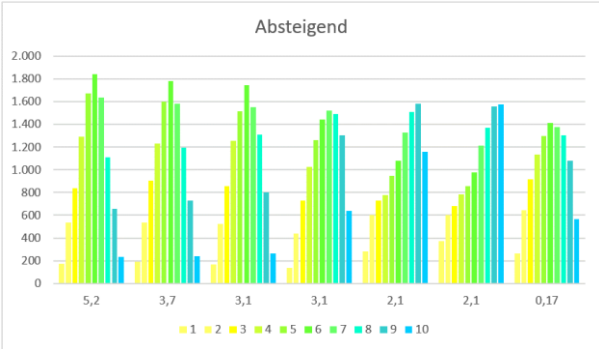
Conclusion: To achieve more equally distributed random numbers you should sort the columns ascending, because the generating formulas reduce the degree of freedom from left to right. If – for whatever reason – you have descending sorted corridor widths, you will have to expect far more extreme distributions.

With a Triang Distribution

With the triangular distribution *sbRandTriang* you get with 10,000 generated rows:



The corresponding formula in cell B5:
*=sbRandTriang(MAX(B$2,$B$1-SUM($A5:A5) - SUM(C$3:$I$3)),MIN(MAX(MAX(B$2,$B$1 - SUM($A5:A5) - SUM(C$3:$I$3)),B$2+($B$1 - (SUM($A5:A5) + SUM(B$2:$I$2))) / (SUM(B$3:$I$3) - SUM(B$2:$I$2)) * (B$3-B$2)),MIN(B$3,$B$1 - SUM($A5:A5) - SUM(C$2:$I$2))),MIN(B$3,$B$1 - SUM($A5:A5) - SUM(C$2:$I$2)))*.

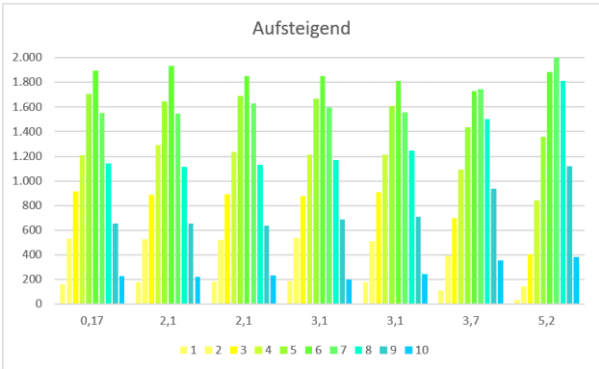


Rounded Results

IF the results needed to be rounded to a specified number of decimals, for example 2, then you can embed above formula in =ROUND(…, 2).

But keep in mind that you need to round at least to the maximal number of digits used in the corridor widths to ensure

- that the results are still within corridors after rounding,
- that parts of the corridors will not become unreachable,
- and that the target result will be achieved.



Correlated Random Numbers

Cholesky Decomposition

You can easily generate correlated random numbers with the Cholesky (pronounce: "koleski") decomposition:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1					Rho								Generation with Cholesky				Generation with RandCorr			
2						A	B	C	D											
3	Generate Random Numbers				A	1	0,5	0,2	0,01				1	0,45432	0,1687	0,0023	1	0,4372	0,15923	0,02459
4					B	0,5	1	0,01	0,3				0,45432	1	0,00085	0,32805	0,4372	1	0,02932	0,36533
5					C	0,2	0,01	1	0,3				0,1687	0,00085	1	0,31172	0,15923	0,02932	1	0,36185
6	A	B	C	D	D	0,01	0,3	0,3	1				0,0023	0,32805	0,31172	1	0,02459	0,36533	0,36185	1
7	-2,080207974	-1,131306224	-0,887397044	-0,576239068									-2,8291	-1,08381	-1,06176	-0,50458	-1,21847	-0,50723	0,57849	-0,50377
8	1,14906907	-0,242925297	-1,384032261	0,473747189					C^T				0,75554	0,09483	-1,1863	0,41484	0,27312	0,74424	-0,30034	0,06143
9	0,421931087	-0,461340439	0,59825812	0,900221447									0,31991	-0,15506	0,89092	0,78828	-1,97632	0,45212	0,99737	0,65117
10	-0,433013984	-0,375922188	0,727517026	0,09245532	Cholesky Decomposition				1	0,5	0,2	0,01	-0,47455	-0,36967	0,74044	0,08096	0,10233	2,35297	2,66745	2,683
11	1,200827849	0,843256856	-0,354650557	-0,860278461					0	0,866	-0,1	0,341	1,54292	0,4741	-0,63992	-0,7533	-0,45754	-0,66715	-0,25575	-0,4137
12	0,967097572	-0,151942651	-0,696613024	-0,246078912	C				0	0	0,974	0,342	0,74934	-0,14302	-0,7629	-0,21548	-1,28518	-0,6099	0,39879	0,52421
13	1,286836647	-0,22330551	-1,627856478	-1,625908615					0	0	0	0,876	0,83335	-0,57806	-2,14236	-1,42373	1,29686	-1,00004	-0,1363	-1,03376
14	1,996939096	-0,352785038	1,362529838	0,195398068	1	0	0	0	1	0,5	0,2	0,01	2,09501	-0,38056	1,39434	0,1711	-0,23561	-0,60737	-0,02907	0,14776
15	-0,470853633	0,962578832	-0,620654649	-0,065407252	0,5	0,866	0	0	0,5	1	0,01	0,3	-0,11435	0,87584	-0,62707	-0,05727	-0,44165	-0,12886	0,31837	0,80871
16	-2,595683321	1,275000985	1,593140445	-1,544229088	0,2	-0,1	0,974	0	0,2	0,01	1	0,3	-1,655	0,4126	1,0237	-1,3522	1,59747	0,68412	-0,24966	-1,1904
17	0,35170675	-1,504182954	-0,012158866	0,000475113	0,01	0,341	0,342	0,876	0,01	0,3	0,3	1	-0,40281	-1,30124	-0,01168	0,00042	0,44722	-0,64741	-0,79275	-1,64555
18	-0,531580808	-0,238529101	0,732290878	-1,57312347									-0,52012	-0,81854	0,17512	-1,3775	0,35778	-0,0631	-1,40398	-1,42843
19	0,661578647	1,61281996	0,63543325	0,314565819					Check: $C * C^T$ = Rho				1,59822	1,43786	0,72673	0,27545	1,91457	0,82557	-0,41141	0,73061
20	1,162189953	-0,476451735	0,20687178	-0,065385092									0,96468	-0,45639	0,17917	-0,05725	-1,59686	-1,29045	-0,0351	1,28889
21	-0,191969266	0,341903462	-2,247228611	-0,554787453		Quality Check							-0,47601	0,34065	-2,37926	-0,4858	-0,83393	-1,24884	0,20702	-0,96989
22	1,322840548	0,37044719	1,034874071	-0,203440313		Cholesky	0,008						1,713	0,14397	0,93863	-0,17814	1,09816	2,64416	-1,07835	-0,23648
23	1,002265962	-1,192274623	1,051553626	-0,260309811		RandCorr	0,029						0,61384	-1,23049	0,93542	-0,22794	-0,00984	1,4972	-0,65032	-0,00071
24	-0,87923958	-0,053003522	0,841913415	0,545277457									-0,73191	0,05234	1,00685	0,47747	2,78419	0,87705	1,24611	0,79866
25	-1,546911019	0,558851328	0,878997709	0,837635394									-1,08331	0,67796	1,14302	0,73348	-1,53004	-0,26901	-1,07242	-0,4354
26	0,437176144	-0,028827829	1,460406923	-0,37584362									0,71109	-0,30476	1,29421	-0,32911	-1,12103	-0,93551	0,57603	0,69055
27	0,08204136	0,162852216	1,622547641	0,839499041									0,49637	0,25838	1,86808	0,73511	0,77875	0,59299	-0,3612	-0,15179
28	1,473159292	-0,20197898	-0,016601489	-1,421900817									1,35463	-0,65755	-0,50276	-1,24509	-0,26566	0,57813	0,32219	0,15975
29	0,25187145	-1,044340612	-0,39576615	0,404785634									-0,3454	-0,72541	-0,24706	0,35445	0,51096	0,18893	0,81882	0,42309
30	0,514780035	0,248092185	-2,007280397	0,8474031									0,24584	0,71211	-1,66565	0,74203	0,43929	-0,80538	-1,02272	0,37711
31	2,515843349	0,717871931	0,258360588	-1,381000631									2,91264	0,12443	-0,22087	-1,20927	1,39916	-1,22663	0,36814	-0,34592
32	-0,424137204	-0,188254372	0,864570403	-1,031016356									-0,35566	-0,60408	0,4895	-0,90281	1,36842	0,22309	-0,86182	-0,57523
1002	0,815291161	1,508528101	-1,074389368	-1,54698764									1,33921	0,89112	-1,57613	-1,35462	0,03301	-0,69195	-0,41712	-0,02023
1003	-0,338549051	-0,880685946	0,488082368	0,645719931									-0,67482	-0,59346	0,69649	0,56542	-2,04245	-0,5453	0,16855	-0,57991
1004	0,318861792	0,755607341	-1,259719781	-0,776727166									0,43695	0,52071	-1,49311	-0,68014	0,15523	-2,31864	0,93056	-1,10899
1005	-0,649325597	1,447829339	0,675777212	-0,740401552									0,20234	0,93142	0,40502	-0,64833	0,55595	1,17743	-0,76364	0,67567
1006	0,316374012	1,184500809	1,901303267	0,131333329									1,2902	0,87296	1,89732	0,115	0,0355	-0,98565	0,10476	0,4518

Worksheet Formulas		
Range	Formula	
E14	**E14**	=Cholesky(F3:I6)
I10	**I10**	=TRANSPOSE(E14:H17)
I14	**I14**	=MMULT(E14:H17,I10:L13)
Cholesky_Check	**H22**	=SUM((F3:I6-M3:P6)^2)
RandCorr_Check	**H23**	=SUM((F3:I6-Q3:T6)^2)
M3:P6	**M3**	=CORREL(INDEX($M$7:$P$1005,,ROW()-2),INDEX($M$7:$P$1005,,COLUMN()-12))
M7	**M7**	=MMULT(A7:D1006,E14:H17)
Q3:T6	**Q3**	=CORREL(INDEX($Q$7:$T$1005,,ROW()-2),INDEX($Q$7:$T$1005,,COLUMN()-16))
Q7	**Q7**	=RandCorr(1000,Rho)

Cholesky and RandCorr Program Code

```
Function Cholesky(vA As Variant) As Variant
'I suggest to use the Cholesky decomposition just for purposes of demonstration.
'Better options are (in this order): tred2, tqli, eigsrt from Numerical Recipes.
'SVD also works but is computationally more expensive by far since it does not
'make use of symmetry.
'(Thanks to my former colleague Glen R.)
'Bernd Plumhoff 02-Nov-2024 PB V1.1
Dim d As Double
Dim i As Long, j As Long, k As Long, n As Long
With Application.WorksheetFunction
On Error Resume Next
vA = .Transpose(.Transpose(vA))
On Error GoTo 0
n = UBound(vA, 1)
If n <> UBound(vA, 2) Then
  Cholesky = CVErr(xlErrRef)
  Exit Function
End If
```

```
ReDim dR(1 To n, 1 To n) As Double 'Zeroing all elements
For j = 1 To n
  d = 0#
  For k = 1 To j - 1
    d = d + dR(j, k) * dR(j, k)
  Next k
  dR(j, j) = vA(j, j) - d
  If dR(j, j) > 0# Then
    dR(j, j) = Sqr(dR(j, j))
    For i = j + 1 To n
      d = 0#
      For k = 1 To j - 1
        d = d + dR(i, k) * dR(j, k)
      Next k
      dR(i, j) = (vA(i, j) - d) / dR(j, j)
    Next i
  Else
    'Cannot continue with usual Cholesky
    'Fill this column with zeros. Idea: Glen R.
    For i = j To n
      dR(i, j) = 0#
    Next i
  End If
Next j
Cholesky = dR
End With
End Function

Function RandCorr(n As Long, vVarCovar As Variant) As Variant
'Returns Ubound(vVarCovar,1) correlated random number vectors of length n.
'vVarCovar is a square matrix containing the variance/covariance matrix.
'Please notice that you will only get a "proxy" correlation, not an exact one.
'Bernd Plumhoff 06-Nov-2009 PB V0.2
Dim vA As Variant
Dim d As Double
Dim i As Long, j As Long, k As Long, m As Long

With Application.WorksheetFunction
vA = .Transpose(.Transpose(vVarCovar))
m = UBound(vA, 1)
If m <> UBound(vA, 2) Then
  RandCorr = CVErr(xlErrRef)
  Exit Function
End If

ReDim Db(1 To m, 1 To m) As Double
For j = 1 To m
  d = 0#
  For k = 1 To j - 1
    d = d + Db(j, k) * Db(j, k)
  Next k
  Db(j, j) = vA(j, j) - d
  If Db(j, j) <= 0 Then
    RandCorr = CVErr(xlErrNum)
    Exit Function
  End If
  Db(j, j) = Sqr(Db(j, j))

  For i = j + 1 To m
    d = 0#
    For k = 1 To j - 1
      d = d + Db(i, k) * Db(j, k)
    Next k
    Db(i, j) = (vA(i, j) - d) / Db(j, j)
  Next i
Next j

ReDim vR(1 To n, 1 To m) As Variant
For i = 1 To n
  For j = 1 To m
    vR(i, j) = .Norm_S_Inv(Rnd())
  Next j
Next i
vR = .MMult(vR, Db)
RandCorr = vR
End With
End Function
```

Iman-Conover Method

If you need to generate correlated random numbers, the Iman-Conover method is better than the Cholesky decomposition.

