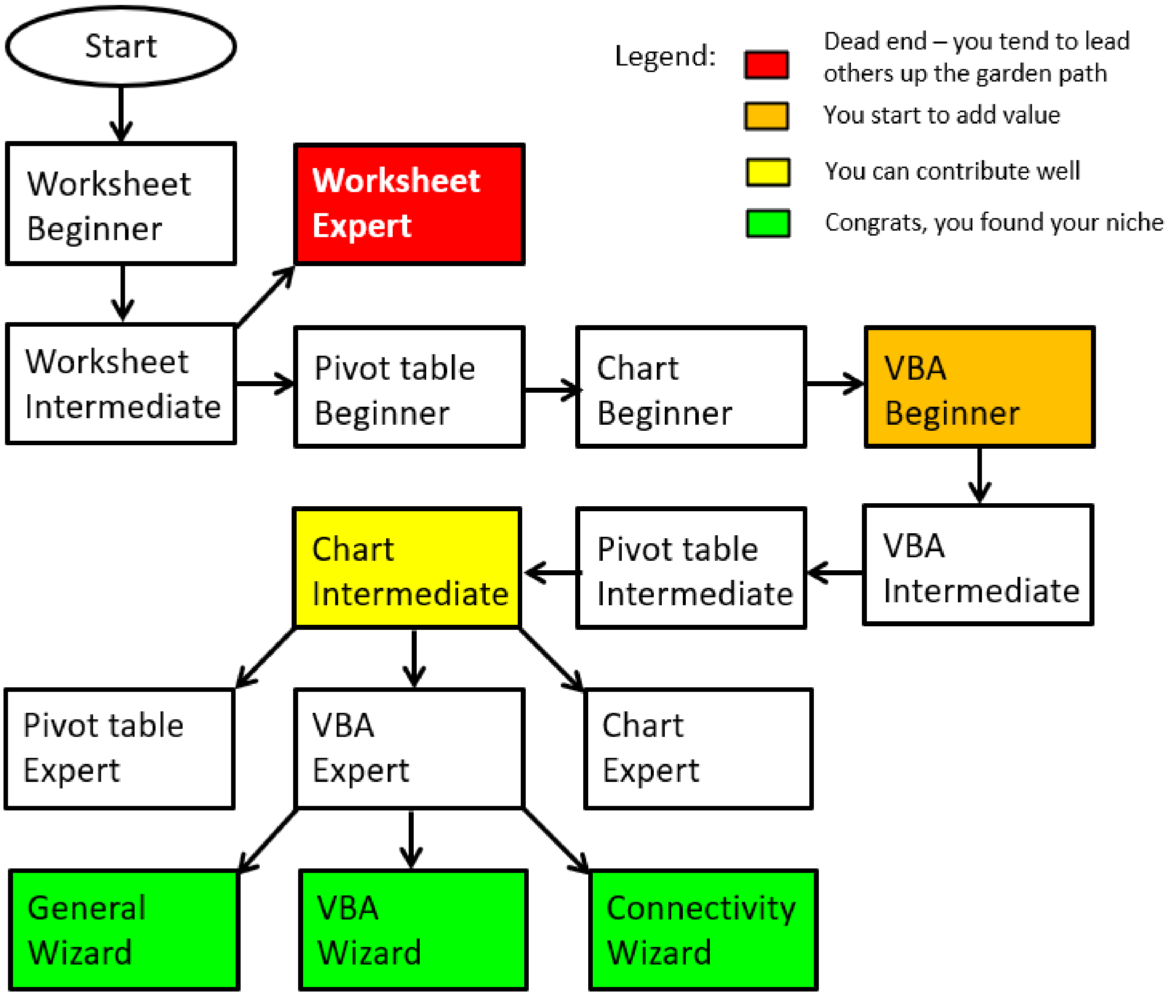
Excel / VBA – A Collection

Excel Careers



Abstract

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

Bernd Plumhoff, 8-Jan-2025

Table of Contents

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.

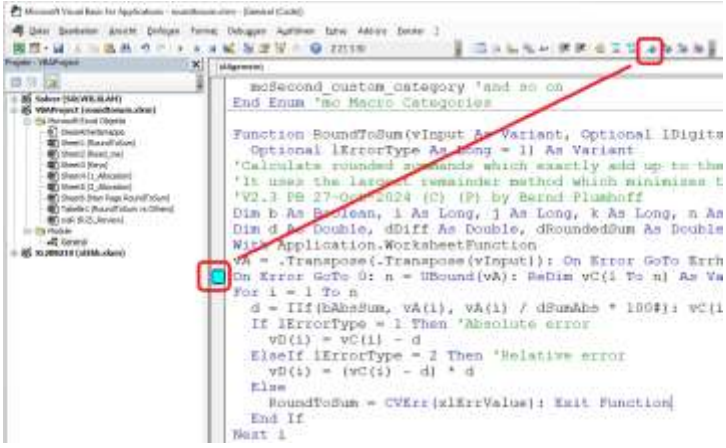
Basics

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.

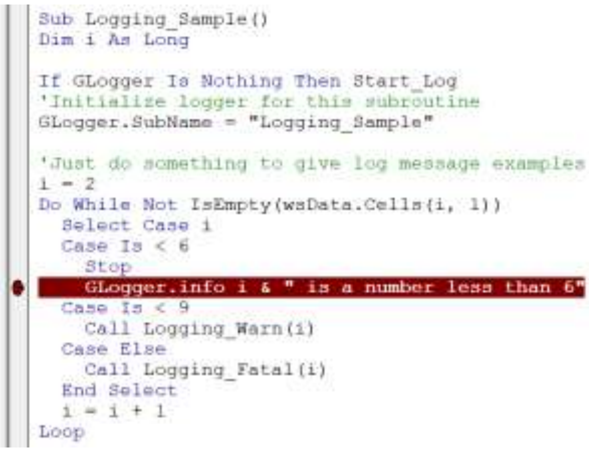
When the cursor is located at a variable or at an object during editing you can go to its declaration with t SHIFT plus the function key F2 (SHIFT + F2). This works for Dim, not ReDim. With STRG + SHIFT + F2 you can jump back to where you were coming from.

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:



```
  moSecond_custom_category 'and so on
End Enum 'mo Macro Categories

Function RoundToSum(vInput As Variant, Optional lDigits
  Optional lErrorType As Long = 1) As Variant
'Calculate rounded summands which exactly add up to the
'It uses the largest remainder method which minimises t
'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
Dim d As Double, dDiff As Double, dRoundedSum As Double
With Application.WorksheetFunction
vA = .Transpose(.Transpose(vInput)): On Error GoTo Errh
On Error GoTo 0: n = UBound(vA): ReDim vC(1 To n) As Va
For i = 1 To n
  d = IIf(bAbsSum, vA(i), vA(i) / dSumAbs * 100#): vC(i
  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
```

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

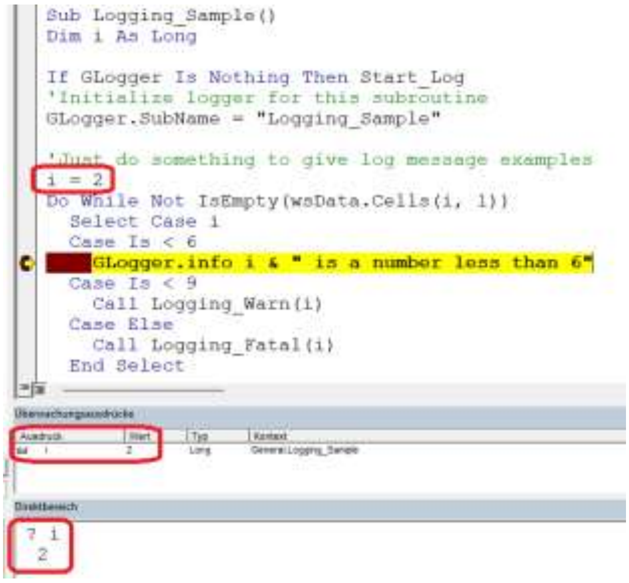


```
Sub Logging_Sample()
Dim i As Long

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.

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://de.wikibooks.org/wiki/VBA_in_Excel/_Namenskonventionen
https://learn.microsoft.com/de-de/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.

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

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
'Bitte beachten: Dies kann NICHT mit "Dim state as New SystemState"
abgekürzt werden!
```

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	Wahr, Falsch	Set to *False* to turn off system prompts.
EnableAnimations	Wahr, Falsch	From Excel 2016 onwards, this disables Excel’s screen animations.
EnableEvents	Wahr, Falsch	Set to *False* to prevent event procedures from executing.
Interactive	Wahr, Falsch	Best not to touch – unless you want to prevent all keyboard inputs.
PrintCommunication	Wahr, Falsch	Set to *False* to change page settings without waiting for a response from the printer.
ScreenUpdating	Wahr, Falsch	Set to *False* to turn off screen updates during program execution.
StatusBar	Falsch, “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.
'Source (EN): http://www.sulprobil.com/systemstate_en/
'Source (DE): http://www.bplumhoff.de/systemstate_de/
'(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 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.
'Version When         Who             What
'      1 Once upon .. Cliff G.        Initial version
'     13 12-Nov-2024  Bernd Plumhoff  Using LibFileTools https://github.com/cristianbuse/VBA-FileTools,
ApplicationVersion updated
#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
'Need to import for next 3 lines: LibFileTools https://github.com/cristianbuse/VBA-FileTools
sPath = GetLocalPath(ThisWorkbook.Path) & "\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
'     13 12-NOv-2024  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
```

ShowExcel 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 Dimensionen of an Array – *ArrayDim*

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

ArrayDim Programmcode

```
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
```

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_9"

Sub sbZip(ByVal vSourceFullPathName As Variant, _
  ByVal vDestinationZipFullPathName As Variant, _
  Optional bCreate As Boolean = True, _
  Optional bUse7zip As Boolean = True)
'Create zip file vDestinationZipFullPathName and insert zipped file or folder vSourceFullPathName.
'Version When         Who    What
'      1 24-Nov-2020  EotG   Original downloaded from https://exceloffthegrid.com/vba-cod-to-zip-unzip/
'      6 17-Dec-2020  Bernd  ByVal to enforce variants, single file feature and parameter bCreate added
'      7 25-Apr-2024  Bernd  lRepeat to avoid endless loops and parameter 16 for CopyHere to avoid
'                            confirmation prompt. No error checking.
'      8 12-Sep-2024  Bernd  Use a valid empty zip template if it exists.
'                            Workaround in case the print sequence fails.
'      9 29-Dec-2024  Bernd  New option bUse7zip and using Logger and LibFileTools
'                            https://github.com/cristianbuse/VBA-FileTools.
Dim iFile             As Integer
Dim lItems            As Long
Dim lRepeat           As Long
Dim sLine             As String
Dim sShellCmd         As String
Dim oExec             As Object
Dim oOutput           As Object
Dim oShell            As Object

If GLogger Is Nothing Then Call Start_Log
If bCreate And Not IsFile(CStr(vDestinationZipFullPathName)) Then
  If Not DeleteFile(CStr(vDestinationZipFullPathName)) Then
    GLogger.warn "Could not delete file '" & vDestinationZipFullPathName & "'"
  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 IsFile(GetLocalPath(ThisWorkbook.Path) & "Zip_Template.zip") Then
    'Workaround in case print sequence in Else clause does not work
    CopyFile ThisWorkbook.Path & "\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
  On Error Resume Next
```

```
    lItems = oShell.Namespace(vDestinationZipFullPathName).Items.Count
    On Error GoTo 0

    Set oShell = CreateObject("Shell.Application")
    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
```

Number Systems, Formats, and Transformations

Abstract

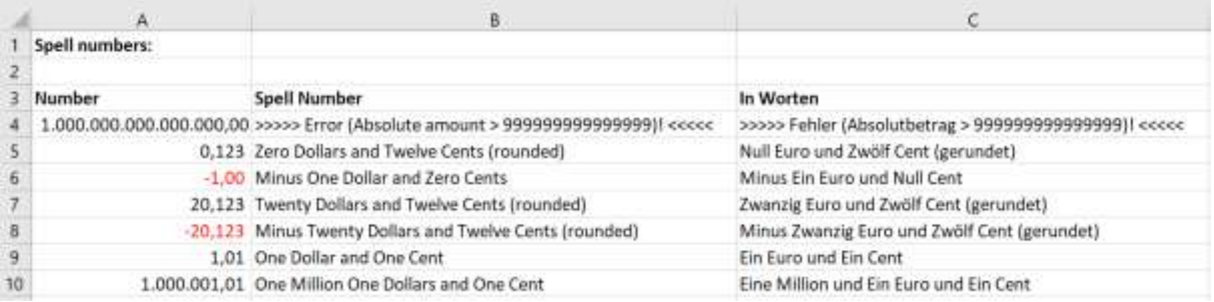
As a programmer, one often deals with number systems and their representation or conversion. Here, I present some programs that I have come to know and use over time.

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 a Decimal Number into its Binary Equivalent or Vice Versa – 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
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111110101101111101101010001111110011101111001011110010000100001110101100100101001100110100 0100
110011110101000010101010010111100111111100
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

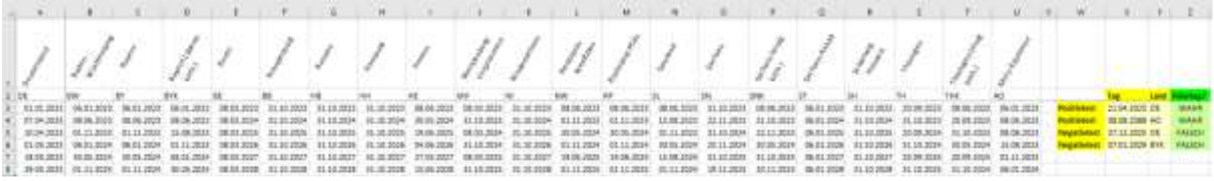
```
Function sbDec2Bin(ByVal sDecimal As String, _
               Optional lBits As Long = 32, _
               Optional blZeroize As Boolean = False) As String
'Convert a decimal number into its binary equivalent.
'(C) (P) by Bernd Plumhoff 18-Dec-2021 PB V0.4
Dim sDec As String, sFrac As String
Dim 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?



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
```

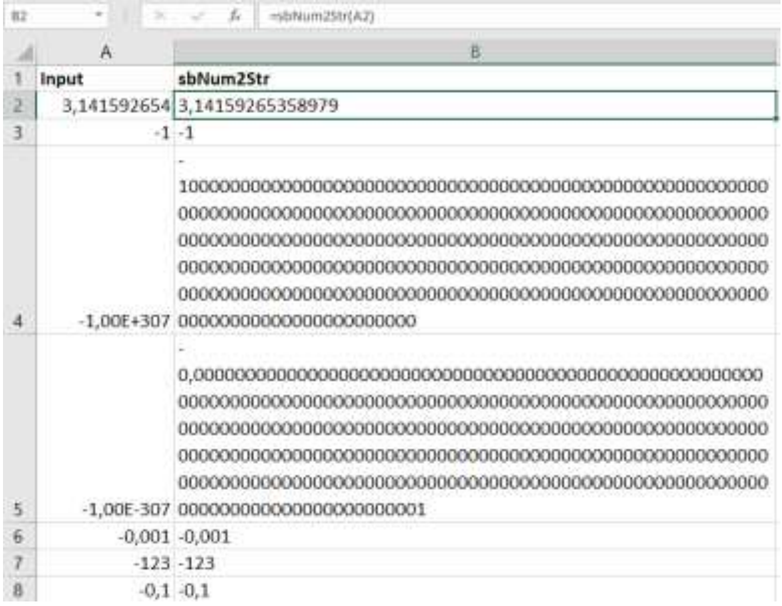
Feiertagsliste_erstellen Code

```
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)

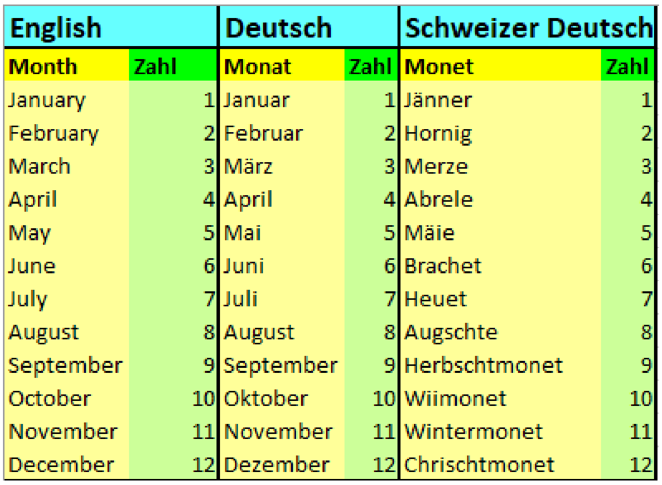
	A	B
1	Input	sbNum2Str
2	3,141592654	3,14159265358979
3	-1	-1
4	-1,00E+307	-10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
5	-1,00E-307	-0,0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
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
```