In 1982, Iman and Conover published their original paper "A distribution-free approach to inducing rank correlation among input variables":

https://www.uio.no/studier/emner/matnat/math/STK4400/v05/undervisningsmateriale/A%20distribution-free%20approach%20to%20rank%20correlation.pdf

In 2021, Rick Wicklin wrote "Simulate correlated variables by using the Iman-Conover transformation". His article includes an SAS implementation of the Iman-Conover method:

https://blogs.sas.com/content/iml/2021/06/14/simulate-iman-conover-transformation.html

In 2005, Stephen J. Mildenhall published "Correlation and Aggregate Loss Distributions with an Emphasis on the Iman-Conover Method":

https://www.mynl.com/old/wp/ic.pdf

I implemented the example from Mildenhall's paper both using Excel spreadsheet functions and with Excel / VBA:
The input matrix X (on the right):

	A	B	C	D
1	123.567	44.770	15.934	13.273
2	126.109	45.191	16.839	15.406
3	138.713	47.453	17.233	16.706
4	139.016	47.941	17.265	16.891
5	152.213	49.345	17.620	18.821
6	153.224	49.420	17.859	19.569
7	153.407	50.686	20.804	20.166
8	155.716	52.931	21.110	20.796
9	155.780	54.010	22.728	20.968
10	161.678	57.346	24.072	21.178
11	161.805	57.685	25.198	23.236
12	167.447	57.698	25.393	23.375
13	170.737	58.380	30.357	24.019
14	171.592	60.948	30.779	24.785
15	178.881	66.972	32.634	25.000
16	181.678	68.053	33.117	26.754
17	184.381	70.592	35.248	27.079
18	206.940	72.243	36.656	30.136
19	217.092	86.685	38.483	30.757
20	240.935	87.138	39.483	35.108

Input_Matrix_X | Target_Correlation_S | Cholesky_C | Inter

The target correlation S:

	A	B	C	D
1	1	0,8	0,4	0
2	0,8	1	0,3	-0,2
3	0,4	0,3	1	0,1
4	0	-0,2	0,1	1

... | Target_Correlation_S | Cholesky_C | Intermediate_M | Covariance

Worksheet Formulas		
Range	Formula	
A2;A3:B3;A4:C4	**A2**	=INDEX($A$1:$D$4,COLUMN(),ROW())

The Cholesky decomposition C of S:

	A	B	C	D
1	1,0000	0,8000	0,4000	0,0000
2	0,0000	0,6000	-0,0333	-0,3333
3	0,0000	0,0000	0,9159	0,0970
4	0,0000	0,0000	0,0000	0,9378

... | Cholesky_C | Intermediate_M | Covariance_E | Cholesky_F

Worksheet Formulas		
Range	Formula	
A1	**A1**	=TRANSPOSE(Cholesky(Target_Correlation_S!A1:D4))

The intermediate matrix M (constant values, identical to Mildenhall's data):

Worksheet Formulas		
Range	Formula	
I1	**I1**	=NORM.S.INV(SEQUENCE(20,1,1,1)/21)/STDEVPA(NORM.S.INV(SEQUENCE(20,1,1,1)/21))
J1:L1	**J1**	=TRANSPOSE(randomshuffle($A$1:$A$20))

	A	B	C	D
1	-1,92062	1,22896	-1,00860	-0,49584
2	-1,50709	-1,50709	-1,50709	0,82015
3	-1,22896	1,92062	0,82015	-0,65151
4	-1,00860	-0,20723	1,00860	-1,00860
5	-0,82015	0,82015	0,34878	1,92062
6	-0,65151	-1,22896	-0,65151	0,20723
7	-0,49584	-0,65151	1,22896	-0,34878
8	-0,34878	-0,49584	-0,49584	-0,06874
9	-0,20723	-1,00860	0,20723	0,65151
10	-0,06874	0,49584	0,06874	-1,22896
11	0,06874	-0,34878	-1,22896	0,49584
12	0,20723	0,34878	0,65151	0,34878
13	0,34878	-0,06874	-0,20723	1,22896
14	0,49584	-1,92062	-0,82015	-0,20723
15	0,65151	0,20723	1,92062	-1,92062
16	0,82015	1,00860	1,50709	1,50709
17	1,00860	-0,82015	-1,92062	1,00860
18	1,22896	1,50709	0,49584	-1,50709
19	1,50709	0,06874	-0,06874	0,06874
20	1,92062	0,65151	-0,34878	-0,82015

... Intermediate_M | Covariance_E | Cholesky_F | Intermediat

You can create similar data with this formula in A1:A20:
=NORM.S.INV(SEQUENCE(20,1,1,1)/21) / STDEVPA(NORM.S.INV(SEQUENCE(20,1,1,1)/21))
and with the formula
=MTRANS(RandomShuffle($A$1:$A$20))
in B1:B20 (copy to columns C and D).

Now you get covariance matrix E:

	A	B	C	D
1	1,0000	0,0486	0,0898	-0,0960
2	0,0486	1,0000	0,4504	-0,2408
3	0,0898	0,4504	1,0000	-0,3192
4	-0,0960	-0,2408	-0,3192	1,0000

... Covariance_E | Cholesky_F | Intermediate_T | Check_Cc ...

Worksheet Formulas		
Range	Formula	
A1:D4	**A1**	=COVAR(INDEX(Intermediate_M!$A$1:$D$20,,ROW()),INDEX(Intermediate_M!$A$1:$D$20,,COLUMN()))

And its Cholesky decomposition F:

	A	B	C	D
1	1,0000	0,0486	0,0898	-0,0960
2	0,0000	0,9988	0,4466	-0,2364
3	0,0000	0,0000	0,8902	-0,2303
4	0,0000	0,0000	0,0000	0,9391

... Cholesky_F | Intermediate_T | Check_Correlation | Rank_T

Worksheet Formulas		
Range	Formula	
A1	**A1**	=TRANSPOSE(Cholesky(Covariance_E!A1:D4))

The intermediate matrix T:

Worksheet Formulas		
Range	Formula	
A1	**A1**	=MMULT(MMULT(Intermediate_M!A1:D20,MINVERSE(Cholesky_F!A1:D4)),Cholesky_C!A1:D4)

	A	B	C	D
1	-1,92062	-0,74214	-2,28105	-1,33232
2	-1,50709	-2,06697	-1,30678	0,54577
3	-1,22896	0,20647	-0,51141	-0,94465
4	-1,0086	-0,9019	0,80546	-0,65873
5	-0,82015	-0,13949	-0,31782	1,7696
6	-0,65151	-1,24042	-0,28	0,23988
7	-0,49584	-0,77356	1,42145	0,23612
8	-0,34878	-0,56669	-0,38117	-0,14744
9	-0,20723	-0,76561	0,64214	0,97494
10	-0,06874	0,24487	-0,19673	-1,33695
11	0,06874	-0,15653	-1,06954	0,14014
12	0,20723	0,36925	0,56695	0,51206
13	0,34878	0,22754	-0,06362	1,1955
14	0,49584	-0,77155	0,26828	0,03168
15	0,65151	0,62666	2,08987	-1,21744
16	0,82015	1,23804	1,32493	1,8568
17	1,0086	0,28474	-1,23688	0,59246
18	1,22896	1,85259	0,17411	-1,62428
19	1,50709	1,20294	0,39517	0,13931
20	1,92062	1,87176	-0,04335	-0,97245

Cholesky_F | Intermediate_T | Check_Correlation | Rank_T

You can check the generated correlations:

	A	B	C	D
1	1,0000	0,8000	0,4000	0,0000
2	0,8000	1,0000	0,3000	-0,2000
3	0,4000	0,3000	1,0000	0,1000
4	0,0000	-0,2000	0,1000	1,0000
5				
6	**Difference to Target Correlation:**			
7				
8	0	0	0	-4,44089E-17
9	0	0	0	0
10	0	0	0	0
11	-4,44089E-17	0	0	0

... | Cholesky_F | Intermediate_T | Check_Correlation | Rank_T | Result_Y | Check_Correlatic

Worksheet Formulas		
Range	Formula	
A1:D4	**A1**	=CORREL(INDEX(Intermediate_T!$A$1:$D$20,,ROW()),INDEX(Intermediate_T!$A$1:$D$20,,COLUMN()))
A8	**A8**	=A1:D4-Target_Correlation_S!A1:D4

Calculate the ranks of numbers in columns of T in sheet Rank_T:

Worksheet Formulas		
Range	Formula	
A1:D1	**A1**	=RANK(Intermediate_T!A1:A20,Intermediate_T!A1:A20,1)

	A	B	C	D
1	1	7	1	3
2	2	1	2	15
3	3	11	5	6
4	4	3	17	7
5	5	10	7	19
6	6	2	8	13
7	7	4	19	12
8	8	8	6	8
9	9	6	16	17
10	10	13	9	2
11	11	9	4	11
12	12	15	15	14
13	13	12	10	18
14	14	5	13	9
15	15	16	20	4
16	16	18	18	20
17	17	14	3	16
18	18	19	12	1
19	19	17	14	10
20	20	20	11	5

Finally you will get the results in sheet Result_Y:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	123.567	50.686	15.934	16.706		VBA	123.567	44.770	17.265	24.785		Worksheet formulae	123.567	50.686	15.934	16.706
2	126.109	44.770	16.839	25.000			126.109	45.191	33.117	24.019			126.109	44.770	16.839	25.000
3	138.713	57.685	17.620	19.569			138.713	50.686	17.233	13.273			138.713	57.685	17.620	19.569
4	139.016	47.453	35.248	20.166			139.016	57.685	20.804	30.136			139.016	47.453	35.248	20.166
5	152.213	57.346	20.804	30.757			152.213	57.346	30.357	21.178			152.213	57.346	20.804	30.757
6	153.224	45.191	21.110	24.019			153.224	49.420	16.839	20.968			153.224	45.191	21.110	24.019
7	153.407	47.941	38.483	23.375			153.407	47.941	15.934	26.754			153.407	47.941	38.483	23.375
8	155.716	52.931	17.859	20.796			155.716	47.453	22.728	19.569			155.716	52.931	17.859	20.796
9	155.780	49.420	33.117	27.079			155.780	52.931	25.393	35.108			155.780	49.420	33.117	27.079
10	161.678	58.380	22.728	15.406			161.678	49.345	25.198	23.236			161.678	58.380	22.728	15.406
11	161.805	54.010	17.265	23.236			161.805	86.685	36.656	15.406			161.805	54.010	17.265	23.236
12	167.447	66.972	32.634	24.785			167.447	66.972	30.779	16.706			167.447	66.972	32.634	24.785
13	170.737	57.698	24.072	30.136			170.737	54.010	35.248	20.796			170.737	57.698	24.072	30.136
14	171.592	49.345	30.357	20.968			171.592	68.053	17.859	18.821			171.592	49.345	30.357	20.968
15	178.881	68.053	39.483	16.891			178.881	57.698	38.483	27.079			178.881	68.053	39.483	16.891
16	181.678	72.243	36.656	35.108			181.678	70.592	17.620	16.891			181.678	72.243	36.656	35.108
17	184.381	60.948	17.233	26.754			184.381	58.380	24.072	20.166			184.381	60.948	17.233	26.754
18	206.940	86.685	25.393	13.273			206.940	72.243	32.634	30.757			206.940	86.685	25.393	13.273
19	217.092	70.592	30.779	21.178			217.092	60.948	39.483	23.375			217.092	70.592	30.779	21.178
20	240.935	87.138	25.198	18.821			240.935	87.138	21.110	25.000			240.935	87.138	25.198	18.821