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:

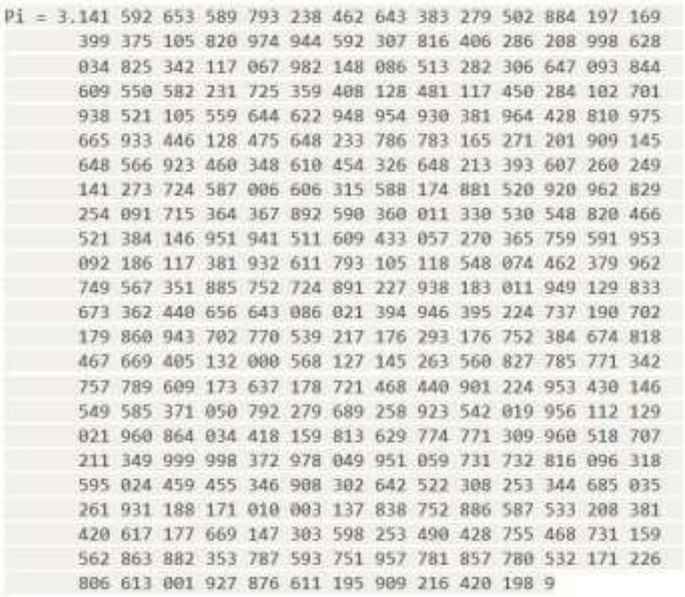
$$(*)\ \arctan x + \arctan y = \arctan \frac{x+y}{1-xy}$$

Pi 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
```

The 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:

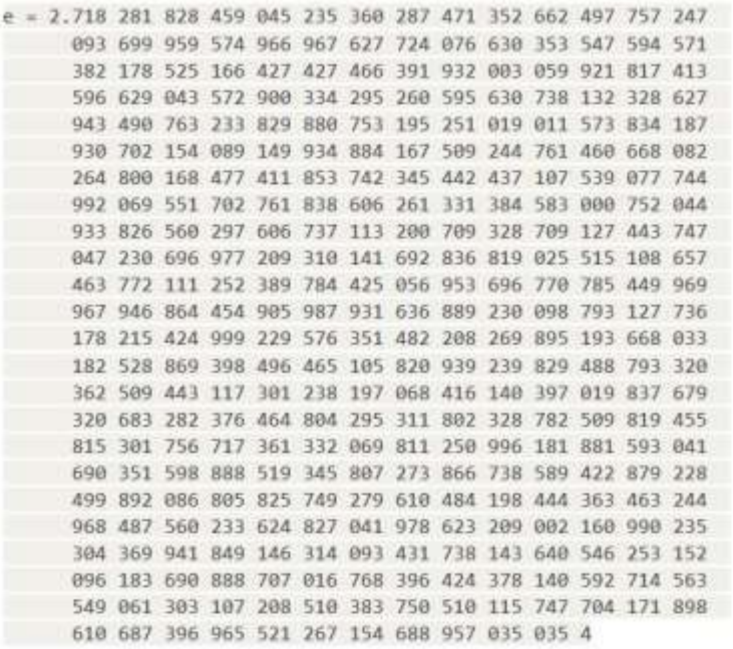
$$e - y_n < \frac{1}{n!n} < 10^{-1000}$$

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

e 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

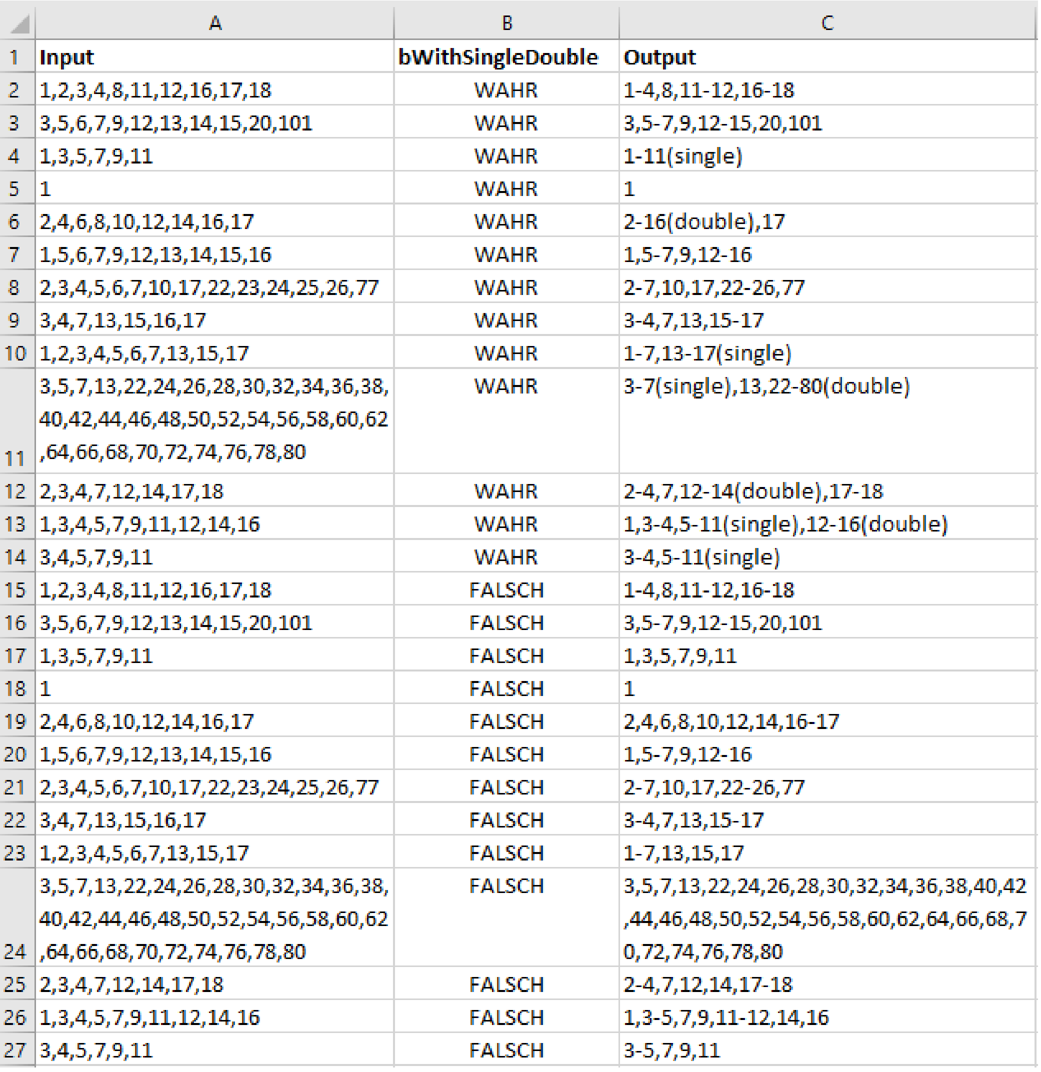
Nievergelt, Farrer, Reingold: Computer Approaches to Mathematical Problems; Prentice Hall, Inc. 1974, p. 191 + 192, p. 198 - 202.

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.

Return a Shortened Representation of a Number Sequence – 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
```

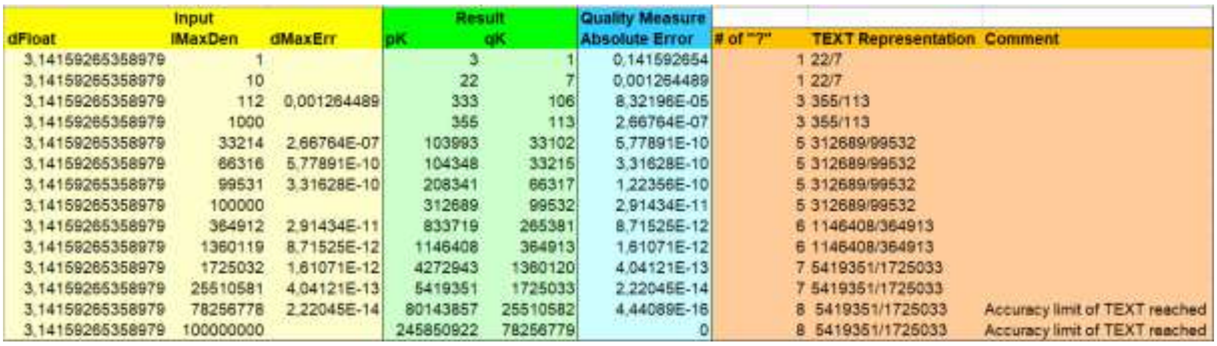
Rational Numbers = Fractions

Compute 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 "7"	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.

Calculation Limits

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

Literature

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

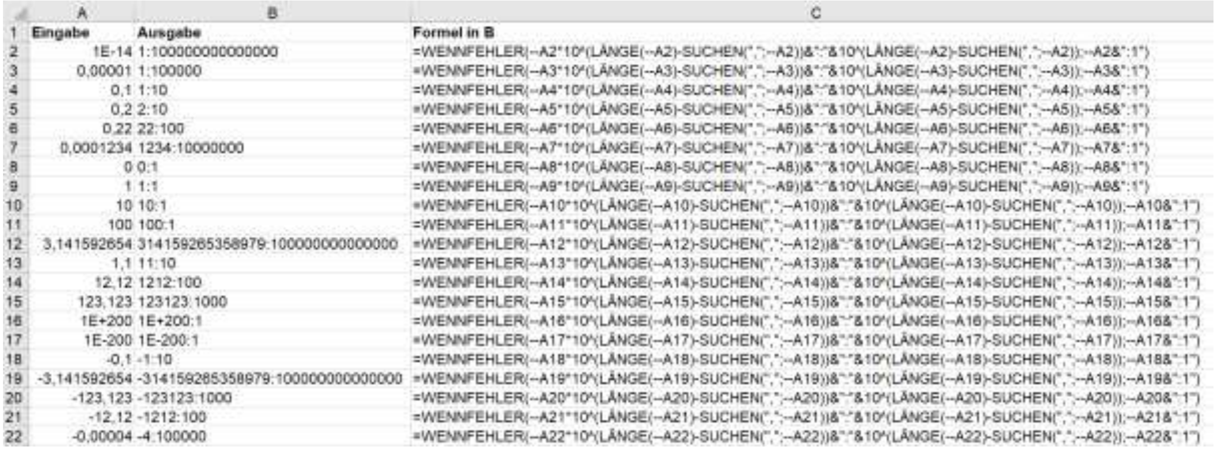
George Chrystal, Algebra an Elementary Text-Book, Part II, Chapter 32, p. 423 ff, 1900

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

Excursus

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")
```



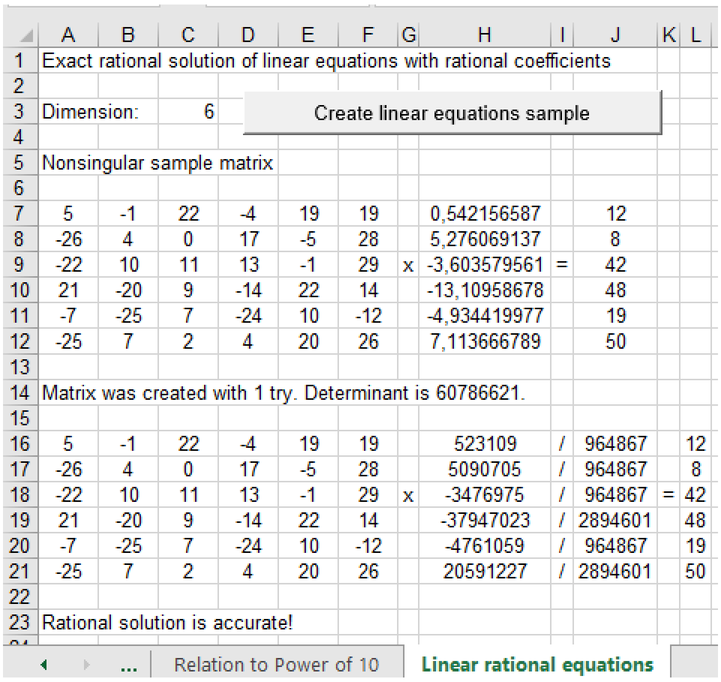
	A	B	C
1	Eingabe	Ausgabe	Formel in B
2	1E-14	1:100000000000000	=WENNFEHLER(--A2*10^(LÄNGE(--A2)-SUCHEN(",";--A2))&":"&10^(LÄNGE(--A2)-SUCHEN(",";--A2));--A2&":1")
3	0,00001	1:100000	=WENNFEHLER(--A3*10^(LÄNGE(--A3)-SUCHEN(",";--A3))&":"&10^(LÄNGE(--A3)-SUCHEN(",";--A3));--A3&":1")
4	0,1	1:10	=WENNFEHLER(--A4*10^(LÄNGE(--A4)-SUCHEN(",";--A4))&":"&10^(LÄNGE(--A4)-SUCHEN(",";--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!											

Sample Code

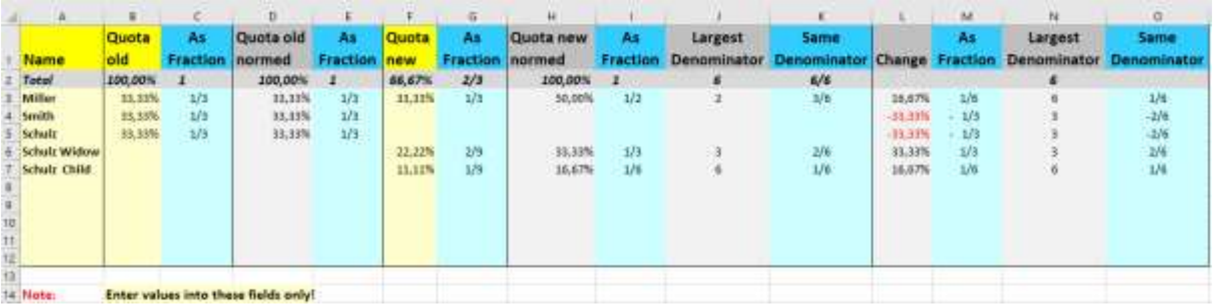
```
Sub Generate_linear_equations_and_solve()
'Calculates next rational numbers to the double-precision solution of
'given linear equations. Accuracy of rational solution is then reported.
'(C) (P) by Bernd Plumhoff 05-Jun-2024 PB V0.1

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:

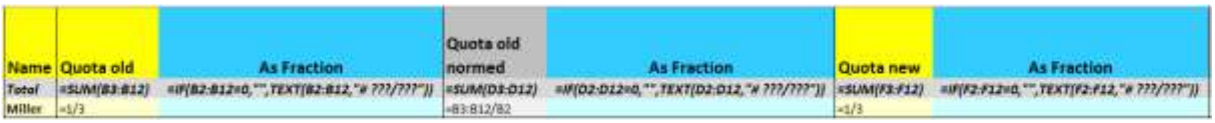
Example: Miller, Smith, and Schulz form a joint heirship. Smith dies without an heir. His quota will be distributed. Schulz also dies. His widows will receive 2/3 of his quota, his only child will get 1/3. Please note that you need to enter this as =1/3 * 2/3 resp. =1/3 * 1/3 whereby the first 1/3 represents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



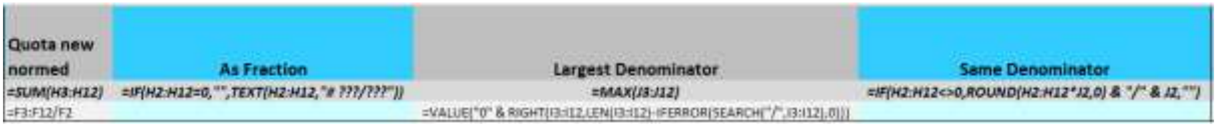
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Name	Quota old	As Fraction	Quota old normed	As Fraction	Quota new	As Fraction	Quota new normed	As Fraction	Largest Denominator	Same Denominator	Change	As Fraction	Largest Denominator	Same Denominator
2	*Total*	*100,00%*	*1*	*100,00%*	*1*	*66,67%*	*2/3*	*100,00%*	*1*	*6*	*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

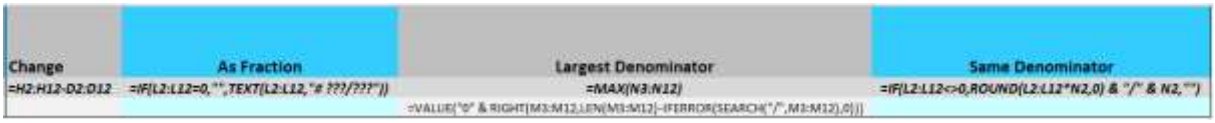
You can solve this problem easily with Excel spreadsheet formulas by using Excel's internal representation of fractions as text:



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)))	



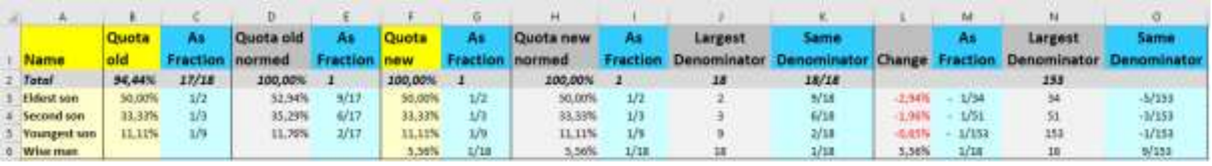
Change	As Fraction	Largest Denominator	Same Denominator
=H2:H12-D2:D12	*=IF(L2:L12=0,"",TEXT(L2:L12,"# ???/???"))*	*=MAX(N3:N12)*	*=IF(L2:L12<>0,ROUND(L2:L12*N2,0) & "/" & N2,"")*
		=VALUE("0" & RIGHT(M3:M12,LEN(M3:M12)-IFERROR(SEARCH("/",M3:M12),0)))	

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:



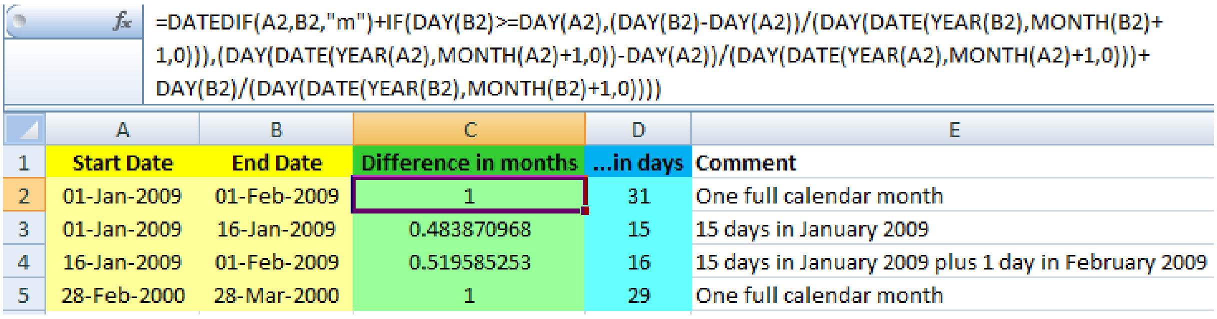
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Name	Quota old	As Fraction	Quota old normed	As Fraction	Quota new	As Fraction	Quota new normed	As Fraction	Largest Denominator	Same Denominator	Change	As Fraction	Largest Denominator	Same Denominator
2	*Total*	*94,44%*	*17/18*	*100,00%*	*1*	*100,00%*	*1*	*100,00%*	*1*	*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)))
```

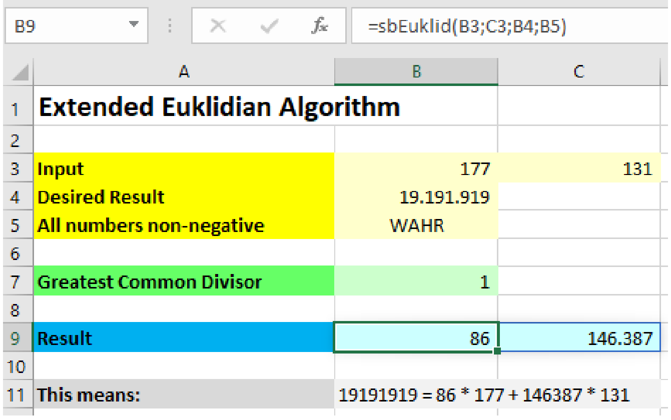
If you like to represent this result as a rational number I suggest to use sbNRN.

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

```
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

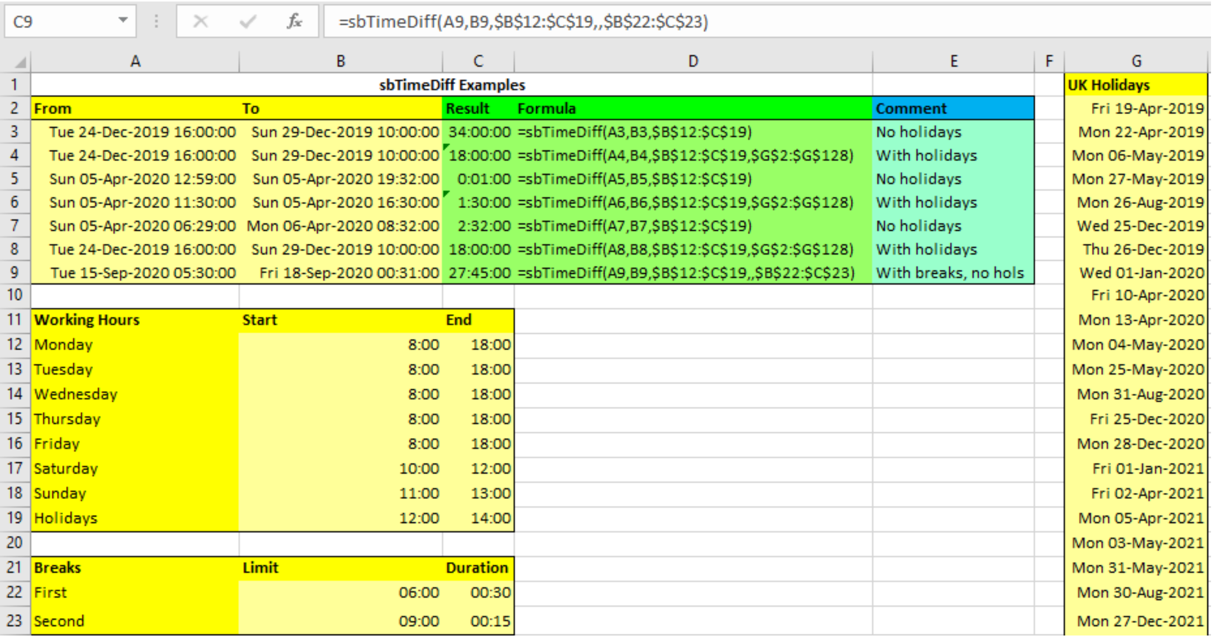
dtTo - Datetime to count to

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

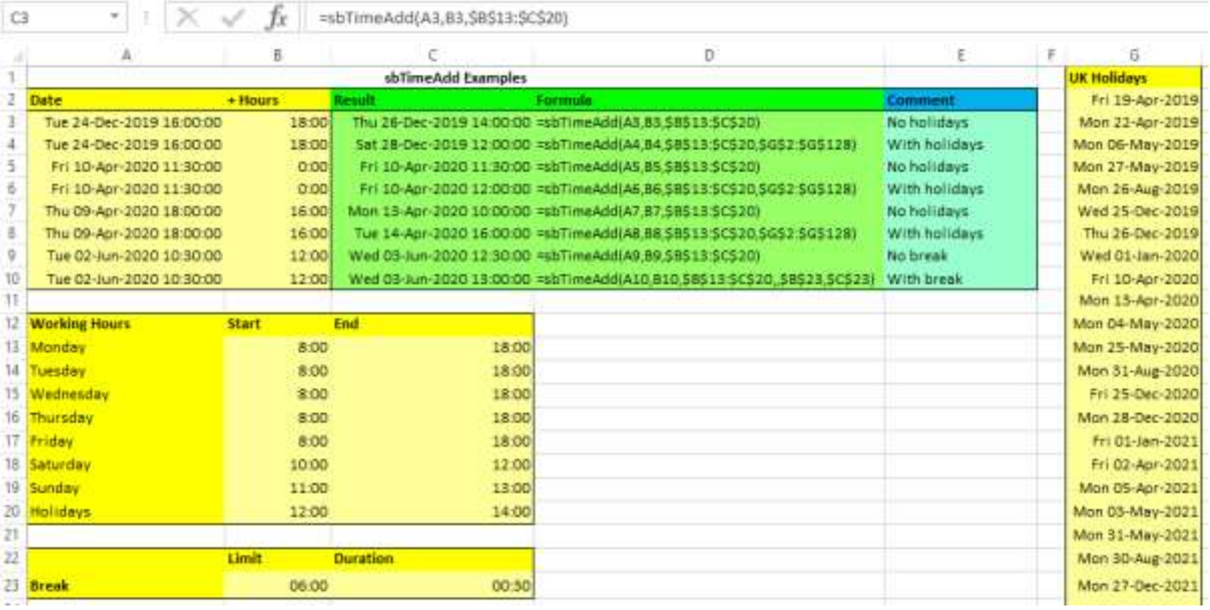
dh - Hours (plus minutes) of type Double to add to *dt* *vwh* - 8 by 2 matrix defining start time and end time for each weekday and for holidays, first row for Mondays, 8th row for holidays

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

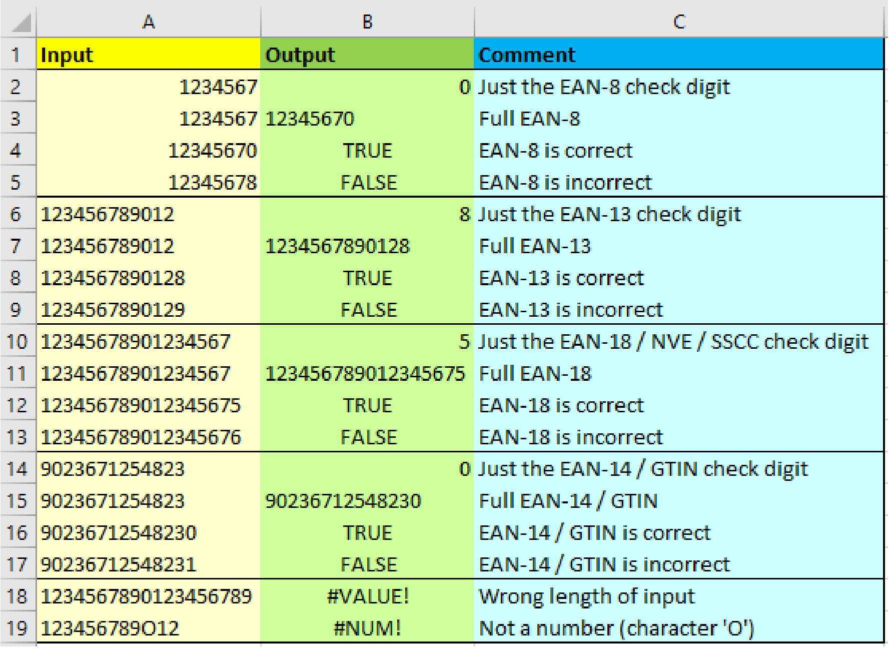
Please note that this solution requires MS Outlook to be installed properly.

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("thstndrdthththththth",8),2*MOD(A1,100)+1,2)
```

Rounding Values Preserving Their Sum with *RoundToSum* (Excel / VBA)

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 decimal 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)
    )
)
```

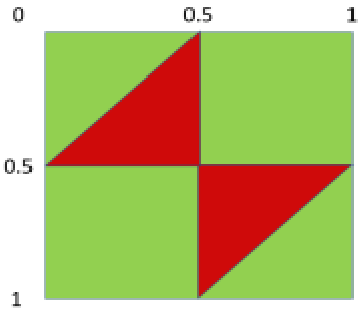
UniqRank is defined as:

```
=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))
        )
    )
)
```

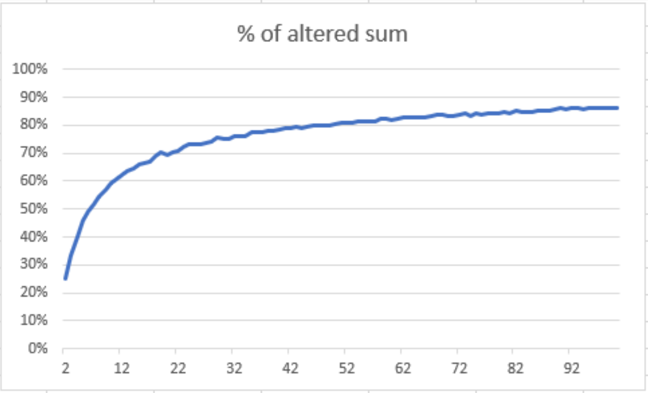
Rounding Values Alters Their Sum



How likely is it that a sum of rounded values is not identical to their rounded sum?

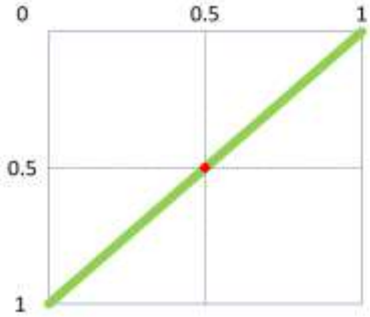
For two random floating point numbers this is obvious: The likelihood is around 25% - that is the percentage of **red** in this picture:

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.