Worksheet Formulas		
Range	Formula	
A1:D1;M1:P1	**A1**	=INDEX(Input_Matrix_X!$A$1:$A$20,Rank_T!$A$1:$A$20)
G1	**G1**	=ImanConover(Input_Matrix_X!$A$1:$D$20,Target_Correlation_S!$A$1:$D$4)

You can check the differences to your target correlation in sheet Check_Correlation_Result:

	A	B	C	D	E	F
1	1,00	0,85	0,26	-0,11		
2	0,85	1,00	0,19	-0,20		
3	0,26	0,19	1,00	0,10		
4	-0,11	-0,20	0,10	1,00		
5						
6	**Difference to Target Correlation:**					**Maximal absolute error:**
7						
8	0	0,049673836	-0,13590681	-0,11360723		0,135906814
9	0,049673836	0	-0,11151318	0,000215604		
10	-0,13590681	-0,11151318	0	-0,00429182		
11	-0,11360723	0,000215604	-0,00429182	0		

Worksheet Formulas		
Range	Formula	
A1:D4	**A1**	=CORREL(INDEX(Result_Y!$A$1:$D$20,,ROW()),INDEX(Result_Y!$A$1:$D$20,,COLUMN()))
A8	**A8**	=A1:D4-Target_Correlation_S!A1:D4
F8	**F8**	=MAX(ABS(A8:D11))

RandomShuffle Program Code

```
Function RandomShuffle(vtemp As Variant) As Variant
Dim j As Long, k As Long, n As Long
Dim temp As Double, u As Double
'Application.Volatile 'Uncomment if you think you need this.
With Application.WorksheetFunction
On Error Resume Next 'Ignore error: VBA calls already with 1-dim array.
vtemp = .Transpose(vtemp)
On Error GoTo 0
n = UBound(vtemp)
j = n
Do While j > 0
  u = Rnd()
  k = Int(j * u + 1)
  temp = vtemp(j)
  vtemp(j) = vtemp(k)
  vtemp(k) = temp
  j = j - 1
Loop
RandomShuffle = vtemp
End With
End Function
```

IndexX Program Code

```
Function IndexX(n As Long, arr As Variant, colNo As Long) As Variant
'Indexes an array arr[1..n], i.e., outputs the array indx[1..n] such
'that arr[indx[j]] is in ascending order for j = 1, 2, . . . ,n. The
'input quantities n and arr are not changed. Translated from [31].
Const m As Long = 7
Const NSTACK As Long = 50
Dim i As Long, indxt As Long, ir As Long, itemp As Long, j As Long, k As Long, l As Long
Dim jstack As Long, istack(1 To NSTACK) As Long, a As Double

ir = n: l = 1
ReDim indx(1 To n) As Long
For j = 1 To n: indx(j) = j: Next j

Do While 1
  If (ir - l < m) Then
    For j = l + 1 To ir
      indxt = indx(j)
      a = arr(indxt, colNo)
      For i = j - 1 To l Step -1
        If (arr(indx(i), colNo) <= a) Then Exit For
        indx(i + 1) = indx(i)
      Next i
      indx(i + 1) = indxt
    Next j
    If (jstack = 0) Then Exit Do
    ir = istack(jstack)
    jstack = jstack - 1
    l = istack(jstack)
    jstack = jstack - 1
  Else
    k = (l + ir) / 2
    itemp = indx(k)
    indx(k) = indx(l + 1)
    indx(l + 1) = itemp
    If (arr(indx(l), colNo) > arr(indx(ir), colNo)) Then
      itemp = indx(l)
      indx(l) = indx(ir)
      indx(ir) = itemp
    End If
    If (arr(indx(l + 1), colNo) > arr(indx(ir), colNo)) Then
      itemp = indx(l + 1)
      indx(l + 1) = indx(ir)
      indx(ir) = itemp
    End If
    If (arr(indx(l), colNo) > arr(indx(l + 1), colNo)) Then
      itemp = indx(l)
      indx(l) = indx(l + 1)
      indx(l + 1) = itemp
    End If
    i = l + 1
    j = ir
    indxt = indx(l + 1)
    a = arr(indxt, colNo)
    Do While 1
      Do
        i = i + 1
      Loop While (arr(indx(i), colNo) < a)
      Do
        j = j - 1
      Loop While (arr(indx(j), colNo) > a)
      If (j < i) Then Exit Do
      itemp = indx(i)
      indx(i) = indx(j)
      indx(j) = itemp
    Loop
    indx(l + 1) = indx(j)
    indx(j) = indxt
    jstack = jstack + 2
    If (jstack > NSTACK) Then
      'STACK too small in indexx
      IndexX = CVErr(xlErrNum)
      Exit Function
    End If
    If (ir - i + 1 >= j - l) Then
      istack(jstack) = ir
      istack(jstack - 1) = i
      ir = j - 1
    Else
      istack(jstack) = j - 1
      istack(jstack - 1) = l
      l = i
    End If
  End If
Loop
IndexX = indx
End Function
```

ImanConover Program Code

This is the VBA implementation of the Iman-Conover method. I use this code in *sbGenerateTestData*, too.

Please notice that the function ImanConover uses (calls) the user-defined functions IndexX and RandomShuffle mentioned above as well as the function *Cholesky* (*Cholesky*).

```
Function ImanConover(rInputMatrix As Range, _
        rTargetCorrelation As Range) As Variant
'Implements the Iman-Conover method to generate random
'number vectors with a given correlation.
'Algorithm as described in:
'Mildenham, November 27, 2005
'Correlation and Aggregate Loss Distributions With An
'Emphasis On The Iman-Conover Method
'V0.3 PB 02-Nov-2024 by Bernd Plumhoff
Dim vX As Variant   'Input matrix
Dim vS As Variant   'Target correlation matrix
Dim vC As Variant   'Cholesky decomposition of vS
Dim vM As Variant   'Intermediate matrix M
Dim vE As Variant   'Covariance matrix E
Dim vF As Variant   'Cholesky decomposition of vE
Dim vT As Variant   'Intermediate matrix T
Dim d As Double, dS As Double
Dim i As Long, j As Long, k As Long
Dim lRow As Long, lCol As Long
Dim state As SystemState

With Application
Set state = New SystemState
vX = .Transpose(.Transpose(rInputMatrix))
lRow = rInputMatrix.Rows.Count
lCol = rInputMatrix.Columns.Count

'##############################################################################
'#                                Check inputs                                #
'##############################################################################

If lCol <> rTargetCorrelation.Columns.Count _
  And rTargetCorrelation.Rows.Count <> rTargetCorrelation.Columns.Count Then
  'Structure of target correlation matrix needs to fit input matrix
  ImanConover = CVErr(xlErrNum)
  Exit Function
End If
vS = .Transpose(.Transpose(rTargetCorrelation))
For i = 1 To lCol
  If vS(i, i) <> 1# Then
    'Target correlation matrix not 1 on diagonal
    ImanConover = CVErr(xlErrValue)
    Exit Function
  End If
  For j = 1 To i - 1
    If vS(i, j) <> vS(j, i) Then
      'Target correlation matrix not symmetric
      ImanConover = CVErr(xlErrValue)
      Exit Function
    End If
  Next j
Next i

vC = .Transpose(Cholesky(vS))

'##############################################################################
'#                          Create intermediate matrix M                      #
'##############################################################################

ReDim vMV(1 To lRow) As Double
d = 0#
dS = 0#
For i = 1 To Int(lRow / 2)
  vMV(i) = .NormSInv(i / (lRow + 1))
  vMV(lRow - i + 1) = -vMV(i)
  d = d + 2# * vMV(i) * vMV(i)
Next i
If lRow Mod 2 = 1 Then vMV((lRow + 1) / 2) = 0 'Just for clarity, it's already 0
d = Sqr(d / lRow)
For i = 1 To lRow
  vMV(i) = vMV(i) / d
Next i

vM = vX
For i = 1 To lRow
```

```
  vM(i, 1) = vMV(i)
Next i

Dim vMW As Variant
For i = 2 To lCol
  vMW = RandomShuffle(vMV)
  For j = 1 To lRow
    vM(j, i) = vMW(j)
  Next j
Next i

'##############################################################################
'#                        Calculate covariance matrix E                       #
'##############################################################################

vE = vC
For i = 1 To lCol
  vE(i, i) = .Covar(.Index(.Transpose(vM), i), .Index(.Transpose(vM), i))
  For j = i + 1 To lCol
    vE(i, j) = .Covar(.Index(.Transpose(vM), i), .Index(.Transpose(vM), j))
    vE(j, i) = vE(i, j)
  Next j
Next i

vF = .Transpose(Cholesky(vE))

vT = .MMult(.MMult(vM, .MInverse(vF)), vC)

'##############################################################################
'#                        Compute ranks of matrix T                           #
'##############################################################################

Dim vRT As Variant, vR As Variant
vRT = vX
For j = 1 To lCol
  vR = IndexX(lRow, vT, j)
  For i = 1 To lRow
    vRT(i, j) = vR(i)
  Next i
  vR = IndexX(lRow, vX, j)
  For i = 1 To lRow
    vX(i, j) = vX(vR(i), j)
  Next i
Next j

'##############################################################################
'#                        Calculate result matrix Y                           #
'##############################################################################

Dim vY As Variant
vY = vX
For i = 1 To lRow
  For j = 1 To lCol
    vY(i, j) = vX(vRT(i, j), j)
  Next j
Next i

ImanConover = vY
End With
End Function
```

Practical Applications of General Random Numbers

Generating Test Data – *sbGenerateTestData*

When you want to thoroughly test an application or program, you often need test data. The *sbGenerateTestData* application is designed to help you generate random test data in numerical form or as text.

For example, if you want to generate six boolean values, with 50% TRUE and 50% FALSE, once in the generated sequence and once randomly shuffled::

H6 | fx | 1

	A	B	C	D	F	G	H	I	J
1	Test Input 1	Test Input 2	Test Result	Correct Result		Test Data Generator	Input 1	Input 2	Explanation
2	TRUE	TRUE	Test formulae go here	Cross check formulae		Generate Test Data			
3	TRUE	TRUE				Number of test records	6	6	
4	TRUE	FALSE				Shuffle after generation	FALSE	TRUE	Perform a random shuffle afte
5	FALSE	TRUE				Data type			
6	FALSE	FALSE				Boolean	1	1	*Weight across data types*
7	FALSE	FALSE				True	1	1	Weight within Boolean data type
8						False	1	1	Weight within Boolean data type

Or you need 4 amounts of money in British pounds (GBP), the first series between 10 GBP and 20 GBP, and the second with an average value of 6 GBP and a standard deviation of 2 GBP:

	A	B	C	D	F	G	H	I	
1	Test Input 1	Test Input 2	Test Result	Correct Result		Test Data Generator	Input 1	Input 2	Explanation
2	£14.97	£7.56	Test formulae go here	Cross check formulae		Generate Test Data			
3	£11.55	£7.75				Number of test records	4	4	
4	£12.24	£2.76				Shuffle after generation	FALSE	TRUE	Perform a random shuffle
5	£13.26	£5.94				Data type			
9						Currency	1	1	*Weight across data types*
10						Min	10		Choose either Min and Max ...
11						Max	20		
12						Avg		6	... or Avg and StDev
13						StDev		2	

If you need four dates between January 1, 2000, and January 1, 2013, or four dates with an average value of June 30, 2012, and a standard deviation of 180 days:

	A	B	C	D	F	G	H	I	
1	Test Input 1	Test Input 2	Test Result	Correct Result		Test Data Generator	Input 1	Input 2	Explanation
2	19/11/2001 14:33	14/11/2011 06:05	Test formulae go here	Cross check formulae		Generate Test Data			
3	09/05/2003 05:52	27/02/2012 13:35				Number of test records	4	4	
4	14/05/2000 03:28	26/12/2012 13:19				Shuffle after generation	FALSE	TRUE	Perform a random shuffle
5	06/10/2010 16:36	19/12/2012 14:59				Data type			
14						Date	1	1	*Weight across data types*
15						Min	01/01/2000		Choose either Min and Max ...
16						Max	01/01/2013		
17						Avg		30/06/2012	... or Avg and StDev
18						StDev		180	