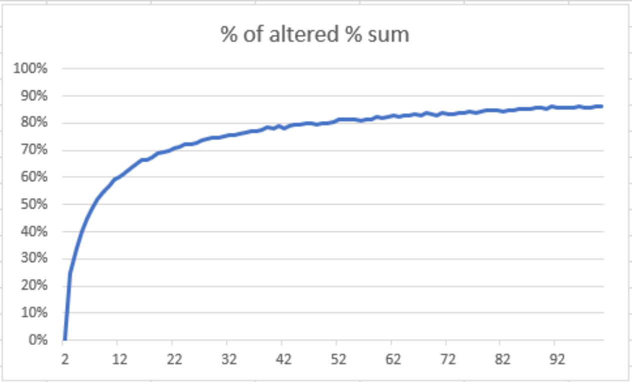
Rounded Percentages



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.
'Source (EN): https://www.sulprobil.com/rounding_values_alters_their_sum_en/
'Source (DE): https://www.bplumhoff.de/werte_runden_aendert_ihre_summe_de/
'(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.
'Source (EN): https://www.sulprobil.com/rounding_values_alters_their_sum_en/
'Source (DE): https://www.bplumhoff.de/werte_runden_aendert_ihre_summe_de/
'(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
        eFehler! Textmarke nicht definiert.(k) = Rnd()
      Else
        eFehler! Textmarke nicht definiert.(k) = 2# * Rnd() - 1#
      End If
      s1 = s1 + eFehler! Textmarke nicht definiert.(k)
    Next k
    s2 = 0#
    For k = 1 To i
      eFehler! Textmarke nicht definiert.(k) = .Round(1000# * e(k) / s1, 0)
      s2 = s2 + eFehler! Textmarke nicht definiert.(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*

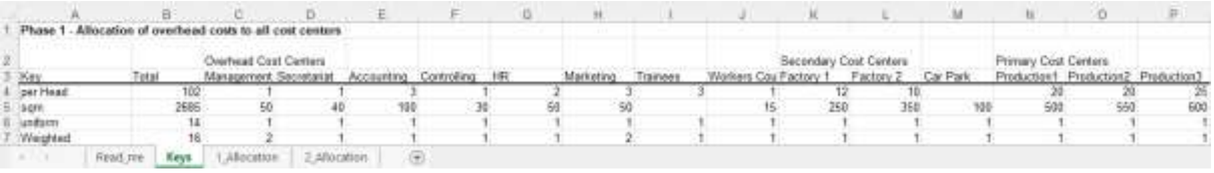
Allocation of Overheads

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.

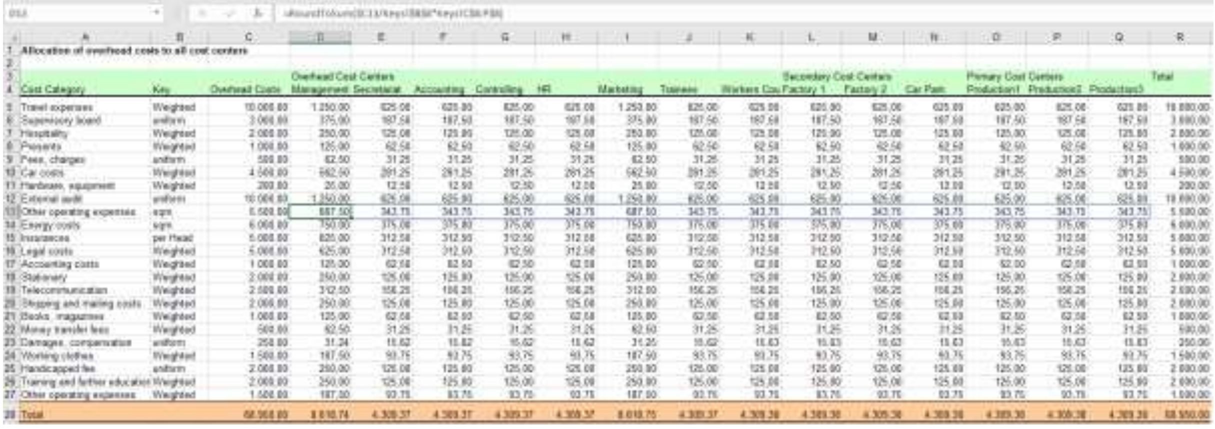
First you define how the overheads have to be allocated to support cost centres (sheet 'Keys'):



Phase 1 - Allocation of overhead costs to all cost centers

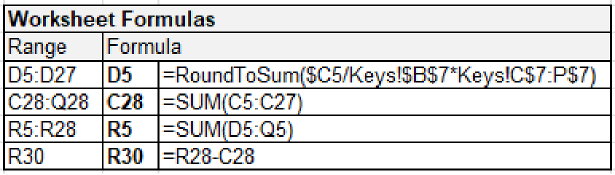
		Overhead Cost Centers								Secondary Cost Centers			Primary Cost Centers		
Key	Total	Management	Secretariat	Accounting	Controlling	HR	Marketing	Trainees	Workers Cou	Factory 1	Factory 2	Car Park	Production1	Production2	Production3
per Head	102	1	1	3	1	2	3	3	1	12	10		20	20	25
sqm	2685	50	40	100	30	50	50		15	250	350	100	500	550	600
uniform	14	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Weighted	16	2	1	1	1	1	2	1	1	1	1	1	1	1	1

The first allocation of overheads uses a rounding correction so that all summands accurately sum up on support cost centre level (sheet '1_Allocation'):



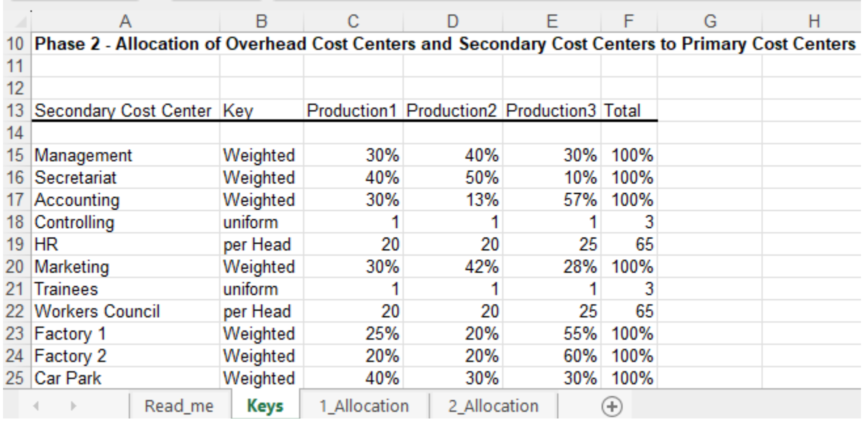
Allocation of overhead costs to all cost centers

			Overhead Cost Centers								Secondary Cost Centers			Primary Cost Centers			Total
Cost Category	Key	Overhead Costs	Management	Secretariat	Accounting	Controlling	HR	Marketing	Trainees	Workers Cou	Factory 1	Factory 2	Car Park	Production1	Production2	Production3	
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
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
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
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
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
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
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
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
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
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
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
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
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
Stationery	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
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
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
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
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
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
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
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
Training and further education	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
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
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



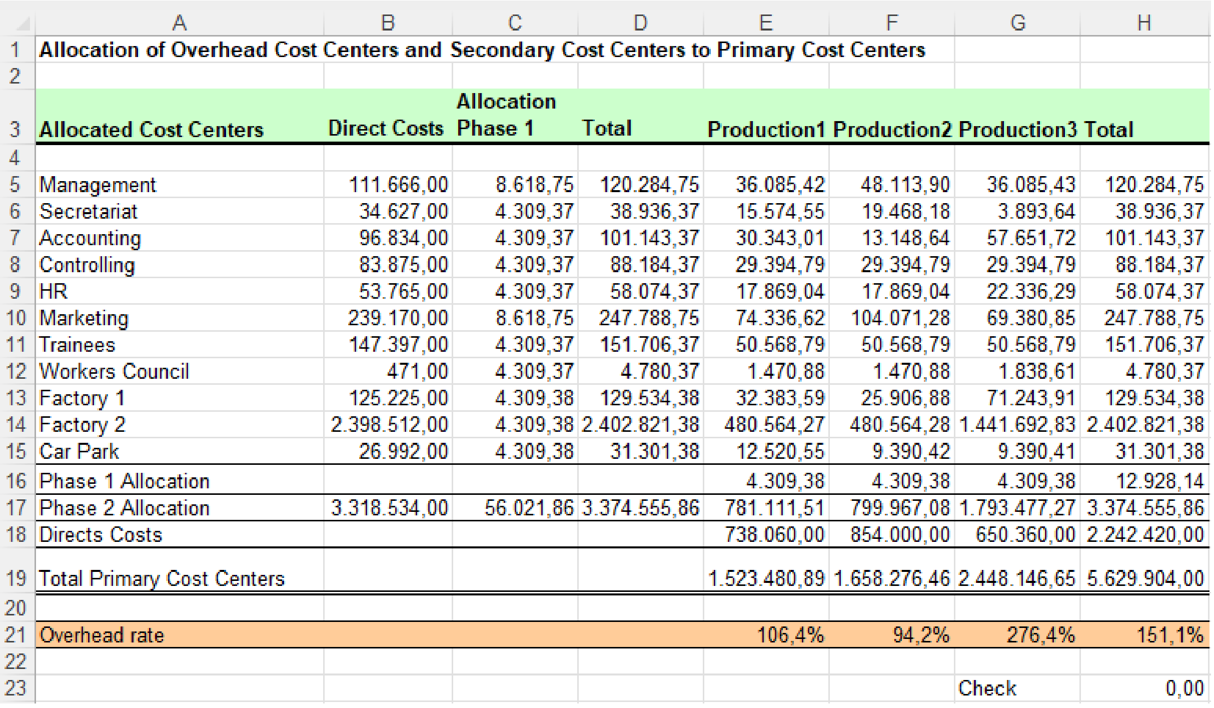
Worksheet Formulas		
Range	Formula	
D5:D27	**D5**	=RoundToSum($C5/Keys!$B$7*Keys!C$7:P$7)
C28:Q28	**C28**	=SUM(C5:C27)
R5:R28	**R5**	=SUM(D5:Q5)
R30	**R30**	=R28-C28

The second allocation of overheads (sheet ‘Keys’) also uses a rounding correction so that all support cost centres get accurately distributed to products:

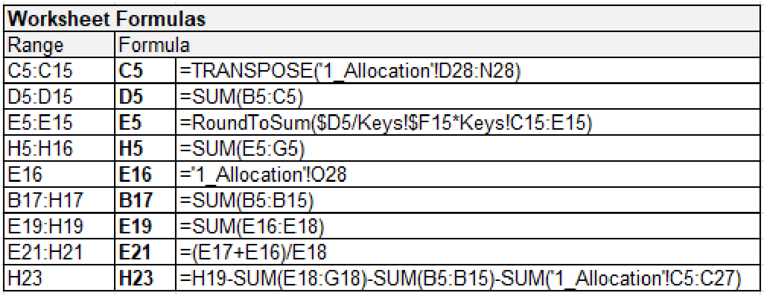


	A	B	C	D	E	F	G	H
10	**Phase 2 - Allocation of Overhead Cost Centers and Secondary Cost Centers to Primary Cost Centers**							
11								
12								
13	Secondary Cost Center	Key	Production1	Production2	Production3	Total		
14								
15	Management	Weighted	30%	40%	30%	100%		
16	Secretariat	Weighted	40%	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%		

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)

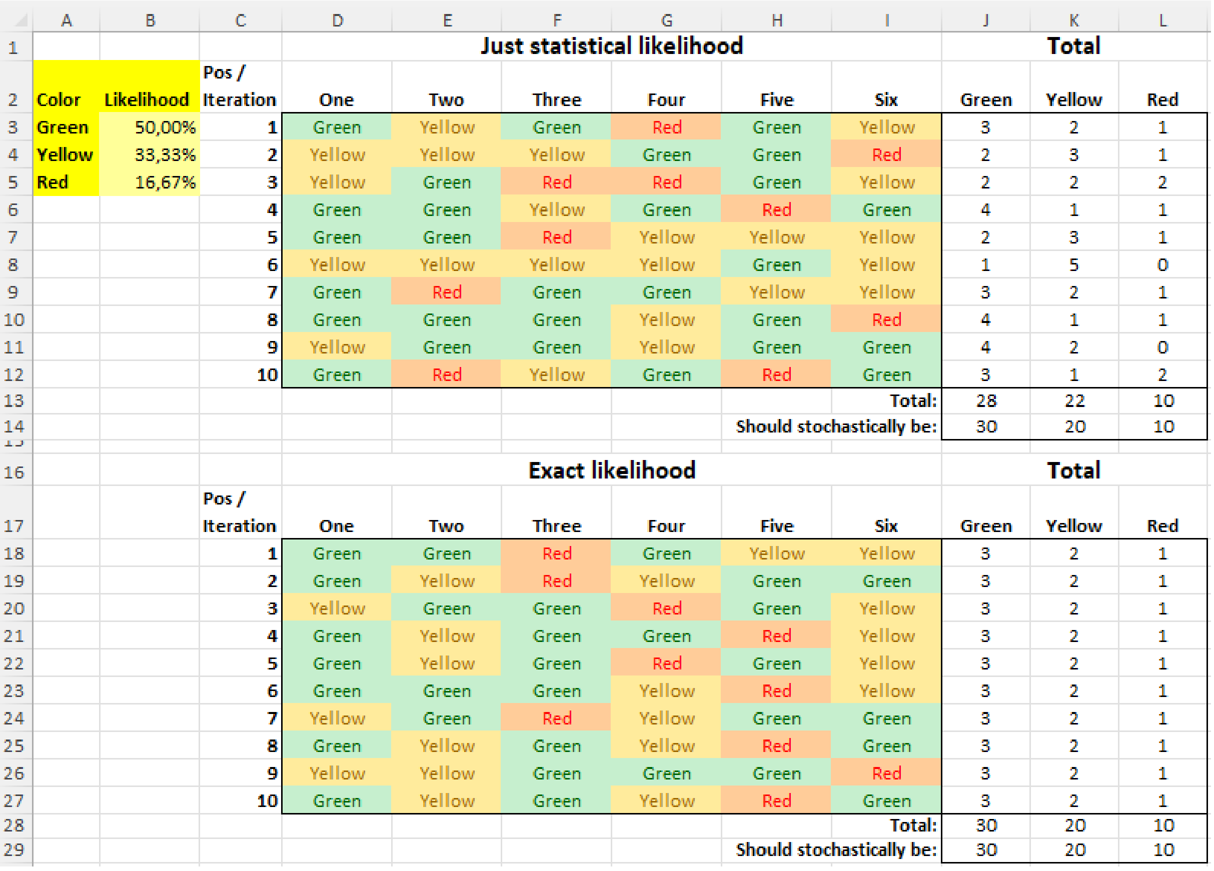
This correct allocation of overheads you will be able to enter into a general ledger without any cent / penny difference.

Example of an Exact Relation of Random Numbers

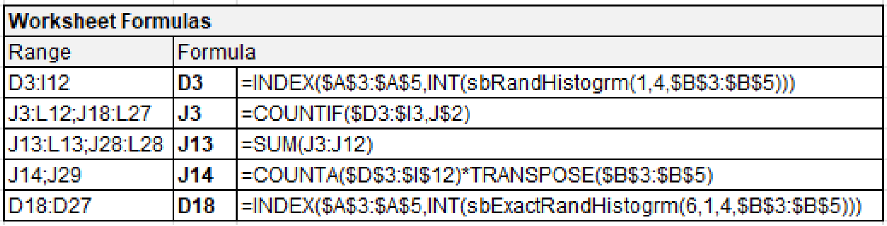
It is fairly easy to create a loaded die, let us say on average the 6 should appear twice as often as all the other numbers 1 thru 5: Enter into A1: *=MIN(INT(RAND()*7+1),6)*

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.
'Source (EN): http://www.sulprobil.de/sbexactrandhistogrm_en/
'Source (DE): http://www.berndplumhoff.de/sbexactrandhistogrm_de/
'(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

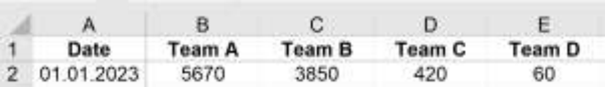
Let us assume your company needs to get some special tasks done. All staff members can do the work. You want the teams to second their staff based on the size of each team. This selection can be done by the user-defined function *RoundToSum*.

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.

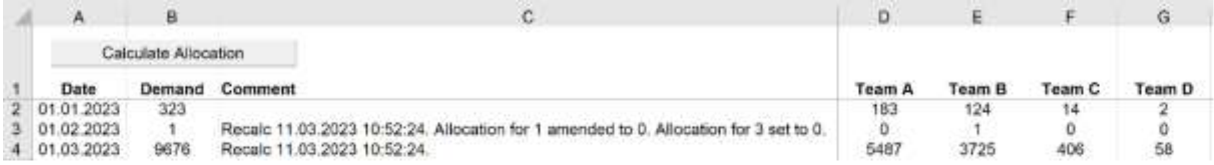
Example

On 1-Jan-2023 these teams exist (sheet ‘Teams’, VBA name ‘wsT’):



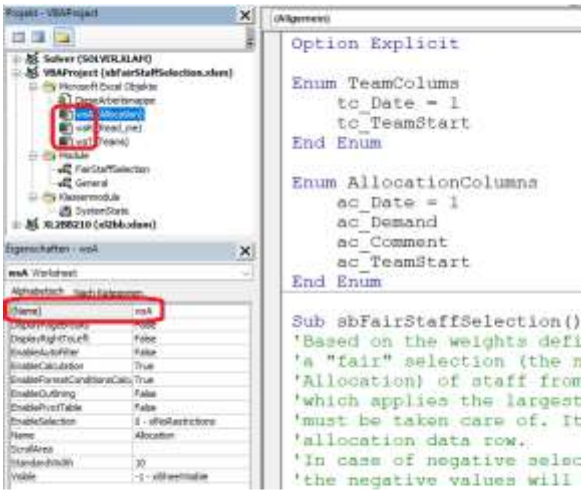
	A	B	C	D	E
1	Date	Team A	Team B	Team C	Team D
2	01.01.2023	5670	3850	420	60

Over the following three months these staff numbers are required for special tasks and are selected (sheet ‘Allocation’, VBA name ‘wsA’):



	A	B	C	D	E	F	G
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.



(Name)	wsA
DisplayPageBreaks	False
DisplayRightToLeft	False
EnableAutoFilter	False
EnableCalculation	True
EnableFormatConditionsCalc	True
EnableOutlining	False
EnablePivotTable	False
EnableSelection	0 - xlNoRestrictions
Name	Allocation
ScrollArea	
StandardWidth	10
Visible	-1 - xlSheetVisible

```
Option Explicit

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 defi
'a "fair" selection (the n
'Allocation) of staff from
'which applies the largest
'must be taken care of. It
'allocation data row.
'In case of negative selec
'the negative values will
```

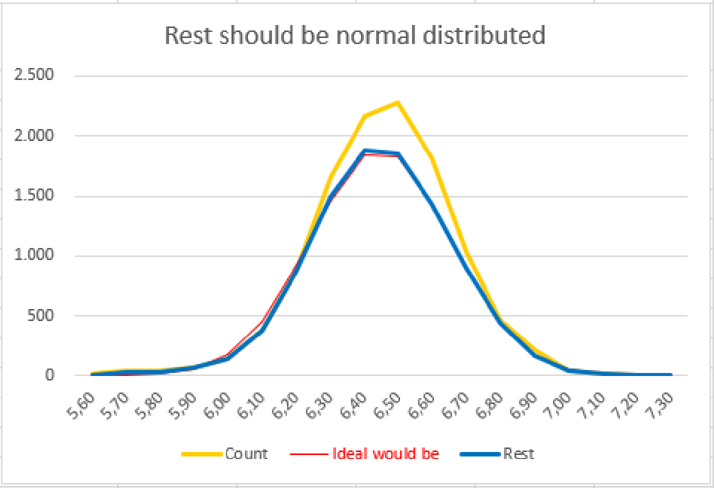
Note: The VBA name of a worksheet can be referred to directly from VBA. It might differ from the sheet name the Excel user sees. Unfortunately you can only manually change it, not via VBA.

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. eFehler! Textmarke nicht definiert.. 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.
'Source (EN): https://www.sulprobil.de/sbfairstaffselection_en
'Source (DE): https://www.bplumhoff.com/sbfairstaffselection_de
'(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
```

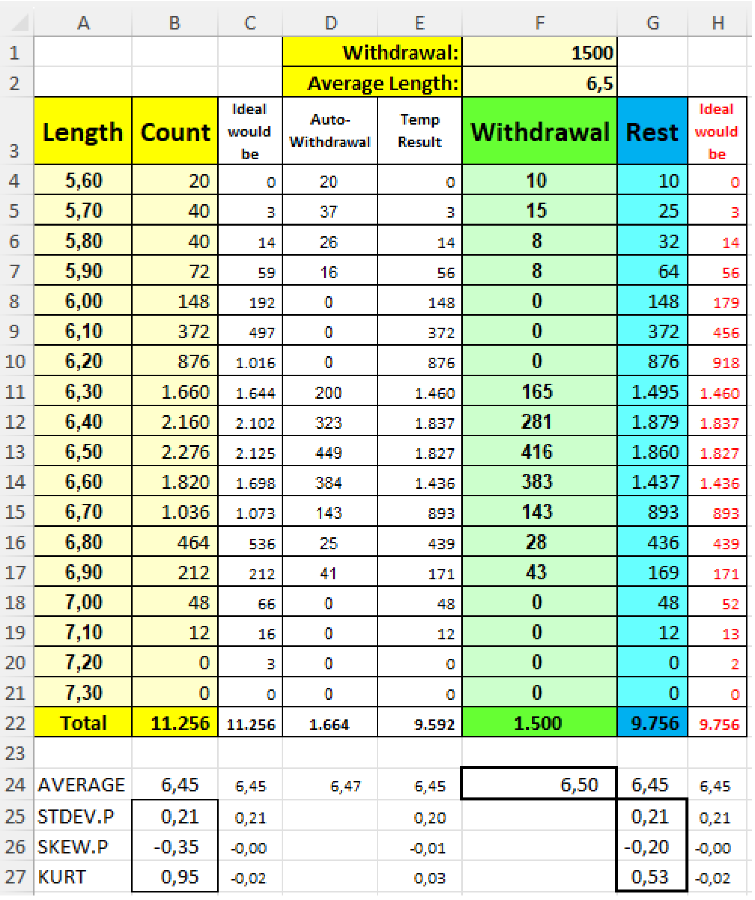
Distribute a Sample Normally

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:

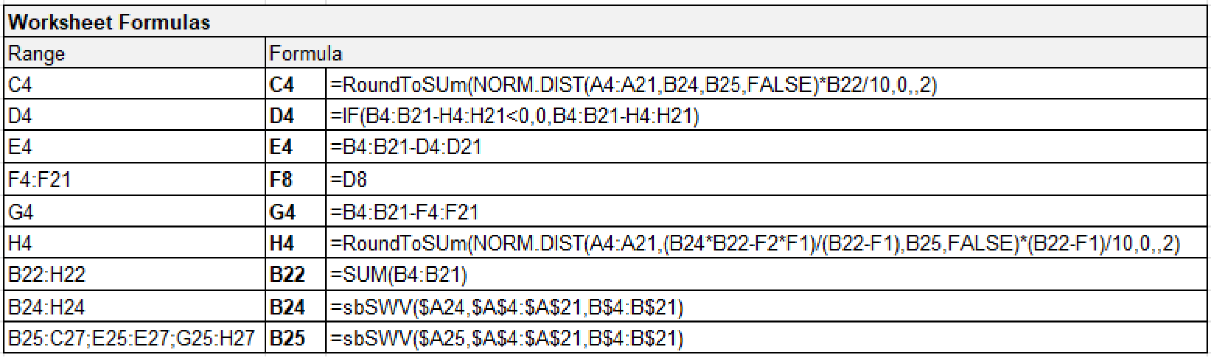


How can you achieve this?

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
'Source (EN): http://www.sulprobil.de/sbswv_en/
'Source (DE): http://www.berndplumhoff.de/sbswv_de/
'(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
```

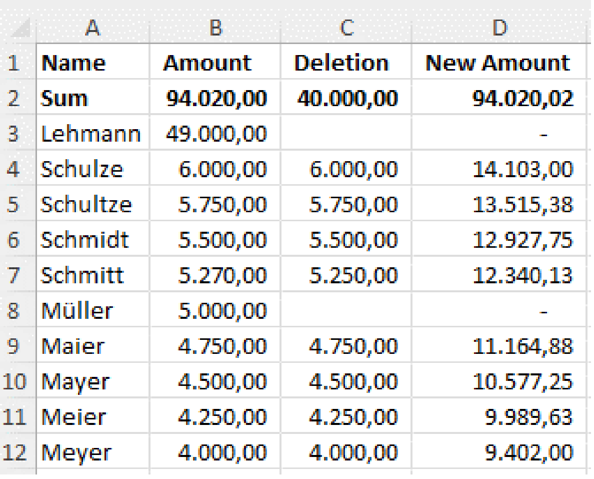
Distribution of Budgets Among Remaining Staff

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?

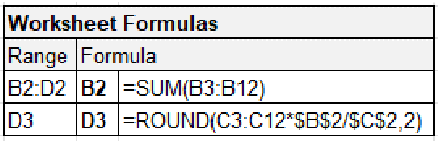
A Simple Approach

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



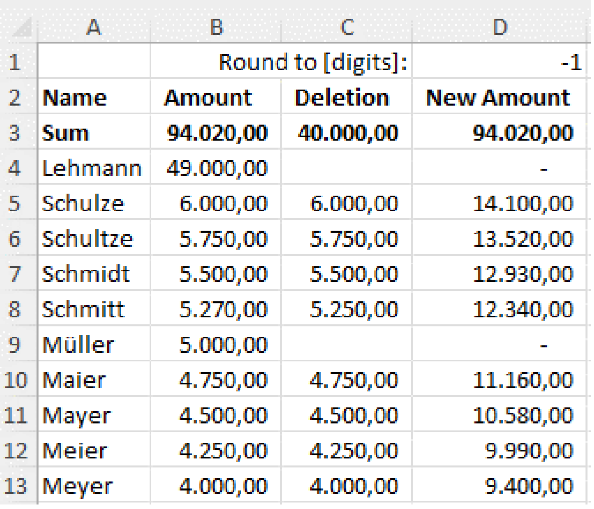
Worksheet Formulas

Range	Formula	
B2:D2	**B2**	=SUM(B3:B12)
D3	**D3**	=ROUND(C3:C12*$B$2/$C$2,2)

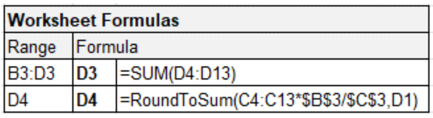
A Correct Calculation

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



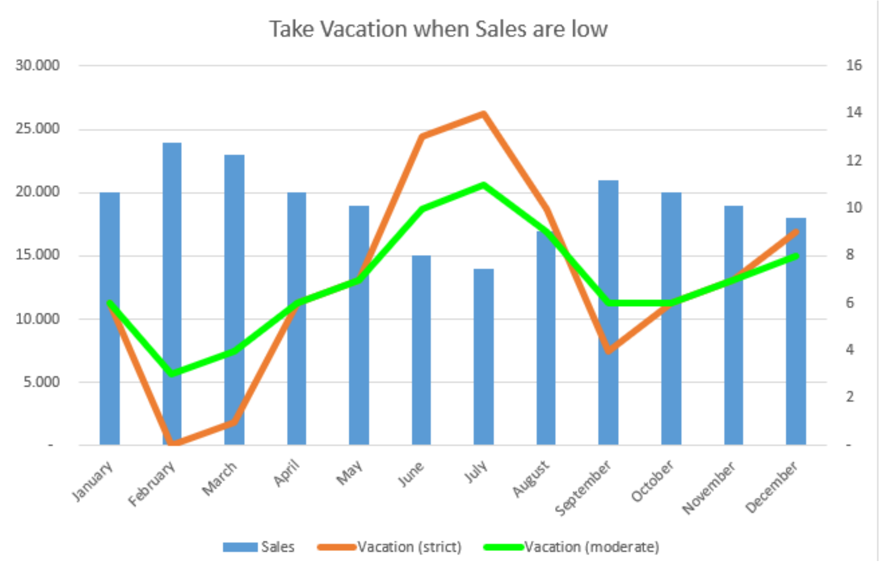
Worksheet Formulas

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

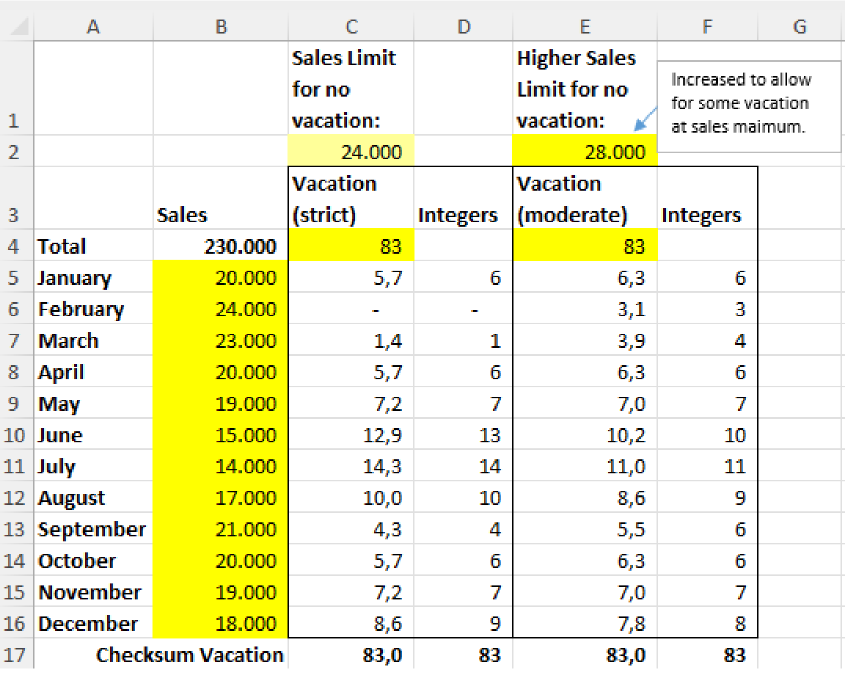
If your business fluctuates strongly seasonally, you can plan the vacation of your staff accordingly and consider hiring seasonal staff:

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.