If you want to generate four country names, one from Africa, one from Asia, and two from Europe; or if you need two Asian and two European country names (move the "Countries" sheet to the right of the "Data" sheet so that it becomes Sheet 2).:

	A	B	C	D	E	F	G	H	I	J
1	Test Input 1	Test Input 2	Test Result	Correct Result			Test Data Generator	Input 1	Input 2	Explanation
2	Macau	Philippines	Test formulae go here	Cross check formulae			Generate Test Data			
3	Iceland	Gibraltar					Number of test records	4	4	
4	Slovakia	Vatican City					Shuffle after generation	FALSCH	WAHR	Perform a random shuffle after data generation
5	South Africa	Pakistan					Data type			
36							Length			Either a simple string ...
37							Min	A	a	"A" or "a"
38							Max	Z	z	"Z" or "z"
39							NextTabRepeat			... or an item from next tab ... First define how often eac
40							NextTabColumn	2	2	Column of items in next tab
41							NextTabItemRepeat	1	1	... or an item from weighted item groups
42							NextTabItemColumn	1	1	Column of items in next tab
43							NextTabGroupColumn	2	2	Column of item groups in next tab
44							Asia	1	1	List item groups on the left and then their weights
45							Europe	2	1	
46							Africa	1		
47							Oceania			
48							North America			
49							Antarctica			
50							South America			

If you want to randomly select first names from a given list, move the "First_Names" sheet to the right of the "Data" sheet. After pressing the "Generate Test Data" button again, you will receive a warning. Simply click "Ok":

	A	B	C	D	E	F	G	H	I	
1	Test Input 1	Test Input 2	Test Result	Correct Result			Test Data Generator	Input 1	Input 2	Explanation
2	BURTON	CHESMU	Test formulae go here	Cross check formulae			Generate Test Data			
3	CELESTE	AOLANI					Number of test records	4	4	
4	DONELLE	CHYNNA					Shuffle after generation	FALSCH	WAHR	Perform a random shuffle aft
5	CASSIDY	BYRD					Data type			
36							Length			Either a simple string ...
37							Min	A	a	"A" or "a"
38							Max	Z	z	"Z" or "z"
39							NextTabRepeat			... or an item from next tab ... First
40							NextTabColumn	2	2	Column of items in next tab
41							NextTabItemRepeat	1	1	... or an item from weighted item
42							NextTabItemColumn	1	1	Column of items in next tab
43							NextTabGroupColumn	2	2	Column of item groups in next tab
44							M	1	1	List item groups on the left and the
45							F	2	1	
46							E	1		

Data | First_Names | Countries | MyNumbers | Gen_Corr_Target_Corr | Gen_Corr_Input_Series | Gen_Corr_Output_Series

Note: The columns for the list items and their groups are not randomly identical here. This has been intentionally designed so that you can easily change the desired values by moving the corresponding sheet next to the "Data" sheet, either to first names or to country names.

With this application, you can also generate correlated pseudorandom numbers. I implemented the Iman-Conover method using VBA.

The sheets:

I4

	A	B	C	D	E	F	G	H	I	J
1	Test Input 1	Test Input 2	Test Result	Correct Result			Test Data Generator	Input 1	Input 2	Explanation
2	Slovenia	Andorra	Test formulae go here	Cross check formulae			Generate Test Data			
3	Russian Federation	Macedonia					Number of test records	10	5	
4	Guernsey	Isle of Man					Shuffle after generation			Perform a random shuffle after data g
5	Sweden	France					Data type			
6	San Marino	United Kingdom					Boolean			*Weight across data types*
7	Andorra						True			Weight within Boolean data type
8	Slovakia						False			Weight within Boolean data type
9	Gibraltar						Currency			*Weight across data types*
10	Isle of Man						Min	10		Choose either Min and Max ...
11	Monaco						Max	20		

Data | Countries | MyNumbers | First_Names | Gen_Corr_Target_Corr | Gen_Corr_Input_Series | Gen_Corr_Output_Series

Sheet ‚Countries‘:
Source: https://raw.github.com/wikimedia/limn-data/master/geo/country-codes.csv

	A	B
1	**Country Name**	**Continent Name**
2	Afghanistan	Asia
3	Åland	Europe
4	Albania	Europe
5	Algeria	Africa
6	American Samoa	Oceania
7	Andorra	Europe
8	Angola	Africa
9	Anguilla	North America
10	Antarctica	Antarctica
11	Antigua and Barbuda	North America
12	Argentina	South America
13	Armenia	Asia
14	Aruba	North America
15	Australia	Oceania
16	Austria	Europe
17	Azerbaijan	Asia

Data | **Countries** | First_Names | Gen_Corr_T

Sheet ‚First_Names‘:
Sources: https://www2.gov.bc.ca/assets/gov/birth-adoption-death-marriage-and-divorce/statistics-reports/baby-names-trends-m-2023.csv
https://www2.gov.bc.ca/assets/gov/birth-adoption-death-marriage-and-divorce/statistics-reports/baby-names-trends-m-2023.csv

	A	B
1	**Name**	**Gender**
2	AADHYA	F
3	AADYA	F
4	AAHANA	F
5	AALIYAH	F
6	AANYA	F
7	AARNA	F
8	AARYA	F
9	AARZA	F
10	AASHVI	F
11	ABBEY	F
12	ABBIE	F
13	ABBIGAIL	F
14	ABBY	F
15	ABBYGAIL	F
16	ABIGAIL	F
17	ABIGALE	F

First_Names

Correlated random numbers you generate with this application as follows:

	A	B	C	D
1	1	0,8	0,4	0
2	0,8	1	0,3	-0,2
3	0,4	0,3	1	0,1
4	0	-0,2	0,1	1
5				
6	**Result correlation**			
7	1	0,789976	0,385342	0,024391
8	0,789976	1	0,307256	-0,19452
9	0,385342	0,307256	1	0,073247
10	0,024391	-0,19452	0,073247	1
11				
12	**Correlation difference**			
13	0	-0,01002	-0,01466	0,024391
14	-0,01002	0	0,007256	0,00548
15	-0,01466	0,007256	0	-0,02675
16	0,024391	0,00548	-0,02675	0
17				
18	**Largest absolute error**			
19	0,026753			

Generate correlated data

How to generate correlated columns of numbers:
1. Enter your input series into sheet Gen_Corr_Input_Series
2. Enter the desired target correlation matrix into this sheet, top left corner
3. Press the button "Generate correlated data"
Adjust constant CMaxIter in module TestCorrelatedNumbers if necessary
Re-press button if your largest absolute error is too high
Finally use correlated numbers in sheet Gen_Corr_Output_Series

Worksheet Formulas		
Range	Formula	
A7:D10	**A7**	=CORREL(INDEX(Gen_Corr_Output_Series!$A$1:$D$20,,ROW()-6),INDEX(Gen_Corr_Output_Series!$A$1:$D$20,,COLUMN()))
A13	**A13**	=A7 - A1
A19	**A19**	=MAX(ABS(A13:D16))

	A	B	C	D
1	123.567	44.770	15.934	13.273
2	126.109	45.191	16.839	15.406
3	138.713	47.453	17.233	16.706
4	139.016	47.941	17.265	16.891
5	152.213	49.345	17.620	18.821
6	153.224	49.420	17.859	19.569
7	153.407	50.686	20.804	20.166
8	155.716	52.931	21.110	20.796
9	155.780	54.010	22.728	20.968
10	161.678	57.346	24.072	21.178
11	161.805	57.685	25.198	23.236
12	167.447	57.698	25.393	23.375
13	170.737	58.380	30.357	24.019
14	171.592	60.948	30.779	24.785
15	178.881	66.972	32.634	25.000
16	181.678	68.053	33.117	26.754
17	184.381	70.592	35.248	27.079
18	206.940	72.243	36.656	30.136
19	217.092	86.685	38.483	30.757
20	240.935	87.138	39.483	35.108

Gen_Corr_Input_Series | Gen_Corr_Output_Series

	A	B	C	D
1	123.567	44.770	17.859	20.796
2	126.109	45.191	15.934	23.375
3	138.713	50.686	17.620	20.968
4	139.016	47.453	17.233	15.406
5	152.213	49.345	35.248	30.757
6	153.224	66.972	25.198	24.019
7	153.407	49.420	17.265	19.569
8	155.716	52.931	38.483	23.236
9	155.780	57.685	25.393	21.178
10	161.678	60.948	21.110	35.108
11	161.805	57.698	36.656	16.706
12	167.447	57.346	16.839	18.821
13	170.737	47.941	39.483	30.136
14	171.592	58.380	20.804	26.754
15	178.881	87.138	32.634	16.891
16	181.678	54.010	24.072	27.079
17	184.381	68.053	33.117	13.273
18	206.940	72.243	22.728	25.000
19	217.092	70.592	30.357	24.785
20	240.935	86.685	30.779	20.166

Gen_Corr_Input_Series | Gen_Corr_Output_Series

sbGenerateTestData Program Code

```
Enum types
  ty_start = 0 'So that we can iterate from ty_start + 1 to ty_end - 1
  ty_boolean
  ty_currency
  ty_date
  ty_decimal
  ty_double
  ty_long
  ty_string
  ty_end  'So that we can iterate from ty_start + 1 to ty_end - 1
End Enum 'types

Enum param_rows
  pr_records = 3
  pr_shuffle
  pr_Boolean = 6
    pr_bTrue
    pr_bFalse
  pr_Currency
    pr_ccyMin
    pr_ccyMax
    pr_ccyAvg
    pr_ccyStDev
  pr_Date
    pr_dtMin
    pr_dtMax
    pr_dtAvg
    pr_dtStDev
  pr_Decimal
    pr_decMin
    pr_decMax
    pr_decAvg
    pr_decStDev
  pr_Double
    pr_dMin
    pr_dMax
    pr_dAvg
    pr_dStDev
  pr_Long
    pr_lSum
    pr_lMin1
    pr_lMin2
    pr_lMax
    pr_lMaxRepeat
  pr_String
    pr_sLength
    pr_sMin
    pr_sMax
    pr_sNextTabRepeat
    pr_sNextTabColumn
    pr_sNextTabItemRepeat
    pr_sNextTabItemColumn
    pr_sNextTabGroupColumn
    pr_sNextTabGroupWeights 'Item group weights start from here and can go down any number
End Enum 'param_rows

Enum param_columns
  pc_Output1 = 1
  pc_Output2
  pc_ItemGroups = 7
  pc_Input1 = 8
  pc_Input2
End Enum 'param_columns

Private Enum xlCI 'Excel Color Index
: xlCIBlack = 1: xlCIWhite: xlCIRed: xlCIBrightGreen: xlCIBlue '1 - 5
: xlCIYellow: xlCIPink: xlCITurquoise: xlCIDarkRed: xlCIGreen '6 - 10
: xlCIDarkBlue: xlCIDarkYellow: xlCIViolet: xlCITeal: xlCIGray25 '11 - 15
: xlCIGray50: xlCIPeriwinkle: xlCIPlum: xlCIIvory: xlCILightTurquoise '16 - 20
: xlCIDarkPurple: xlCICoral: xlCIOceanBlue: xlCIIceBlue: xlCILightBrown '21 - 25
: xlCIMagenta2: xlCIYellow2: xlCICyan2: xlCIDarkPink: xlCIDarkBrown '26 - 30
: xlCIDarkTurquoise: xlCISeaBlue: xlCISkyBlue: xlCILightTurquoise2: xlCILightGreen '31 - 35
: xlCILightYellow: xlCIPaleBlue: xlCIRose: xlCILavender: xlCITan '36 - 40
: xlCILightBlue: xlCIAqua: xlCILime: xlCIGold: xlCILightOrange '41 - 45
: xlCIOrange: xlCIBlueGray: xlCIGray40: xlCIDarkTeal: xlCISeaGreen '46 - 50
: xlCIDarkGreen: xlCIGreenBrown: xlCIBrown: xlCIDarkPink2: xlCIIndigo '51 - 55
: xlCIGray80 '56
End Enum

Sub sbGenerateTestData()
'Randomly generate test data as specified in input area.
'Bernd Plumhoff 06-Apr-2021 PB V0.2

Dim bGroupsUpToDate As Boolean
Dim dAvg As Double
Dim dmax As Double
```

```
Dim dmin As Double
Dim dStDev As Double
Dim dSumWeights As Double
ReDim dTypeWeight(ty_start + 1 To ty_end - 1) As Double
ReDim sTypeName(ty_start + 1 To ty_end - 1) As String
Dim i As Long
Dim j As Long
Dim k As Long
Dim lCol As Long
Dim lLength As Long
Dim lRecord As Long
Dim lRow As Long
Dim lIdx As Long
Dim lTypeSum As Long
Dim objItem As Object
Dim objGroup As Object
Dim s As String
Dim sErrMsg As String
Dim v As Variant
Dim vThisType As Variant
Dim vType As Variant
Dim vGroup As Variant
Dim wsItem As Worksheet
Dim state As SystemState

Set state = New SystemState
Randomize

With Application.WorksheetFunction

'Clear input
wsD.Range("A:A").Offset(, pc_Output1 - 1).ClearContents
wsD.Range("A:A").Offset(, pc_Output2 - 1).ClearContents
wsD.Range("A:A").Offset(, pc_Output1 - 1).ClearFormats
wsD.Range("A:A").Offset(, pc_Output2 - 1).ClearFormats
wsD.Range("A:A").Offset(, pc_Output1 - 1).Interior.ColorIndex = xlCIGray25
wsD.Range("A:A").Offset(, pc_Output2 - 1).Interior.ColorIndex = xlCIGray25
With wsD.Range("A1").Offset(, pc_Output1 - 1)
  .Formula = "Test Input 1"
  .Font.Bold = True
  .Interior.ColorIndex = xlCIBrightGreen
End With
With wsD.Range("A1").Offset(, pc_Output2 - 1)
  .Formula = "Test Input 2"
  .Font.Bold = True
  .Interior.ColorIndex = xlCIBrightGreen
End With

sTypeName(ty_boolean) = "Boolean"
sTypeName(ty_currency) = "Currency"
sTypeName(ty_date) = "Date"
sTypeName(ty_decimal) = "Decimal"
sTypeName(ty_double) = "Double"
sTypeName(ty_long) = "Long"
sTypeName(ty_string) = "String"

For lCol = pc_Input1 To pc_Input2
  sErrMsg = ""
  lRecord = wsD.Cells(pr_records, lCol)
  If lRecord <= 0 Then
    Call MsgBox("Number of test records must be greater zero!" & vbCrLf, vbOKOnly, "Error")
    Exit Sub
  End If
  wsD.Cells(2, lCol - pc_Input1 + pc_Output1).Resize(lRecord).Interior.ColorIndex = xlCILightGreen
  ReDim vInput(1 To lRecord) As Variant
  lIdx = 1
  dTypeWeight(ty_boolean) = wsD.Cells(pr_Boolean, lCol)
  dTypeWeight(ty_currency) = wsD.Cells(pr_Currency, lCol)
  dTypeWeight(ty_date) = wsD.Cells(pr_Date, lCol)
  dTypeWeight(ty_decimal) = wsD.Cells(pr_Decimal, lCol)
  dTypeWeight(ty_double) = wsD.Cells(pr_Double, lCol)
  dTypeWeight(ty_long) = wsD.Cells(pr_Long, lCol)
  dTypeWeight(ty_string) = wsD.Cells(pr_String, lCol)
  dSumWeights = 0#
  For i = LBound(dTypeWeight) To UBound(dTypeWeight)
    If dTypeWeight(i) < 0 Then sErrMsg = sErrMsg & _
      "Weight for data type " & sTypeName(i) & " must be greater equal zero!" & vbCrLf
    dSumWeights = dSumWeights + dTypeWeight(i)
  Next i
  If dSumWeights <= 0 Then sErrMsg = sErrMsg & _
    "Sum of weights for data types (Boolean, ..., String) must be greater zero!" & vbCrLf

  If Len(sErrMsg) > 0 Then
    Call MsgBox(sErrMsg & vbCrLf, vbOKOnly, "Error")
    Exit Sub
  End If
  For i = LBound(dTypeWeight) To UBound(dTypeWeight)
    dTypeWeight(i) = dTypeWeight(i) / dSumWeights * lRecord
  Next i
  'Decide how many records to generate for each data type
```

```
vType = RoundToSum(dTypeWeight, 0)

For i = LBound(vType, 1) To UBound(vType, 1)
  If vType(i) > 0 Then
    Select Case i
    Case ty_boolean
      ReDim dThisTypeWeight(1 To 2) As Double
      If Abs(wsD.Cells(pr_bTrue, lCol) + wsD.Cells(pr_bFalse, lCol)) < 0.0000000000001 Then
        'No weights means equal weights
        dThisTypeWeight(1) = vType(i) / 2
        dThisTypeWeight(2) = dThisTypeWeight(1)
      Else
        dThisTypeWeight(1) = wsD.Cells(pr_bTrue, lCol) / _
                             (wsD.Cells(pr_bTrue, lCol) + _
                             wsD.Cells(pr_bFalse, lCol)) * _
                             vType(i)
        dThisTypeWeight(2) = wsD.Cells(pr_bFalse, lCol) / _
                             (wsD.Cells(pr_bFalse, lCol) + _
                             wsD.Cells(pr_bTrue, lCol)) * _
                             vType(i)
      End If
      vThisType = RoundToSum(dThisTypeWeight, 0)
      For j = 1 To vThisType(1)
        vInput(lIdx) = True
        lIdx = lIdx + 1
      Next j
      For j = 1 To vThisType(2)
        vInput(lIdx) = False
        lIdx = lIdx + 1
      Next j
    Case ty_currency
      If IsEmpty(wsD.Cells(pr_ccyAvg, lCol)) Or IsEmpty(wsD.Cells(pr_ccyStDev, lCol)) Then
        'Work with Min and Max
        dmin = wsD.Cells(pr_ccyMin, lCol)
        dmax = wsD.Cells(pr_ccyMax, lCol)
        For j = 1 To vType(i)
          vInput(lIdx) = CCur(dmin + Rnd() * (dmax - dmin))
          lIdx = lIdx + 1
        Next j
      Else
        'Work with Avg and StDev
        ReDim dThisDouble(1 To vType(i)) As Double
        For j = 1 To vType(i)
          dThisDouble(j) = Rnd()
        Next j
        dAvg = .Average(dThisDouble)
        dStDev = .StDevP(dThisDouble)
        If dStDev < 0.0000000000001 Then
          If vType(i) = 1 Then
            vInput(lIdx) = CCur(dAvg)
            lIdx = lIdx + 1
          Else
            Call MsgBox("StDev of data type " & sTypeName(ty_currency) & _
                " must not be zero!", vbOKOnly, "Error!")
            Exit Sub
          End If
        End If
        For j = 1 To vType(i)
          vInput(lIdx) = CCur(wsD.Cells(pr_ccyAvg, lCol) + _
                          (dThisDouble(j) - dAvg) * _
                          wsD.Cells(pr_ccyStDev, lCol) / dStDev)
          lIdx = lIdx + 1
        Next j
      End If
    Case ty_date
      If IsEmpty(wsD.Cells(pr_dtAvg, lCol)) Or IsEmpty(wsD.Cells(pr_dtStDev, lCol)) Then
        'Work with Min and Max
        dmin = wsD.Cells(pr_dtMin, lCol)
        dmax = wsD.Cells(pr_dtMax, lCol)
        For j = 1 To vType(i)
          vInput(lIdx) = CDate(dmin + Rnd() * (dmax - dmin))
          lIdx = lIdx + 1
        Next j
      Else
        'Work with Avg and StDev
        ReDim dThisDouble(1 To vType(i)) As Double
        For j = 1 To vType(i)
          dThisDouble(j) = Rnd()
        Next j
        dAvg = .Average(dThisDouble)
        dStDev = .StDevP(dThisDouble)
        If dStDev < 0.0000000000001 Then
          If vType(i) = 1 Then
            vInput(lIdx) = CDate(dAvg)
            lIdx = lIdx + 1
          Else
            Call MsgBox("StDev of data type " & sTypeName(ty_date) & _
                " must not be zero!", vbOKOnly, "Error!")
            Exit Sub
          End If
```

```
      End If
      For j = 1 To vType(i)
        vInput(lIdx) = CDate(wsD.Cells(pr_dtAvg, lCol) + _
                       (dThisDouble(j) - dAvg) * _
                       wsD.Cells(pr_dtStDev, lCol) / dStDev)
        lIdx = lIdx + 1
      Next j
    End If
  Case ty_decimal
    If IsEmpty(wsD.Cells(pr_decAvg, lCol)) Or IsEmpty(wsD.Cells(pr_decStDev, lCol)) Then
      'Work with Min and Max
      dmin = wsD.Cells(pr_decMin, lCol)
      dmax = wsD.Cells(pr_decMax, lCol)
      For j = 1 To vType(i)
        vInput(lIdx) = CDec(dmin + Rnd() * (dmax - dmin))
        lIdx = lIdx + 1
      Next j
    Else
      'Work with Avg and StDev
      ReDim dThisDouble(1 To vType(i)) As Double
      For j = 1 To vType(i)
        dThisDouble(j) = Rnd()
      Next j
      dAvg = .Average(dThisDouble)
      dStDev = .StDevP(dThisDouble)
      If dStDev < 0.0000000000001 Then
        If vType(i) = 1 Then
          vInput(lIdx) = CDec(dAvg)
          lIdx = lIdx + 1
        Else
          Call MsgBox("StDev of data type " & sTypeName(ty_decimal) & _
               " must not be zero!", vbOKOnly, "Error!")
          Exit Sub
        End If
      End If
      For j = 1 To vType(i)
        vInput(lIdx) = CDec(wsD.Cells(pr_decAvg, lCol) + _
                       (dThisDouble(j) - dAvg) * _
                       wsD.Cells(pr_decStDev, lCol) / dStDev)
        lIdx = lIdx + 1
      Next j
    End If
  Case ty_double
    If IsEmpty(wsD.Cells(pr_dAvg, lCol)) Or IsEmpty(wsD.Cells(pr_dStDev, lCol)) Then
      'Work with Min and Max
      dmin = wsD.Cells(pr_dMin, lCol)
      dmax = wsD.Cells(pr_dMax, lCol)
      For j = 1 To vType(i)
        vInput(lIdx) = CDbl(dmin + Rnd() * (dmax - dmin))
        lIdx = lIdx + 1
      Next j
    Else
      'Work with Avg and StDev
      ReDim dThisDouble(1 To vType(i)) As Double
      For j = 1 To vType(i)
        dThisDouble(j) = Rnd()
      Next j
      dAvg = .Average(dThisDouble)
      dStDev = .StDevP(dThisDouble)
      If dStDev < 0.0000000000001 Then
        If vType(i) = 1 Then
          vInput(lIdx) = CDbl(dAvg)
          lIdx = lIdx + 1
        Else
          Call MsgBox("StDev of data type " & sTypeName(ty_double) & _
               " must not be zero!", vbOKOnly, "Error!")
          Exit Sub
        End If
      End If
      For j = 1 To vType(i)
        vInput(lIdx) = CDbl(wsD.Cells(pr_dAvg, lCol) + _
                       (dThisDouble(j) - dAvg) * _
                       wsD.Cells(pr_dStDev, lCol) / dStDev)
        lIdx = lIdx + 1
      Next j
    End If
  Case ty_long
    If IsEmpty(wsD.Cells(pr_lSum, lCol)) Then
      If IsEmpty(wsD.Cells(pr_lMaxRepeat, lCol)) Then
        'Work with arbitrary repetitions
        dmin = wsD.Cells(pr_lMin2, lCol)
        dmax = wsD.Cells(pr_lMax, lCol)
        For j = 1 To vType(i)
          vInput(lIdx) = Int(dmin + Rnd() * (dmax - dmin + 1))
          lIdx = lIdx + 1
        Next j
      Else
        If (wsD.Cells(pr_lMax, lCol) - wsD.Cells(pr_lMin2, lCol) + 1) * _
          wsD.Cells(pr_lMaxRepeat, lCol) < vType(i) Then
          Call MsgBox("Not enough random numbers for data type " & sTypeName(ty_long) & _
```

```
            "!", vbOKOnly, "Error!")
          Exit Sub
        End If
        v = sbRandInt(CLng(vType(i)), wsD.Cells(pr_lMin2, lCol), wsD.Cells(pr_lMax, lCol), _
            wsD.Cells(pr_lMaxRepeat, lCol))
        For j = 1 To vType(i)
          vInput(lIdx) = v(j)
          lIdx = lIdx + 1
        Next j
      End If
    Else
      v = sbLongRandSumN(wsD.Cells(pr_lSum, lCol), vType(i), _
          wsD.Cells(pr_lMin1, lCol))
      For j = 1 To vType(i)
        vInput(lIdx) = v(j)
        lIdx = lIdx + 1
      Next j
    End If
  Case ty_string
    If Not IsEmpty(wsD.Cells(pr_sLength, lCol)) Then
      'Simple string
      lLength = wsD.Cells(pr_sLength, lCol)
      If lLength <= 0 Then lLength = 1
      dmin = Asc(wsD.Cells(pr_sMin, lCol))
      dmax = Asc(wsD.Cells(pr_sMax, lCol))
      For j = 1 To vType(i)
        s = ""
        For k = 1 To lLength
          s = s & Chr(dmin + Rnd() * (dmax - dmin))
        Next k
        vInput(lIdx) = s
        lIdx = lIdx + 1
      Next j
    ElseIf Not IsEmpty(wsD.Cells(pr_sNextTabRepeat, lCol)) Then
      'Simple items from next tab
      Set wsItem = Sheets(2)
      If (wsItem.Cells(1, wsD.Cells(pr_sNextTabColumn, lCol)).End(xlDown).Row - 1) * _
        wsD.Cells(pr_sNextTabRepeat, lCol) < vType(i) Then
        Call MsgBox("Not enough random numbers for data type " & sTypeName(ty_string) & _
            "!", vbOKOnly, "Error!")
        Exit Sub
      End If
      v = sbRandInt(CLng(vType(i)), 2, _
          wsItem.Cells(1, wsD.Cells(pr_sNextTabColumn, lCol)).End(xlDown).Row, _
          wsD.Cells(pr_sNextTabRepeat, lCol))
      For j = 1 To vType(i)
        vInput(lIdx) = wsItem.Cells(1, wsD.Cells(pr_sNextTabColumn, lCol))(v(j))
        lIdx = lIdx + 1
      Next j
    Else
      'Items from weighted groups from next tab
      Set wsItem = Sheets(2)
      Set objGroup = CreateObject("Scripting.Dictionary")
      j = 2
      Do While Not IsEmpty(wsItem.Cells(j, wsD.Cells(pr_sNextTabGroupColumn, lCol)))
        objGroup.Item(wsItem.Cells(j, wsD.Cells(pr_sNextTabGroupColumn, lCol)).Value) = _
            objGroup.Item(wsItem.Cells(j, wsD.Cells(pr_sNextTabGroupColumn, lCol)).Value) + 1
        j = j + 1
      Loop
      'Are the item groups still identical to the ones in the param list?
      bGroupsUpToDate = True
      j = 0
      Do While Not IsEmpty(wsD.Cells(pr_sNextTabGroupWeights + j, pc_ItemGroups))
        If objGroup.Item(wsD.Cells(pr_sNextTabGroupWeights + j, pc_ItemGroups).Value) > 0 Then
          objGroup.Item(wsD.Cells(pr_sNextTabGroupWeights + j, pc_ItemGroups).Value) = 0
        Else
          Set objGroup = Nothing
          Set objGroup = CreateObject("Scripting.Dictionary")
          j = 2
          Do While Not IsEmpty(wsItem.Cells(j, wsD.Cells(pr_sNextTabGroupColumn, lCol)))
            objGroup.Item(wsItem.Cells(j, wsD.Cells(pr_sNextTabGroupColumn, lCol)).Value) = _
                objGroup.Item(wsItem.Cells(j, wsD.Cells(pr_sNextTabGroupColumn, lCol)).Value) + 1
            j = j + 1
          Loop
          bGroupsUpToDate = False
          Exit Do
        End If
        j = j + 1
      Loop
      If j <> objGroup.Count Then bGroupsUpToDate = False
      If Not bGroupsUpToDate Then
        Range(wsD.Cells(pr_sNextTabGroupWeights, pc_ItemGroups), _
          wsD.Cells(pr_sNextTabGroupWeights, pc_ItemGroups).End(xlDown)).ClearContents
        wsD.Cells(pr_sNextTabGroupWeights, pc_ItemGroups).Resize(objGroup.Count).FormulaArray = _
          .Transpose(objGroup.keys)
        If vbCancel = MsgBox("Item groups from next tab are not up to date!" & vbCrLf & _
          vbCrLf & "OK to continue anyway" & _
          vbCrLf & "Cancel to stop", vbOKCancel, "Warning") Then
          Exit Sub
        End If
```

```
          End If
          dSumWeights = 0#
          j = 0
          Do While Not IsEmpty(wsD.Cells(pr_sNextTabGroupWeights + j, pc_ItemGroups))
            dSumWeights = dSumWeights + wsD.Cells(pr_sNextTabGroupWeights + j, lCol)
            j = j + 1
          Loop
          ReDim dGroupWeights(1 To j) As Double
          For j = LBound(dGroupWeights) To UBound(dGroupWeights)
            dGroupWeights(j) = wsD.Cells(pr_sNextTabGroupWeights + j - 1, lCol) / dSumWeights * vType(i)
          Next j
          'Decide how many records to generate for each item group
          vGroup = RoundToSum(dGroupWeights, 0)
          For j = LBound(vGroup, 1) To UBound(vGroup, 1)
            If vGroup(j) > 0 Then
              Set wsItem = Sheets(2)
              Set objItem = CreateObject("Scripting.Dictionary")
              lRow = 2
              Do While Not IsEmpty(wsItem.Cells(lRow, wsD.Cells(pr_sNextTabGroupColumn, lCol)))
                If wsItem.Cells(lRow, wsD.Cells(pr_sNextTabGroupColumn, lCol)).Value = _
                  objGroup.keys()(j - 1) Then
                  objItem.Item(wsItem.Cells(lRow, wsD.Cells(pr_sNextTabItemColumn, lCol)).Value) = _
                      objItem.Item(wsItem.Cells(lRow, wsD.Cells(pr_sNextTabItemColumn, lCol)).Value) + 1
                End If
                lRow = lRow + 1
              Loop
              If objItem.Count * wsD.Cells(pr_sNextTabItemRepeat, lCol) < vGroup(j) Then
                Call MsgBox("Not enough random numbers for data type string, item group " & _
                     wsD.Cells(pr_sNextTabGroupWeights + j, pc_ItemGroups).Value & _
                     "!", vbOKOnly, "Error!")
                Exit Sub
              End If
              v = sbRandInt(CLng(vGroup(j)), 1, objItem.Count, wsD.Cells(pr_sNextTabItemRepeat, lCol))
              For k = 1 To vGroup(j)
                vInput(lIdx) = objItem.keys()(v(k) - 1)
                lIdx = lIdx + 1
              Next k
              Set objItem = Nothing
            End If
          Next j
          Set objGroup = Nothing
        End If
      End Select
    End If
  Next i
  'Now shuffle the result vector into random order if specified
  If wsD.Cells(pr_shuffle, lCol) Then
    lRow = 2
    For Each v In UniqRandInt(lRecord, lRecord)
      wsD.Cells(lRow, lCol - pc_Input1 + pc_Output1) = vInput(v)
      lRow = lRow + 1
    Next v
  Else
    For lRow = 2 To lRecord + 1
      wsD.Cells(lRow, lCol - pc_Input1 + pc_Output1) = vInput(lRow - 1)
    Next lRow
  End If
Next lCol
wsD.Calculate
End With

End Sub
```