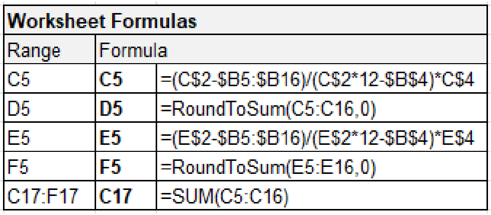


Simple Example

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:



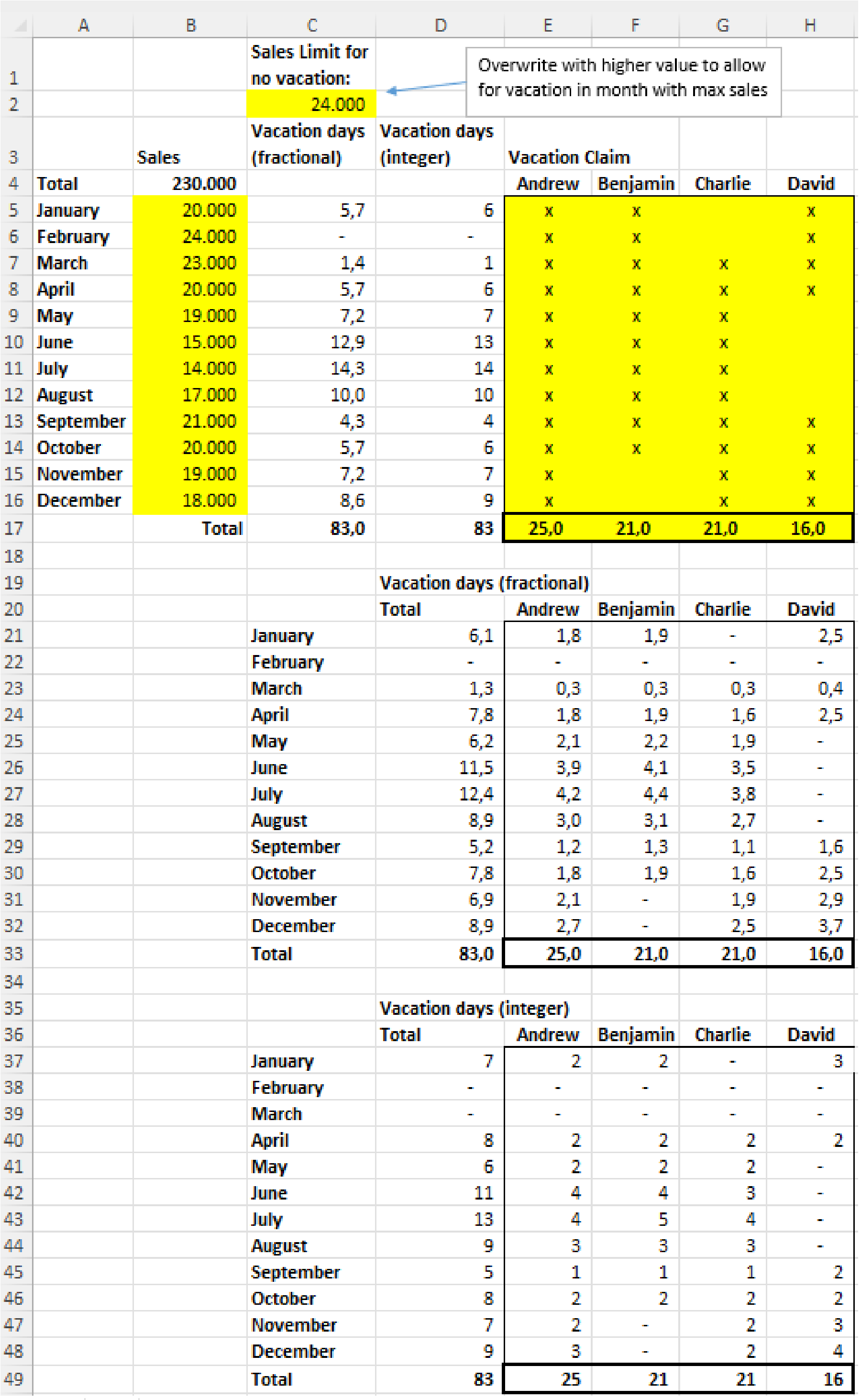
	A	B	C	D	E	F	G
1			Sales Limit for no vacation:		Higher Sales Limit for no vacation:	Increased to allow for some vacation at sales maimum.	
2			24.000		28.000		
3		Sales	Vacation (strict)	Integers	Vacation (moderate)	Integers	
4	Total	230.000	83		83		
5	January	20.000	5,7	6	6,3	6	
6	February	24.000	-	-	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

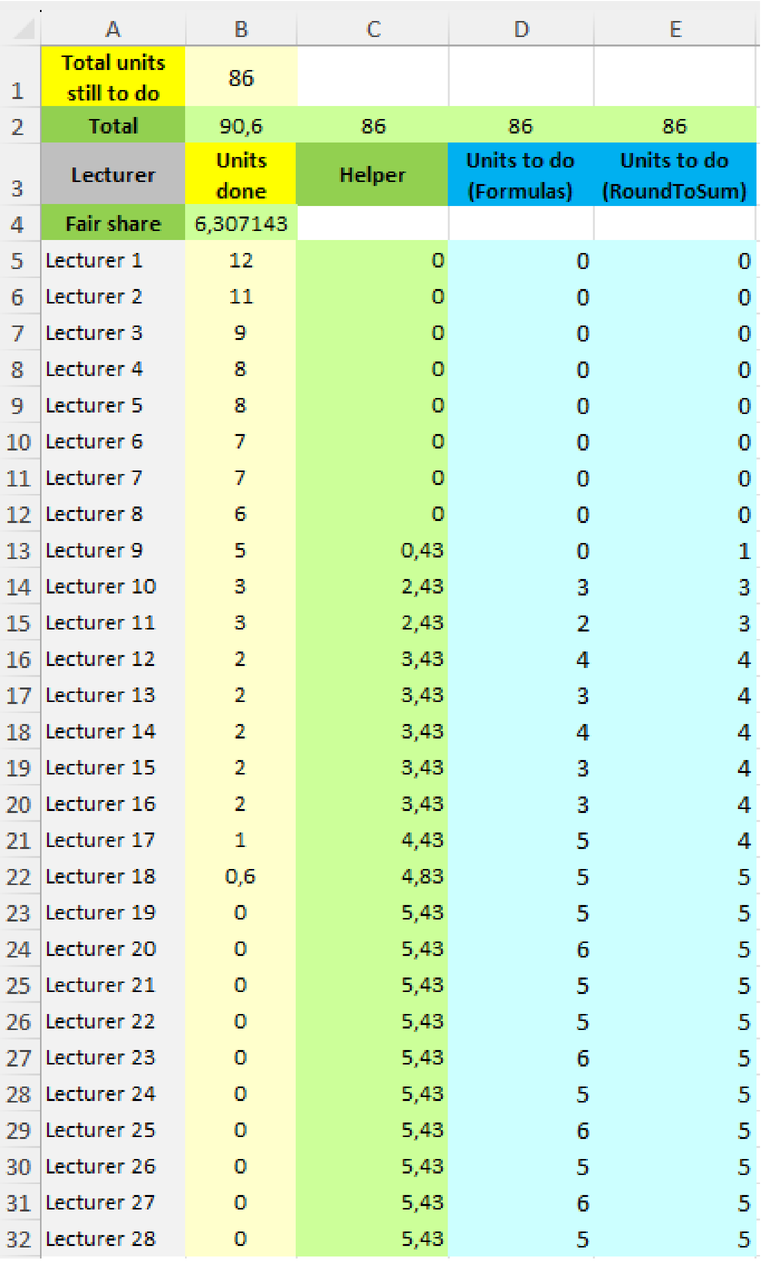
How can you fairly assign work units to your staff while considering the number of units they have already delivered?

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.

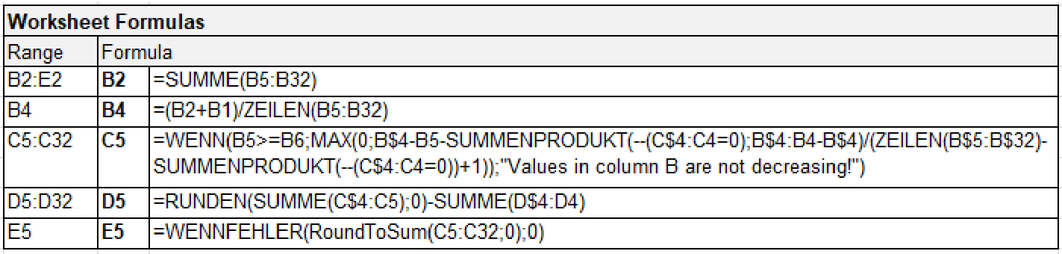
In this example *90.6* units have already been delivered, but *86* more units are to be assigned to *28* lecturers. A fair share for each lecturer would be *(90.6 + 86) / 28 = 6.3*, but *7* lecturers have already delivered more than that.

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



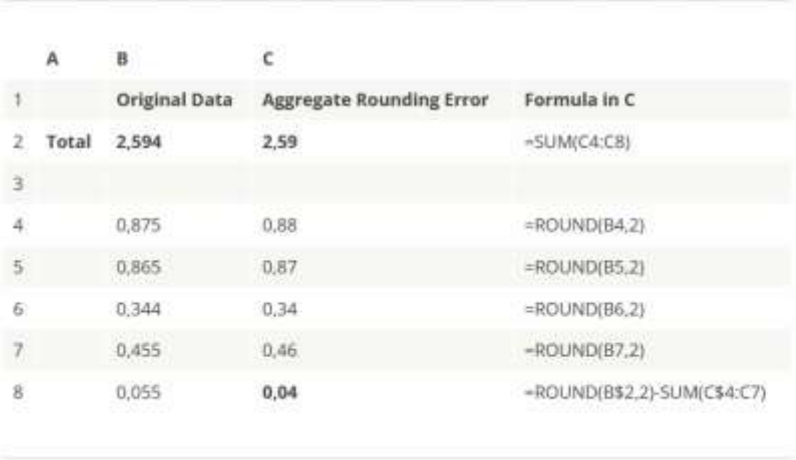
Worksheet Formulas		
Range	Formula	
B2:E2	**B2**	=SUMME(B5:B32)
B4	**B4**	=(B2+B1)/ZEILEN(B5:B32)
C5:C32	**C5**	=WENN(B5>=B6;MAX(0;B$4-B5-SUMMENPRODUKT(--(C$4:C4=0);B$4:B4-B$4)/(ZEILEN(B$5:B$32)-SUMMENPRODUKT(--(C$4:C4=0))+1));"Values in column B are not decreasing!")
D5:D32	**D5**	=RUNDEN(SUMME(C$4:C5);0)-SUMME(D$4:D4)
E5	**E5**	=WENNFEHLER(RoundToSum(C5:C32;0);0)

RoundToSum Versus Other Methods

RoundToSum Versus Other “Simple” Methods

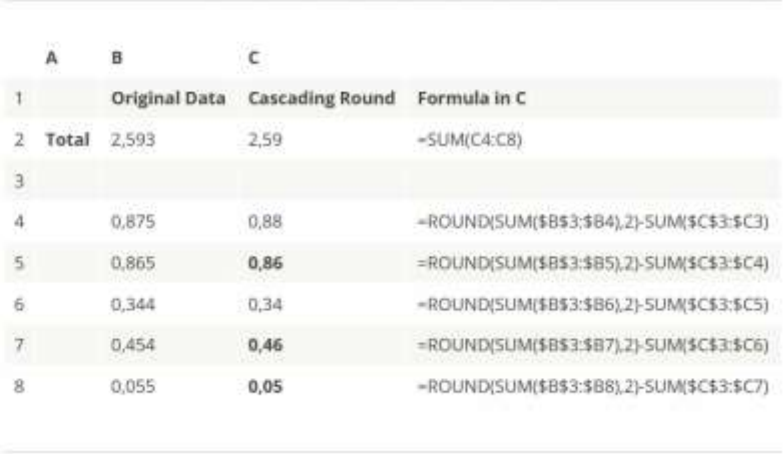
There are several different naïve approaches circulating around which try to round values preserving their rounded sum:

- (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:

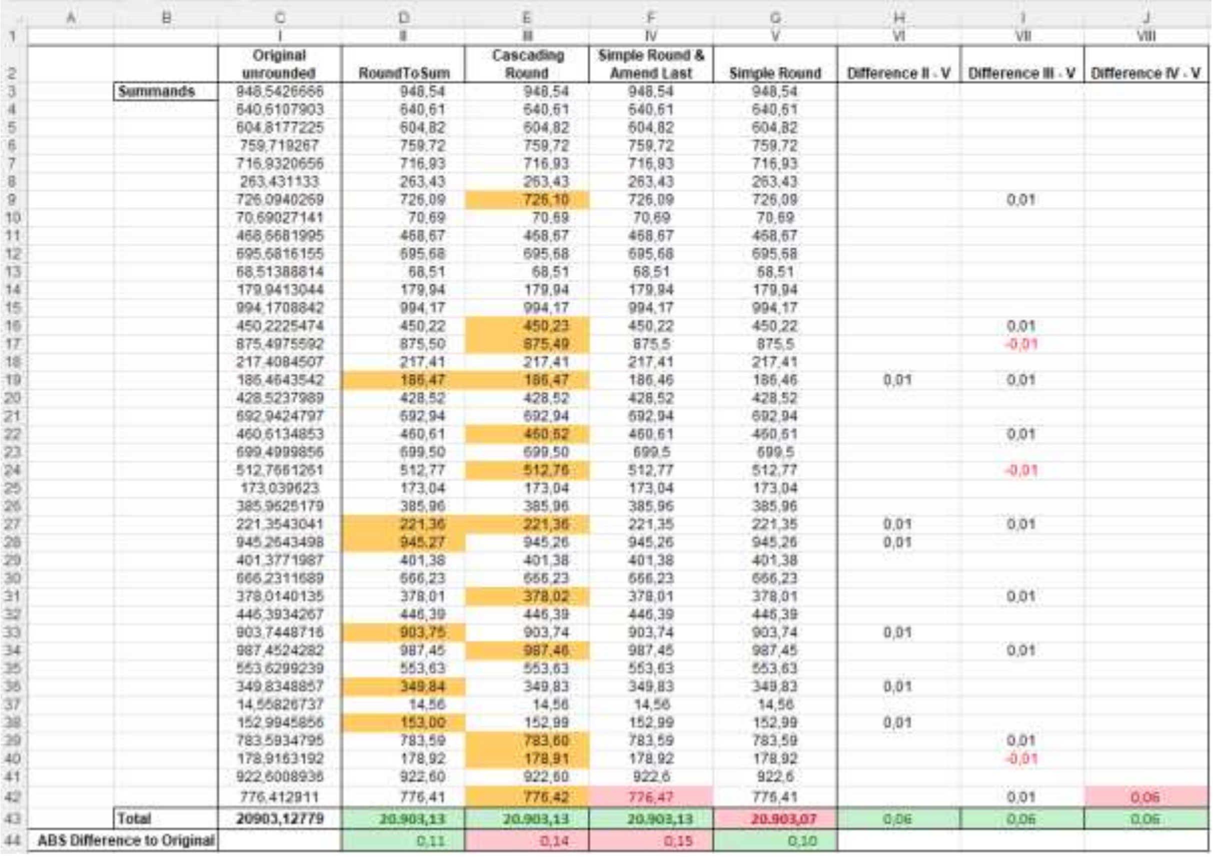


	A	B	C	
1		**Original Data**	**Cascading Round**	**Formula in C**
2	**Total**	2,593	2,59	=SUM(C4:C8)
3				
4		0,875	0,88	=ROUND(SUM($B$3:$B4),2)-SUM($C$3:$C3)
5		0,865	**0,86**	=ROUND(SUM($B$3:$B5),2)-SUM($C$3:$C4)
6		0,344	0,34	=ROUND(SUM($B$3:$B6),2)-SUM($C$3:$C5)
7		0,454	**0,46**	=ROUND(SUM($B$3:$B7),2)-SUM($C$3:$C6)
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

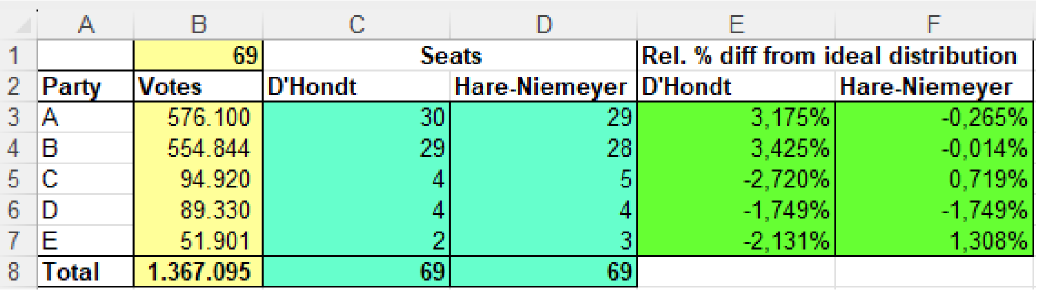
Use RoundToSum. It will apply the least number of changes and it will result in the correct sum with the smallest absolute (or relative) error.

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.