Random Numbers which do not reappear too soon – sbRandomNoRepeatBeforeN

If you want to perform a random pick on some items (names, numbers, whatever), and these items can occur more than once but you need them not to re-appear within the next N draws, the function shown here will help. It calls *sbRandHistogrm* which you will need to include into your VBA code. This function is a fairly advanced example on how to use scripting dictionaries (associative arrays). It shows:

- How to attach values to names (keys)
- How to add, to remove and to change key / value pairs
- How to look up values
- How to access the whole value set as an array
- How to test whether a dictionary is empty (i.e. has no entries)
- How to access values numerically indexed (can be used to numerically index through the key or value set)

An example for N = 3:

B1 {=sbRandomNoRepeatBeforeN(A1:A7,3)}

	A	B
1	A	B
2	A	C
3	A	A
4	B	D
5	B	B
6	C	
7	D	

Please notice that for some data or for some random sequence a solution might be impossible. With the same input data as above but a different random sequence you can get:

B1 {=sbRandomNoRepeatBeforeN(A1:A7,3)}

	A	B
1	A	C
2	A	B
3	A	A
4	B	D
5	B	Error: Cannot fulfil gap condition!
6	C	
7	D	

As you can see, the single "C" and "D" have been picked already and the additional "A"s or "B" would violate the gap condition of 3 cells in B5.

sbRandomNoRepeatBeforeN Program Code

Note: This function requires (uses) the function sbRandHistoGrm (see page 312).

```
Function sbRandomNoRepeatBeforeN(rInput As Range, _
    lN As Long) As Variant
'From names in rInput we create a random draw into the
'selected cells this function is called from as an array
'function (entered with CTRL + SHIFT + ENTER) so that no
'name re-appears in the next lN cells.
'This function needs / calls sbRandHistogrm.
'(C) (P) by Bernd Plumhoff 20-Dec-2012 PB V0.10

Dim i As Long, j As Long, lDrawn As Long
Dim lCol As Long, lRow As Long
Dim obj As Object
ReDim vNames(1 To lN) As Variant
ReDim lCount(1 To lN) As Long

With Application
'Parameter check
If TypeName(.Caller) <> "Range" Then
  sbRandomNoRepeatBeforeN = CVErr(xlErrRef)
  Exit Function
End If
If .Caller.Rows.Count * .Caller.Columns.Count > rInput.Count Then
  sbRandomNoRepeatBeforeN = CVErr(xlErrValue)
  Exit Function
End If

ReDim vR(1 To .Caller.Rows.Count, 1 To .Caller.Columns.Count)

'First read in all names. They may appear more than once.
Set obj = CreateObject("Scripting.Dictionary")
For i = 1 To rInput.Count
  obj.Item(rInput(i).Value) = obj.Item(rInput(i).Value) + 1
Next i

'Now apply the draws. After each draw take drawn name out
'for next lN draws.
i = 1
For lRow = 1 To UBound(vR, 1)
  For lCol = 1 To UBound(vR, 2)
    If obj.Count > 0 Then
      lDrawn = sbRandHistogrm(0#, UBound(obj.Items), obj.Items)
      vR(lRow, lCol) = obj.Keys()(lDrawn)
      If vNames(1) <> "" Then
        'Need to add in again a name
        obj.Add vNames(1), lCount(1)
      End If
      For j = 1 To lN - 1
        vNames(j) = vNames(j + 1)
        lCount(j) = lCount(j + 1)
      Next j
      If obj.Items()(lDrawn) > 1 Then
        vNames(lN) = obj.Keys()(lDrawn)
        lCount(lN) = obj.Items()(lDrawn) - 1
      Else
        vNames(lN) = ""
        'lCount(lN) = 0 'Not necessary but clean
      End If
      obj.Remove obj.Keys()(lDrawn)
      i = i + 1
    Else
      vR(lRow, lCol) = "Error: Cannot fulfil gap condition!"
    End If
  Next lCol
Next lRow
sbRandomNoRepeatBeforeN = vR
End With
End Function
```

Excursus

Calculating Probabilities – Drawing Cards With and Without Replacement

If you draw 7 cards without replacement from a full deck of 52 playing cards, what is the probability that you will have 3 aces in your hand?

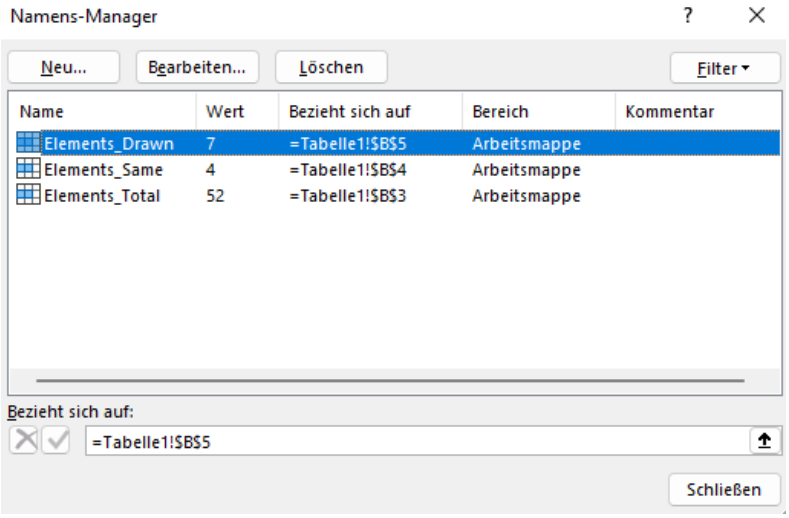
The answer is: approximately 0.58%.

	A	B	C	D	E	F	G	H
1	**Draw 7 of 52 Cards (with or without Replacement)**							
2								
3	Cards total count	**52**						
4	Aces total	**4**						
5	Cards drawn	**7**						
6								
7	**Likelihoods**	**With Replacement**	**Runs**	**No ace**	**1 Ace**	**2 Aces**	**3 Aces**	**4 Aces**
8	**Formula**	WAHR		57,10%	33,31%	8,33%	1,16%	0,10%
9	**Monte Carlo**	WAHR	10000	57,52%	32,87%	8,49%	1,04%	0,08%
10	**Formula**	FALSCH		55,04%	36,69%	7,68%	0,58%	0,01%
11	**Monte Carlo**	FALSCH	10000	54,43%	37,40%	7,63%	0,53%	0,01%

The exact formula for the probability without replacement in Excel 365 or Excel 2021/2024 is:
*=IFERROR(COMBIN(Elements_Drawn, SEQUENCE(1, Elements_Same + 1, 0, 1)) * (Elements_Same / Elements_Total)^SEQUENCE(1, Elements_Same + 1, 0, 1) * IFERROR((1 - Elements_Same / Elements_Total)^(Elements_Drawn - SEQUENCE(1, Elements_Same + 1, 0, 1)), 1), 0)*

With replacement, the formula is:
*=IFERROR(COMBIN(Elements_Same, SEQUENCE(1, Elements_Same + 1, 0, 1)) * COMBIN(Elements_Total - Elements_Same, Elements_Drawn - SEQUENCE(1, Elements_Same + 1, 0, 1)) / COMBIN(Elements_Total, Elements_Drawn), 0)*

The following names have been defined:



Approximately, you can also determine these probabilities using a Monte Carlo simulation:

monte Program Code

```
Function monte(bWithReplacement As Boolean, _
    Optional runs As Long = 100000) As Variant
'(C) (P) by Bernd Plumhoff  27-Oct-2022 PB V0.2
Dim i As Long, j As Long, n As Long
Dim lAces As Long, lCards As Long
Dim lCardsDrawn As Long, lCardsSame As Long, lCardsTotal As Long
Dim r(1 To 5) As Variant
With Application.WorksheetFunction
lCardsTotal = Range("Elements_Total")
lCardsSame = Range("Elements_Same")
lCardsDrawn = Range("Elements_Drawn")
Randomize
For i = 1 To runs
  n = 0
  For j = 1 To lCardsDrawn
    If bWithReplacement Then
      lCards = lCardsTotal
      lAces = lCardsSame
    Else
      lCards = lCardsTotal + 1 - j
      lAces = lCardsSame - n
    End If
    If .RandBetween(1, lCards) < 1 + lAces Then
      n = n + 1
      If n = lCardsSame Then Exit For
    End If
  Next j
  r(1 + n) = r(1 + n) + 1
Next i
For i = 1 To lCardsSame + 1: r(i) = r(i) / runs: Next i
monte = r
End With
End Function
```

Fun without Practical Relevance for Advanced Learners

A Simple VBA Pivot Table – *sbMiniPivot*

"It's not what you look at that matters, it's what you see." [Henry David Thoreau]

Note: Please notice that Excel 365 and Excel 2024 now offer two similar but more powerful worksheet functions than *sbMiniPivot* - GROUPBY and PIVOTBY. Once your Excel can use these functions, this user-defined function *sbMiniPivot* should only be used for VBA training purposes.

	A	B	C	D	E
1	**Input**				
2	a	1	a		
3	a	2	b		
4	a	3	c		
5	b	4	d		
6	b	5	e		
7	c	6	f		
8	c	7	g		
9	c	8	h		
10	c	9	i		
11					
12	**Output numerical minimum**			**Output string minimum**	
13	a	1		a	a
14	b	4		b	d
15					
16	**Output numerical maximum**			**Output string maximum**	
17	a	3		a	c
18	c	9		c	i
19					
20	**Output numerical sum**				
21	a	6			
22	b	9			
23	c	30			

Worksheet Formulas		
Range	Formula	
A13	**A13**	=sbMiniPivot("min", A2:A10<>"c", A2:A10,B2:B10)
A17	**A17**	=sbMiniPivot("max",A2:A10<>"b", A2:A10,B2:B10)
A21	**A21**	=sbMiniPivot("sum", TRUE,A2:A10,B2:B10)
D13	**D13**	=sbMiniPivot("min",A2:A10<>"c", A2:A10,C2:C10)
D17	**D17**	=sbMiniPivot("max",A2:A10<>"b", A2:A10,C2:C10)

Name

sbMiniPivot - Concatenate, count, sum or return min or max of last given input column for all combinations of the previous ones where same row of condition column is *True*.

Synopsis

sbMiniPivot(*sFunction*, *vCondition*, ParamArray *vInput*)

Description

sbMiniPivot performs the function *sFunction* on last given column of *vInput* for all combinations of the previous ones where corresponding elements of *vCondition* are *True*. It returns a variant array.

Options

sFunction - Specifies the function which has to be applied to the combinations. Can be *concatenate* (*cat*), *count*, *max*(*imum*), *min*(*imum*), *sum*.

vCondition - Condition constant *True* or *False* or column which needs to contain *True*/*False* values.

vInput - Two or more columns. *sFunction* will be applied on last input column for all combinations of the previous ones where same row of condition column is *True*.

sbMiniPivot Program Code

```
Function sbMiniPivot(sFunction As String, vCondition As Variant, _
  ParamArray vInput() As Variant) As Variant
'sbMiniPivot performs the function sFunction on last given column of
'vInput() for all combinations of the previous ones where corresponding
'elements of vCondition are TRUE.
'Example:
'    A     B     C
' 1 Smith Adam   1
' 2 Myer  Ben    3
' 3 Smith Ben    2
' 4 Smith Adam   7
' 5 Myer  Ben    4
'Now array-enter into D1
'=sbMiniPivot("sum", B1:B5="Ben", A1:A5,B1:B5,C1:C5) and you will get
'    D      E    F
' 1 Myer   Ben   7
' 2 Smith  Ben   2
'(C) (P) by Bernd Plumhoff 31-Jul-2022 PB V1.1
Dim b As Boolean, bCondition As Boolean
Dim i As Long, j As Long, k As Long, liscount As Long
Dim lvdim As Long, lcdim As Long
Dim obj As Object
Dim s As String, sC As String
Dim vR As Variant

With Application.WorksheetFunction
sC = "|"
k = 0
vInput(0) = .Transpose(.Transpose(vInput(0)))
If LCase(sFunction) = "count" Then liscount = 1
If UBound(vInput) < 1 - liscount Then
  sbMiniPivot = CVErr(xlErrValue)
  Exit Function
End If
lvdim = UBound(vInput(0))
Select Case VarType(vCondition)
Case vbBoolean
  bCondition = True
Case vbArray + vbVariant
  bCondition = False
```

```
  vCondition = .Transpose(.Transpose(vCondition))
  lcdim = UBound(vCondition, 1)
  If lcdim <> lvdim Then
    sbMiniPivot = CVErr(xlErrRef)
    Exit Function
  End If
Case Else
  sbMiniPivot = CVErr(xlErrNA)
  Exit Function
End Select
If lvdim > 100 Then lvdim = 100 'Let us start with small dimension
On Error GoTo ErrHdl
ReDim vR(0 To UBound(vInput) + liscount, 1 To lvdim)
For j = 1 To UBound(vInput)
  vInput(j) = .Transpose(.Transpose(vInput(j)))
  If UBound(vInput(0)) <> UBound(vInput(j)) Then
    sbMiniPivot = CVErr(xlErrRef)
    Exit Function
  End If
Next j
Set obj = CreateObject("Scripting.Dictionary")
For i = 1 To UBound(vInput(0))
  b = bCondition
  If Not b Then b = vCondition(i, 1)
  If b Then
    s = vInput(0)(i, 1)
    For j = 1 To UBound(vInput) - 1 + liscount
      s = s & sC & vInput(j)(i, 1)
    Next j
    If obj.Item(s) > 0 Then
      Select Case LCase(sFunction)
      Case "cat", "concatenate"
        vR(UBound(vInput), obj.Item(s)) = vR(UBound(vInput), _
          obj.Item(s)) & "," & vInput(UBound(vInput))(i, 1)
      Case "count"
        vR(UBound(vInput) + 1, obj.Item(s)) = vR(UBound(vInput) + 1, _
          obj.Item(s)) + 1
      Case "max", "maximum"
        If vR(UBound(vInput), obj.Item(s)) < vInput(UBound(vInput))(i, 1) Then
          vR(UBound(vInput), obj.Item(s)) = vInput(UBound(vInput))(i, 1)
        End If
      Case "min", "minimum"
        If vR(UBound(vInput), obj.Item(s)) > vInput(UBound(vInput))(i, 1) Then
          vR(UBound(vInput), obj.Item(s)) = vInput(UBound(vInput))(i, 1)
        End If
      Case "sum"
        vR(UBound(vInput), obj.Item(s)) = vR(UBound(vInput), _
          obj.Item(s)) + vInput(UBound(vInput))(i, 1)
      Case Else
        sbMiniPivot = CVErr(xlErrRef)
      End Select
    Else
      k = k + 1
      obj.Item(s) = k
      For j = 0 To UBound(vInput)
        vR(j, k) = vInput(j)(i, 1)
      Next j
      If liscount = 1 Then vR(UBound(vInput) + 1, k) = 1
    End If
  End If
Next i
'Reduce result array to used area
If k > 0 Then ReDim Preserve vR(0 To UBound(vInput) + liscount, 1 To k)
sbMiniPivot = .Transpose(vR)
Set obj = Nothing
End With
Exit Function

ErrHdl:
If Err.Number = 9 Then
  If i > lvdim Then
    'Here we normally get if we breach Ubound(vR,2)
    lvdim = 10 * lvdim 'So we need to increase last dimension
    If lvdim > UBound(vInput(0)) Then lvdim = UBound(vInput(0))
    ReDim Preserve vR(0 To UBound(vInput) + liscount, 1 To lvdim)
    Resume 'Back to statement which caused error
  End If
End If
On Error GoTo 0 'Other error - terminate
Resume
End Function
```

TEXTJOIN

The Excel function TEXTJOIN can, starting from Excel version 2019, combine the contents of cell ranges and arrays as text strings and output them with a specified delimiter. For Excel versions older than Excel 2019, you can use the custom function TEXTJOIN presented here.

The good thing is: Once you switch to Excel 2019 or later, you don't need to do anything. Excel will automatically use the built-in TEXTJOIN function.

Please note: I deliberately chose not to address pathological cases such as delimiters being an array or ignoring blanks containing whole numbers like 0 or 1. However, I also did not copy the errors from Excel's TEXTJOIN. For example, Excel's TEXTJOIN function handles overlaps of non-contiguous ranges, like '`(A1:C3,D4:F6,G7:I9) (A1:B2,C3:D4,E5:F6,G7:H8,I9:J10)`', incorrectly. As with almost every analogy, I had to and wanted to draw a boundary.