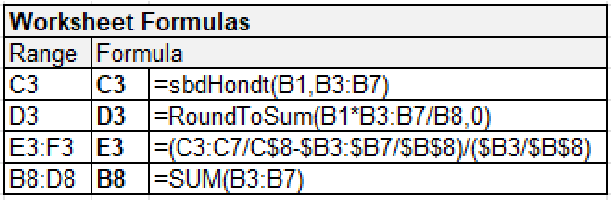
But worst of all is the approach of aggregating all rounding differences in the last summand. Just imagine 1,000 people, each having 49 Cents, adding up to $490, which you should distribute fairly, but rounded to a whole Dollar. In this case you would end up with $490 at the last person, while *RoundToSum* 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.
'Source (EN): http://www.sulprobil.de/sbdhondt_en/
'Source (DE): http://www.berndplumhoff.de/sbdhondt_de/
'(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

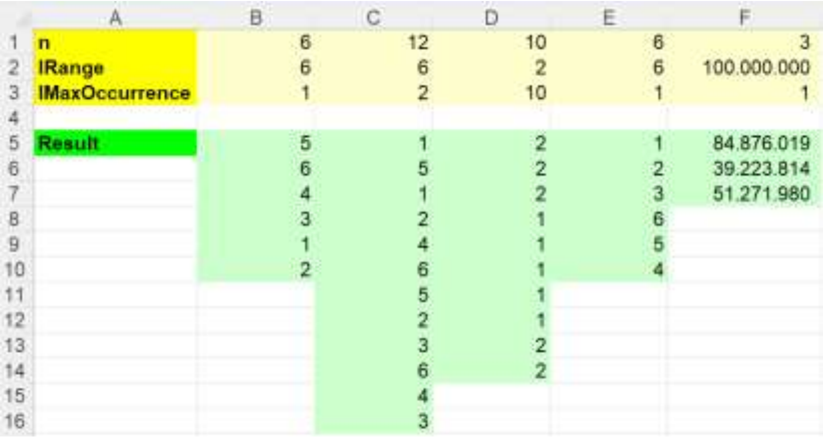
Natural Random Numbers – *UniqRandInt*

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.

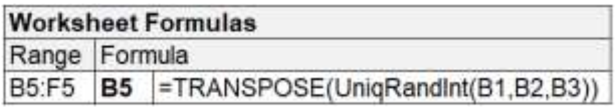
Sometimes, you need random integers that do not repeat, or only repeat a limited number of times. For example, if you need 20 positive random integers between 1 and 100, you would enter: *=UniqRandInt(20,100)*. If the range should be between 100 and 199, you would use *=UniqRandInt(20,100)+99*. In general:

=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	lRange	6	6	2	6	100.000.000
3	lMaxOccurrence	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			



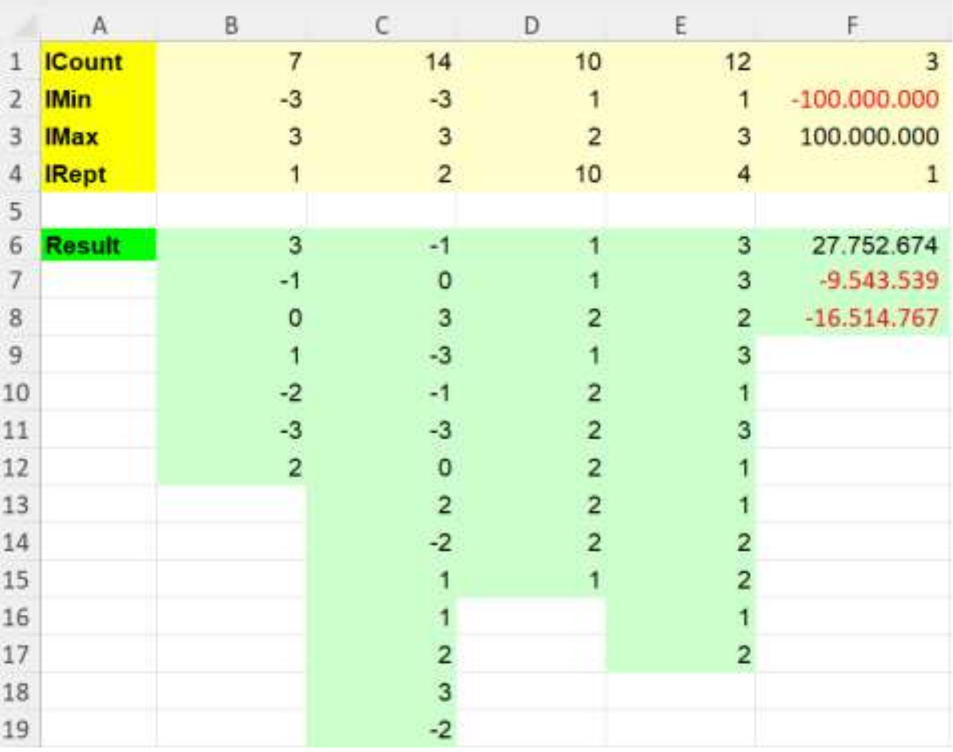
Worksheet Formulas		
Range	Formula	
B5:F5	B5	=TRANSPOSE(UniqRandInt(B1,B2,B3))

UniqRandInt Program Code

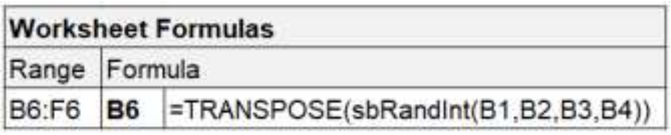
```
'If lRange >> n then set LATE_INITIALISATION to true. For example,
'if lRange=1,000,000 and if 1,000 cells are selected (n=1000).
#Const LATE_INITIALISATION = True
'If random integers may occur more than once, allow repetitions
#Const ALLOW_REPETITION = True

#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	lCount	7	14	10	12	3
2	lMin	-3	-3	1	1	-100.000.000
3	lMax	3	3	2	3	100.000.000
4	lRept	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.

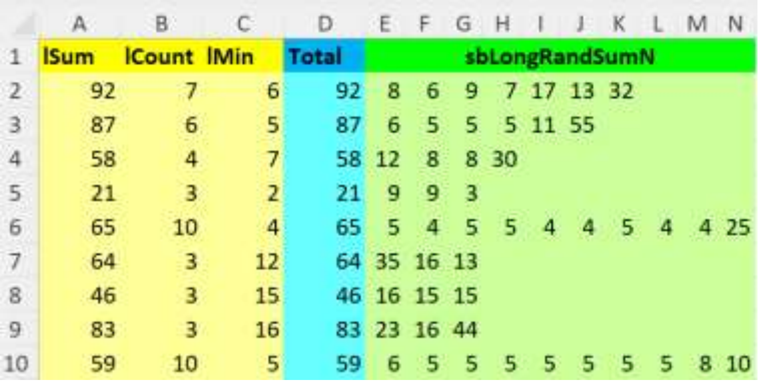
sbRandInt – Program Code

```
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
```

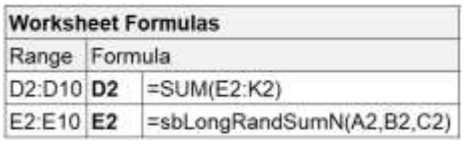
Random Numbers with a Specified Sum

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



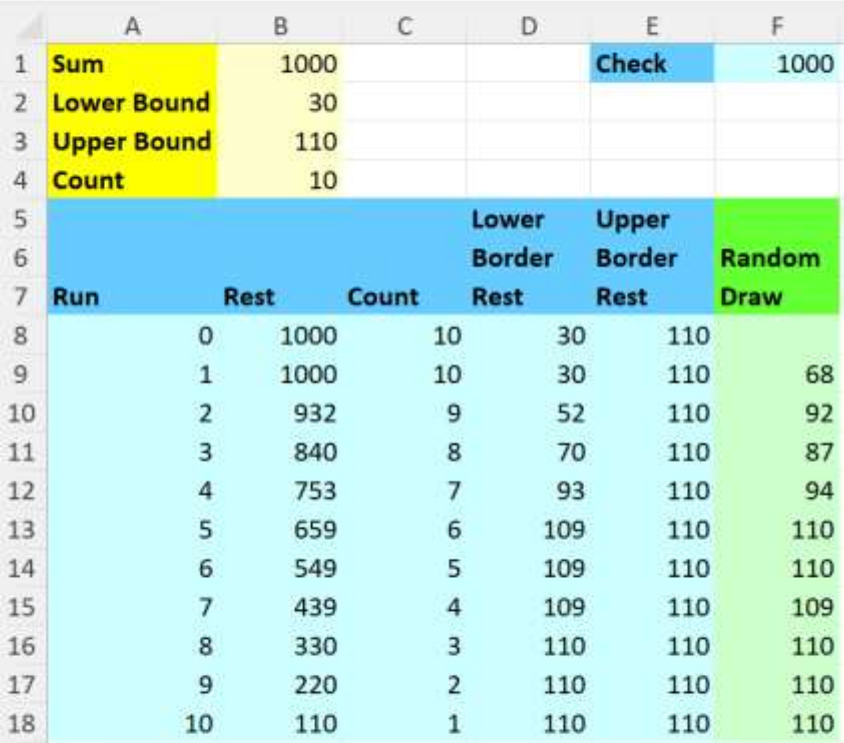
Worksheet Formulas		
Range	Formula	
D2:D10	D2	=SUM(E2:K2)
E2:E10	E2	=sbLongRandSumN(A2,B2,C2)

sbLongRandSumN Program Code

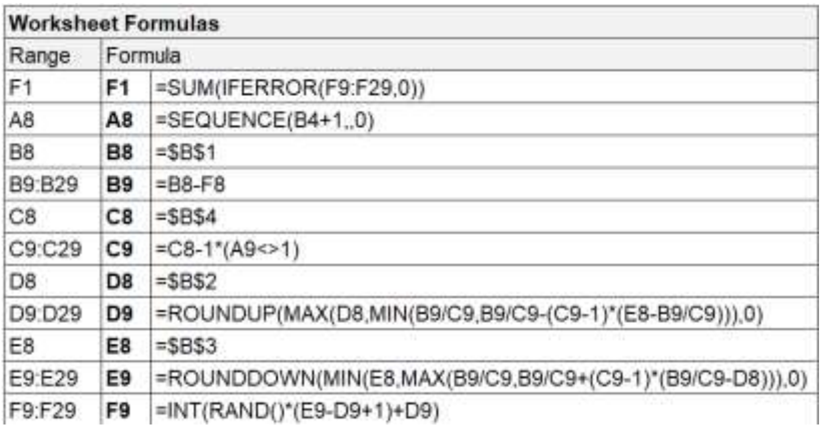
```
Function sbLongRandSumN(lSum As Long, _
    ByVal lCount As Long, _
    Optional ByVal lMin As Long = 0) As Variant
'Generates lCount random integers greater equal lMin
'which sum up to lSum.
'(C) (P) by Bernd Plumhoff 26-Apr-2013 PB V0.1
Dim i As Long
Dim lSumRest As Long

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

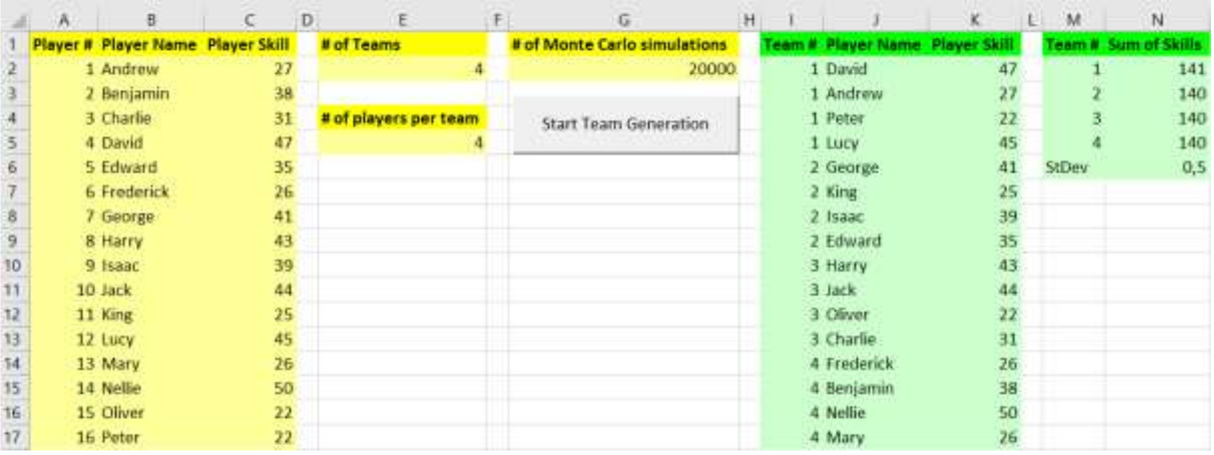
```
Function sbRandIntFixSum(lSum As Long, lMin As Long, _
    lMax As Long, Optional lCount As Long = 0, _
    Optional bUseRandTriang As Boolean = True, _
    Optional bVolatile As Boolean = False) As Variant
'Returns lCount (or selected cell count in case a range is select when
'called as a matrix 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

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*:



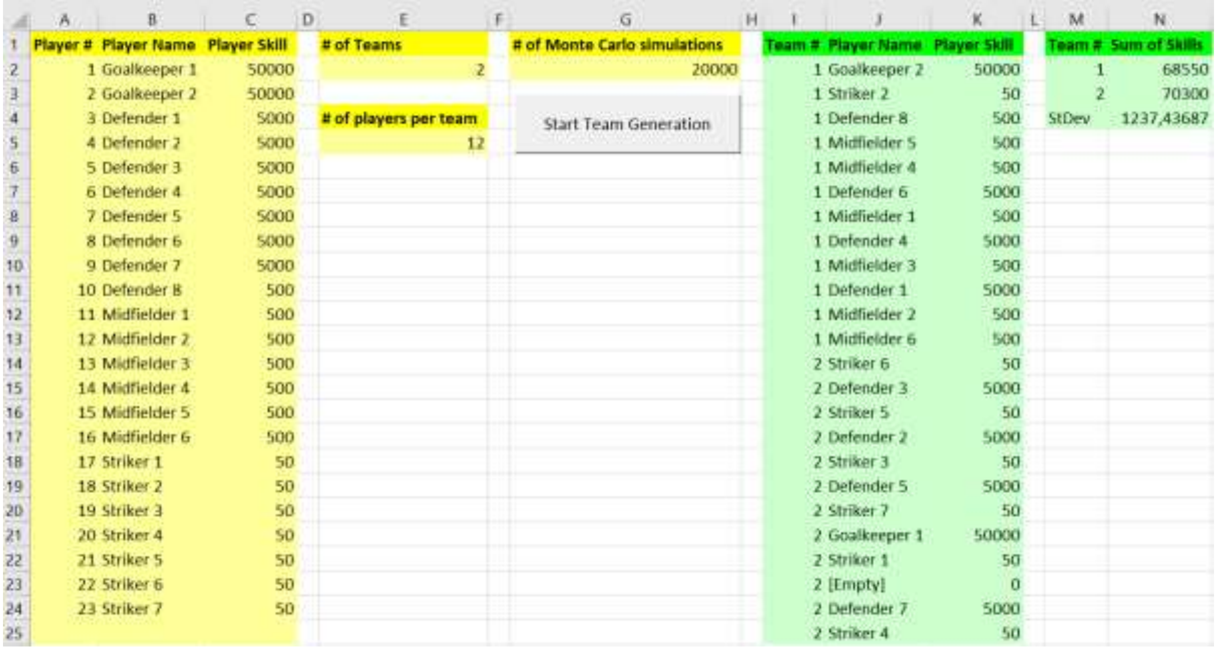
	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Player #	Player Name	Player Skill		# of Teams		# of Monte Carlo simulations		Team #	Player Name	Player Skill		Team #	Sum of Skills
2	1	Andrew	27		4		20000		1	David	47		1	141
3	2	Benjamin	38						1	Andrew	27		2	140
4	3	Charlie	31		# of players per team		Start Team Generation		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:

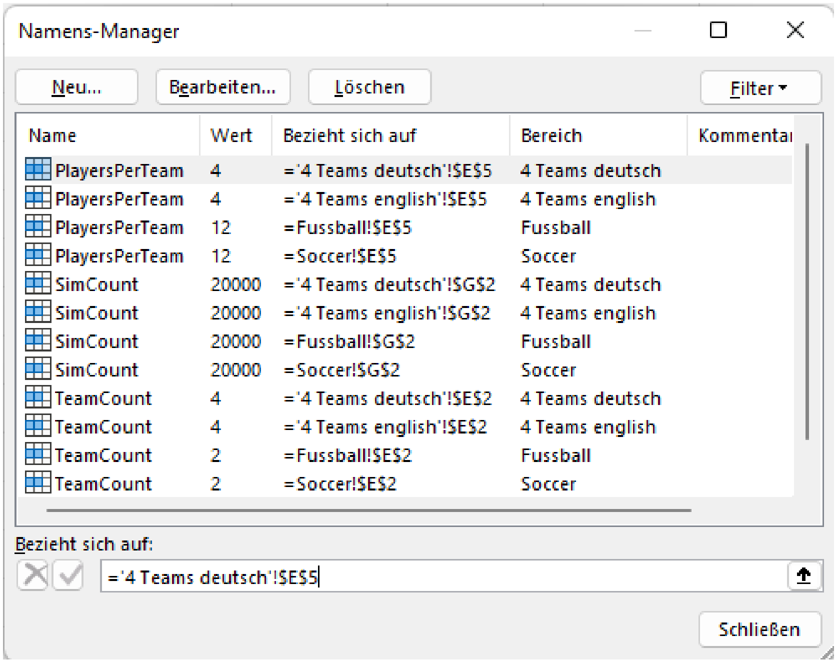


	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Player #	Player Name	Player Skill		# of Teams		# of Monte Carlo simulations		Team #	Player Name	Player Skill		Team #	Sum of Skills
2	1	Goalkeeper 1	50000		2		20000		1	Goalkeeper 2	50000		1	68550
3	2	Goalkeeper 2	50000						1	Striker 2	50		2	70300
4	3	Defender 1	5000		# of players per team		Start Team Generation		1	Defender 8	500		StDev	1237,43687
5	4	Defender 2	5000		12				1	Midfielder 5	500			
6	5	Defender 3	5000						1	Midfielder 4	500			
7	6	Defender 4	5000						1	Defender 6	5000			
8	7	Defender 5	5000						1	Midfielder 1	500			
9	8	Defender 6	5000						1	Defender 4	5000			
10	9	Defender 7	5000						1	Midfielder 3	500			
11	10	Defender 8	500						1	Defender 1	5000			
12	11	Midfielder 1	500						1	Midfielder 2	500			
13	12	Midfielder 2	500						1	Midfielder 6	500			
14	13	Midfielder 3	500						2	Striker 6	50			
15	14	Midfielder 4	500						2	Defender 3	5000			
16	15	Midfielder 5	500						2	Striker 5	50			
17	16	Midfielder 6	500						2	Defender 2	5000			
18	17	Striker 1	50						2	Striker 3	50			
19	18	Striker 2	50						2	Defender 5	5000			
20	19	Striker 3	50						2	Striker 7	50			
21	20	Striker 4	50						2	Goalkeeper 1	50000			
22	21	Striker 5	50						2	Striker 1	50			
23	22	Striker 6	50						2	[Empty]	0			
24	23	Striker 7	50						2	Defender 7	5000			
25									2	Striker 4	50			

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:



Name	Wert	Bezieht sich auf	Bereich	Kommentar
PlayersPerTeam	4	='4 Teams deutsch'!$E$5	4 Teams deutsch	
PlayersPerTeam	4	='4 Teams english'!$E$5	4 Teams english	
PlayersPerTeam	12	=Fussball!$E$5	Fussball	
PlayersPerTeam	12	=Soccer!$E$5	Soccer	
SimCount	20000	='4 Teams deutsch'!$G$2	4 Teams deutsch	
SimCount	20000	='4 Teams english'!$G$2	4 Teams english	
SimCount	20000	=Fussball!$G$2	Fussball	
SimCount	20000	=Soccer!$G$2	Soccer	
TeamCount	4	='4 Teams deutsch'!$E$2	4 Teams deutsch	
TeamCount	4	='4 Teams english'!$E$2	4 Teams english	
TeamCount	2	=Fussball!$E$2	Fussball	
TeamCount	2	=Soccer!$E$2	Soccer	

Bezieht sich auf:
='4 Teams deutsch'!$E$5