TEXTJOIN Program Code

```
Function TEXTJOIN(Delimiter As String, _
    Ignore_empty As Boolean, _
    ParamArray Text() As Variant) As String
'(C) (P) by Bernd Plumhoff 07-Jan-2022 PB V1.1
Dim v, i As Long, s As String, t As String
For i = LBound(Text) To UBound(Text)
  If IsArray(Text(i)) Then
    For Each v In Text(i)
      t = IIf(IsMissing(v), "", v)
      If Not (Ignore_empty And t = "") Then
        TEXTJOIN = TEXTJOIN & s & t
        s = Delimiter
      End If
    Next v
  Else
    t = IIf(IsMissing(Text(i)), "", Text(i))
    If Not (Leer_ignorieren And t = "") Then
      TEXTJOIN = TEXTJOIN & s & t
      s = Delimiter
    End If
  End If
Next i
End Function
```

Index

#Const.....23, 26, 271, 282, 293, 296
#Else 23, 24, 70, 282
#End If..... 23, 24, 25, 26, 27, 70, 272, 282, 293, 294, 296, 298
#If.... 23, 24, 25, 26, 27, 70, 271, 272, 282, 293, 294, 296, 297
1904 date system
  Excel Option.....42
7zip 40
Aberth, Oliver 68
Access Rights 148
Accounts Receivable Problem 155
accumulated trade blotter 125
actual costs.....91
AGGREGATE.....42
Alabama.....266, 267
AlgoOne 241
Allocation of Overheads 261
American_Down_and_Out_Call_GABV.....205
American_Put_EJGABV 204
American_Put_GABV.....204
American_Put_MBTV 202
American_Put_Option_bCNFDM 211
American_Put_Option_bEFDM 209
American_Put_Option_bIFDM 209
American_Spread_Call_Option_bTvB 206
annuities 199
Application.Version 33
ApplicationVersion 22, 33
Apprentice 288
AppVersion 21
array
  associative 348
array formula.....199
array function 349
ArrayDim.....34
article number 84
Asset Allocation 113
associative array 15
associative arrays 348
Baeldung.....223
bank holiday 52
beta
  outlier resistant 114
binary representation.....47
Binomial Method.....201
Birthday List 123
`Black_Scholes_Delta` 220
`Black_Scholes_Gamma` 220
Board Game Risk..... 299
Boxenstopps 141, 142
Breakdown
  linear 93
Brownian bridge 325
budget..... 116
  distribution 116
  planning 116
budget control 91
budget plan..... 91
Business Process 11
cake..... 307
Calendarweek 90
camels..... 75
car racing 141
catch errors..... 14
Cauchy 320
CEILING 89
cell based chart..... 109
CHAR 2
chart
  cell based 109
Check digits..... 84
Cholesky..... 329, 330, 331, 336, 337
Cholesky decomposition..... **329**, 332
Chris Strickland 201
Circle Constant π..... 61
circle method 145
circular references
  accidental..... 42
class.....15, 16, 17, 18, 20, 23, 26, 264, 265
Clewlow 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220
Codename
  Worksheet 21
coding convention 15
Collatz 118
Collatz conjecture 118
Collatz length 118
combinations 225
Combinations with Subsets k of n 131
compare correlation matrices 241
compiler constants 281
Conover..... 331, 336, 339
ConvertTime 84, 153
correlated random numbers 329, 331
correlation matrices 241
Count Cells with certain color..... 112

count trailing zeros 98
CountDown variant 222
Courier New 96
cursor 12, 13, 17
Cursor 17, 18, 19
Daily Data 94
data analyser 158
Data Analysis 158
database 233
Datastats 167, 168, 169, 188, 189
DATE 2
DATEDIF 75
Debug Modus 195, 196
DEC2BIN 47
Declaration, Variables 12
degree of freedom 94
degrees of freedom 307
D'Hondt 280
Diaconis 280
digit
  significant 92
DisplayAlerts 17, 18, 19
documentation 15
Drawing Cards 350
dynamic nested loops 226
Eliminate Points 121
EnableAnimations 17, 18, 19
EnableEvents 17, 18, 19
encapsulation 42
EOMONTH 75
Equations, linear 71
Euklidean Algorithm 76, 77
European Article Number 84
European_Arithmetic_Asian_Call_bMCwGAOCV 219
European_Call_bMCBS 212
European_Call_bMCBS_ADCV 215
European_Call_bMCBS_ADGCV 216
European_Call_bMCBS_AVR 213
European_Call_bMCBS_DCV 214
European_Call_GABV 203
European_Call_MBTV 202
European_Call_Option_bEFDM 208
European_Call_Option_bTT 207
European_Down_and_Out_Call_bMCBS 219
European_Spread_bMCBS 217
European_Spread_bMCBS_SV 218
Evaluate Spreadsheet Formulas 86
Excel Add-in 42
Excel Color Index 293, 342
Excel Don't's 42
ExportAllPics 30
Extreme Cell Values 150
Farrer 66
FastExcel 200
Fichtenholz 64, 66
filling gaps 98, 100, 102, 129
Finite Difference Methods 207
FLOOR 89
Font
  monospaced 96
Fractions 68, 74, 75
Freedman 280
freeware 42
Function Key
  [F9] 86
Function Key CTRL + g 13
Function Key F2 12
Function Key F5 13
Function Key F8 13
Function Key SHIFT + F8 13
fwim 89
Geburtstagsliste 123
getOperatingSystem 22
Goldberg 42
good program 15
GROUPBY 352
Hare-Niemeyer 266, 280
Header 169, 179, 180
Heap sort 222
HSTACK 258
IF 2
Iman 331, 336, 339
Iman-Conover Method **331**, 336
Implementing Derivatives Models 201
Increment Lowest Significant Digit 92
INDEX 200
IndexX 335, 336, 337
INDIRECT 2, 34, 42
Interactive 17, 18, 19
interpolation 98, 100, 102, 129
Interpreter 195, 196
inverse distribution function 321, 322
IstFeiertag 52, 53
Iteration
  Excel Option 42
joint heirship 74
justified
  text 96
Krabat 288
Lambda expression 258
largest remainder method. 256, 257, 264, 265, 266, 267
Learning Path 2
Les Clewlow 201

LibFileTools 22, 23, 25, 40
Limits 170, 171, 177, 178, 188, 189
limits file 162
Linear Breakdown 93
linear combination 76
Linear equations 71
List Named Ranges 32
log files 20, 21
Logger 16, 20, 21, 22, 23, 24, 25, 26, 40
logging 16, 20, 21, 22, 23, 25, 26
LOOKUP variants 133
loop with variable nesting depth 226
Lorentz 320
Lucida Sans Typewriter 96
Lundberg 14
lwim 89
mass storage 233
math test 289
matrix function 349
matrix multiplication 127
Merge two columns into one
  text 96
Michael Schwimmer 155
MID 2
Mildenhall 331
  Stephen J. 331
Mini Calculator 192
minimal number of banknotes and coins 136
Minimum Truck Load Problem 97
Minirechner 192
MOD 89
Monospaced Font 96
monotonously increasing 99
Monte Carlo 260, 291, 293, 296, 300, 303, 304, 351
Monte Carlo Simulation
  Options 212
Monthly Data 94
mortality annuities 199
mortality table 199
Most frequent pairs 127
movement analysis 159
MS SQL Server 233
named ranges 32
naming convention 15
nested loops with recursion 226
Nievergelt 66
NORM.INV 305
NORM.S.INV 305, 332
normally distributed 268, 269, 270, 278
NPV 200
number systems 43
OFFSET 200
optimal pitstops 141
option 14
Option Explicit 12, 23, 25, 26
Optionen 353
Options 201
outlier 114
overhead costs 255, 261
pairings table 145
Peltier 2
PERCENTILE 100
PERCENTRANK 100
Perl 241
permutation algorithm 222
permutations 224
PI 68
Pivot Table 352
PIVOTBY 352
planned costs 91
Portfolio rebalancing 139
PowerShell 27
pre-calculated table 200
Precision as Displayed
  Excel Option 42
Preußler, Otfried 288
PrintCommunication 17, 18, 19
probability 264, 265, 312, 315, 316, 350
profiling 16
Programming Environment 11
pseudo random numbers 281
Quickperm 222, 223
quota changes 74
Ramer–Douglas–Peucker
  algorithm 121
`RandCorr` 221
Random Corridors 327
random integers .. 281, 282, 283, 285, 286, 287
Random Numbers 259, 263, 278, 279, 281
RandomShuffle 332, 334, 336, 337
Range 29
  export 29
  non-contiguous 355
rank
  unique 119
rational number 68, 69, 70, 75
Read Access 148
Rebalancing
  Portfolio 139
Recalculation 111
recorded macro 15
ReDim Preserve 14, 304
redw 313, 314
REFA 103
REFA Time Slicing 103

Regatta Flight Plan....295
Reingold....66
Rights....104
RMLinks.cfg
AolgoOne file....241
Rnd....290
Roles....104
Roles and Rights....104
ROUND....2, 94, 105
Round2Sum....258
Rounding
sum preserving....255
rounding errors....277, 278
RoundToSum....**255**, 256, 257, 261, 264, 265, 266, 267, 269, 270, 273, 275, 276, 277, 279, 280
rww....313
same weekday and calendarweek last year..90
samples....270, 278
Sande....280
sbAccumulatedTadeBlotter....125
sbAllocate....113
sbBin2Dec....47, 48, 49
sbBinNeg....48, 49, 50
sbBirthdayList....123
sbCellWatermarks....150
sbCollatz....118
sbCountMyColor....112
sbDatastats....158, 163
sbDec2Bin....47, 48
sbDecAdd....48, 49, 51
sbDistBudget....116, 117
sbDivBy2....48, 49, 50
sbEAN....84, 85
sbEuklid....76, 77
sbExactRandHistogrm....264, 265
sbExportRange2Picture....29
sbFairStaffSelection....266, 267
sbGenerateTeams....291, 292, 293
sbGenerateTestData....291, 336, 338, 342
sbGenNormDist....305, 306
sbGetCell....34, 36, 37, 38, 39
sbGrowthSeries....325, 326
sbInterp....101, 129, 130, 131
sbLongRandSumN....285, 346
sbMinCash....137
sbMinCoins....136
sbMiniPivot....14, 352, 353
sbMonthNumber....58, 59, 60
sbMostFrequentPairs....127
sbNamedRanges....32
sbNextFloat....40
sbNRN....68, 69, 70, 74, 75
sbNSig....57
sbNum2Str....56
sbOptimalVacationDays....144
sbParseNumSeq....66, 67
sbRandCauchy....320
sbRandCDFInv....321, 322
sbRandCumulative....323, 324
sbRandGeneral....309, 311, 322, 323
sbRandHistogrm....312, 348
sbRandInt....281, 283, 284, 346, 347
sbRandIntFixSum....286, 287, 325
sbRandomNoRepeatBeforeN....348
sbRandPDF....322
sbRandSum1....307, 308
sbRandTriang....287, 315, 318, 319, 321, 328
sbRandTrigen....316, 318, 319
sbRebalancedReturn....139
sbReducePoints....121
sbRegattaFlightPlan....295, 296
sbSpellNumber....43, 44, 46
sbSumMyFormat....111
sbSWV....269, 270, 271, 272
sbTimeAdd....81, 82, 83
sbTimeDiff....78, 79, 80, 81
sbTskList....152
sbUniqRank....119
sbZip....40
Schubert, Hermann....61
Schwimmer
Michael....155
ScreenUpdating....17, 18, 19
scripting dictionaries....348
Seasonal Weights....95
security threat....42
significant digit....92
Significant digits....57
Smoothing Inner Period Values....108
spell numbers....43
Spreadsheet Formulas
Evaluate....86
SQL Server....233
StatusBar....17, 18, 19
Stop....13
stratified sample....321, 322
Strickland....202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220
subsets of n with k elements....131
Sum Numbers with same number format..111
SUMPRODUCT....2, 42
SystemState....17, 18, 54, 141, 149, 153, 267, 291, 292, 293, 295, 296, 300, 301, 303, 304, 336, 343

tasklist....152
test....16, 43
Test Access Rights....148
TEXT....2, 26, 27, 68
TEXTJOIN....355
TEXTVERKETTEN....355
Time Slicing....103
TODAY....2
TRANSPOSE....258, 271
TREND....102
triangular distribution....315, 316, 318, 328
Trinomial Trees....207
Ullman....66
UniqRandInt 281, 282, 283, 291, 292, 293, 294, 295, 296, 297, 347
unique rank....119
vacation....274, 275
Vacation Days
  Optimal Usage....143
values
  extreme....114
  smoothen....108
Variable Declaration....12
Variables....**11**
VBA Editor
  Blue flag....12
  Breakpoint....13
  Immediate Window....13
VLOOKUP....93, 94
volatile....2
Weekday
  first or last in month....89
WEEKDAY....90
Weight Calculation....224
Weight Sum
  identical....224
Weights
  seasonal....95
Wicklin
  Rick....331
work units....276
Worksheet....22, 261, 262
  Codename....21, 156, **266**
  internal VBA name....21
  Name....156, **266**
Worksheet formula....42
Write Access....148
Write-Log
  PowerShell....27
xlCalculationAutomatic....17, 18
xlCalculationManual....17, 19
xlCalculationSemiautomatic....17
xlDefault....17, 18
xlErrNA....136
xlErrNull....136
xlErrNum....136
xlErrValue....136
XLOOKUP....2
xlWait....17
π 61, 63, 68