```
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
```

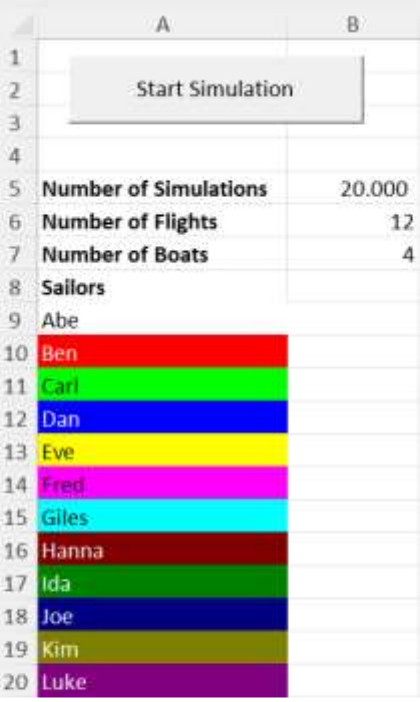
Monte Carlo Simulation for a Regatta Flight Plan – *sbRegattaFlightPlan*

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

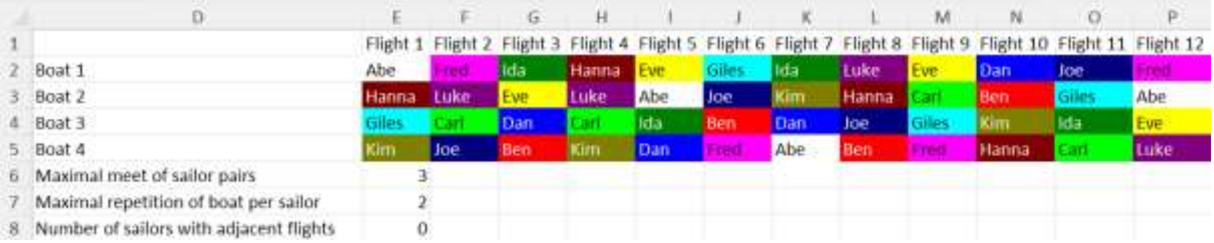
then hopefully this program, which uses *UniqRandInt* the class *SystemState*, will help you.

Input:



	A	B
1		
2		
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

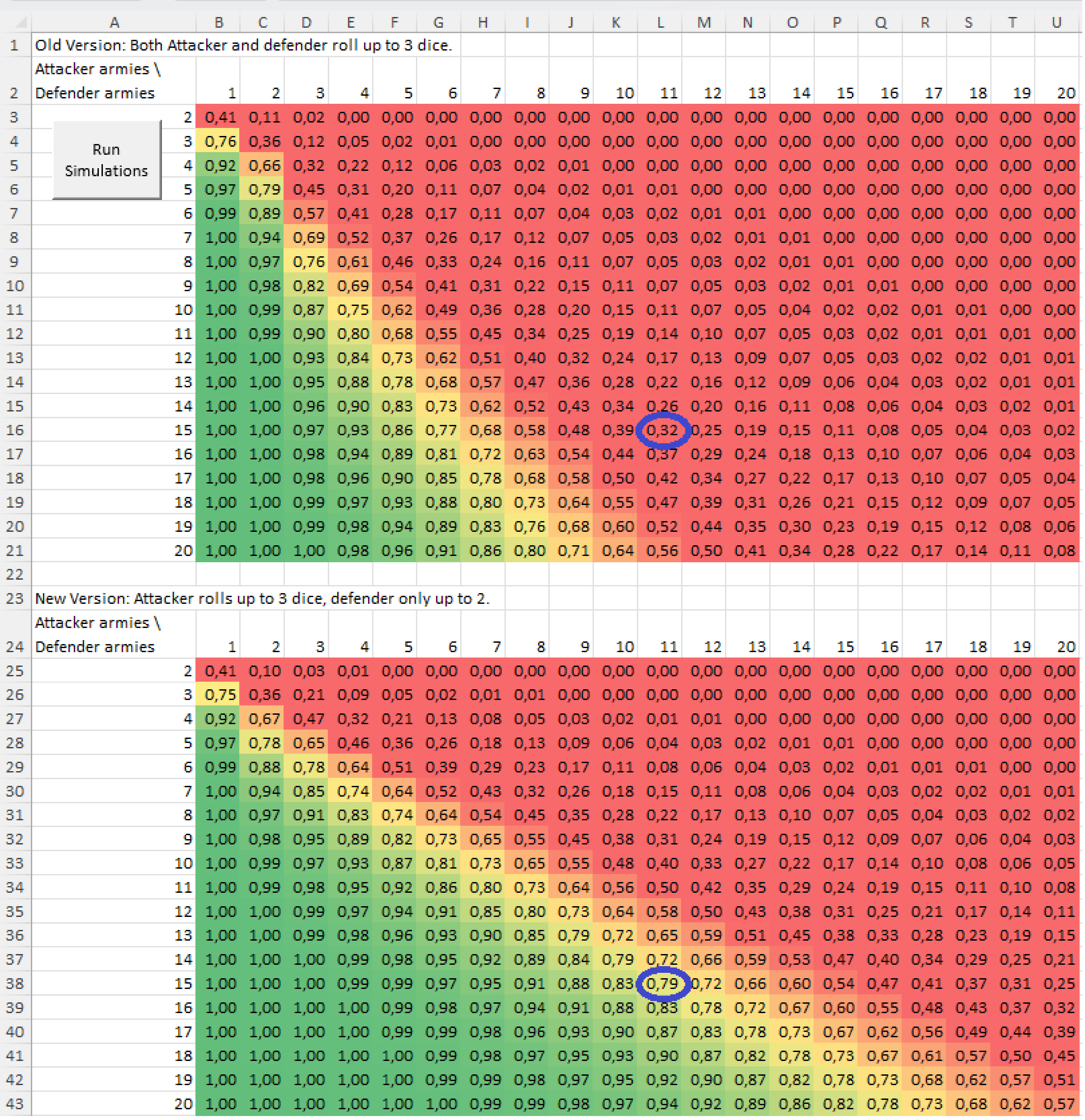
This program requires *UniqRandInt* and uses the class *SystemState*.

```
#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
```

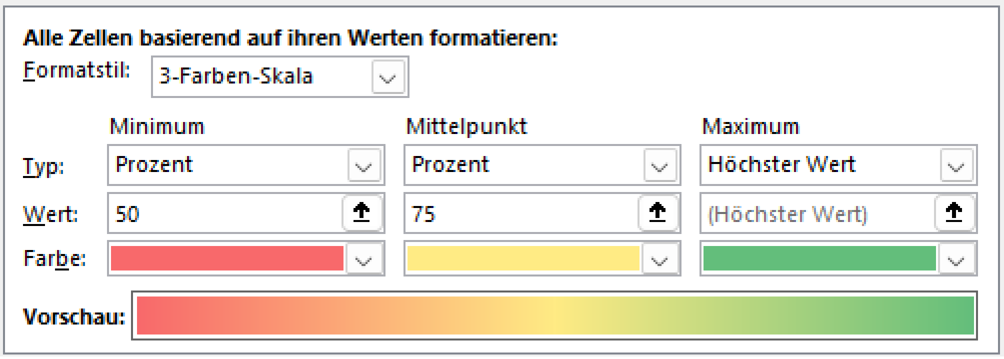
Chances at Board Game Risk

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	Run Simulations 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

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:



Alle Zellen basierend auf ihren Werten formatieren:

Formatstil: 3-Farben-Skala

	Minimum	Mittelpunkt	Maximum
Typ:	Prozent	Prozent	Höchster Wert
Wert:	50	75	(Höchster Wert)
Farbe:	🟥	🟨	🟩

Vorschau:

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

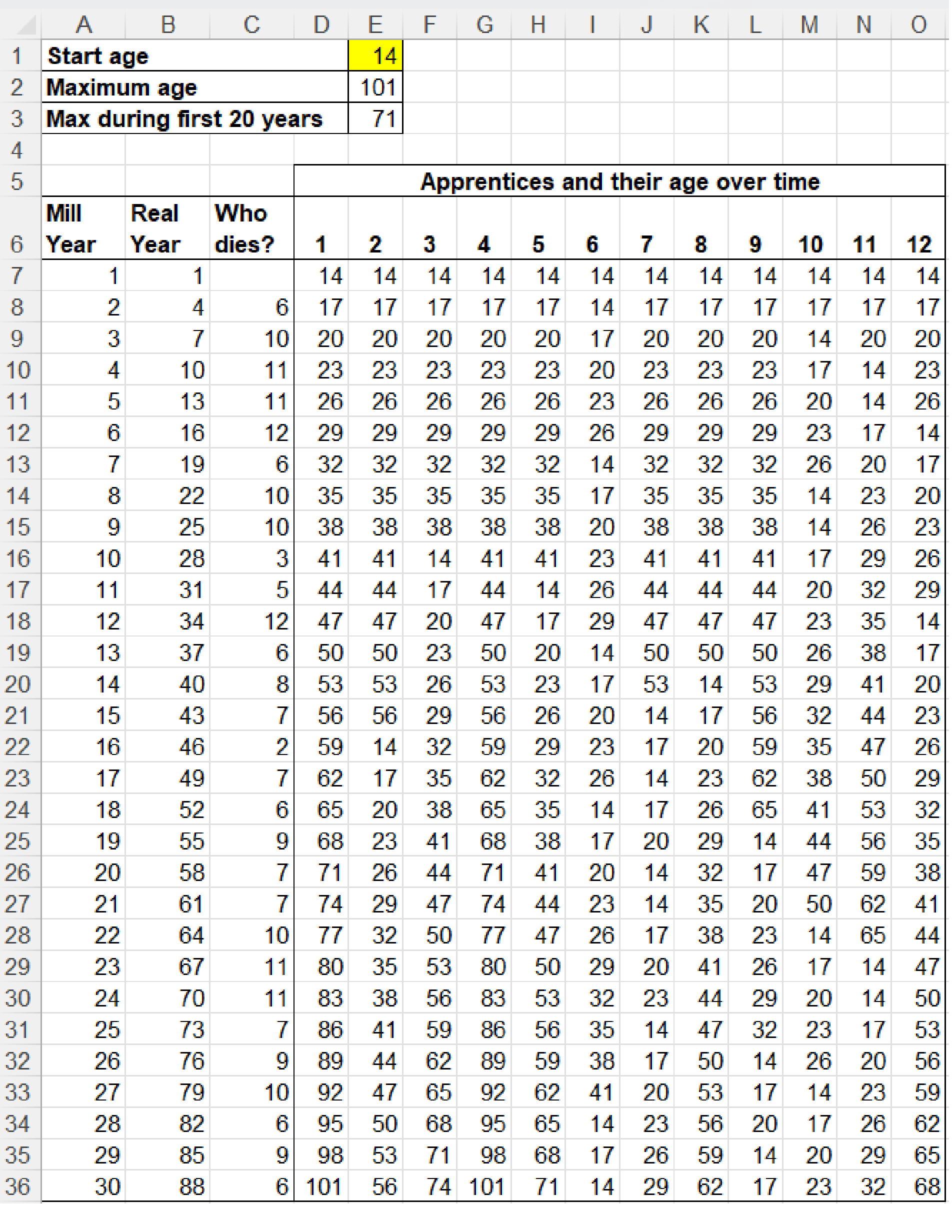
```
Const GCMonteCarloRuns = 10000

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
```

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:



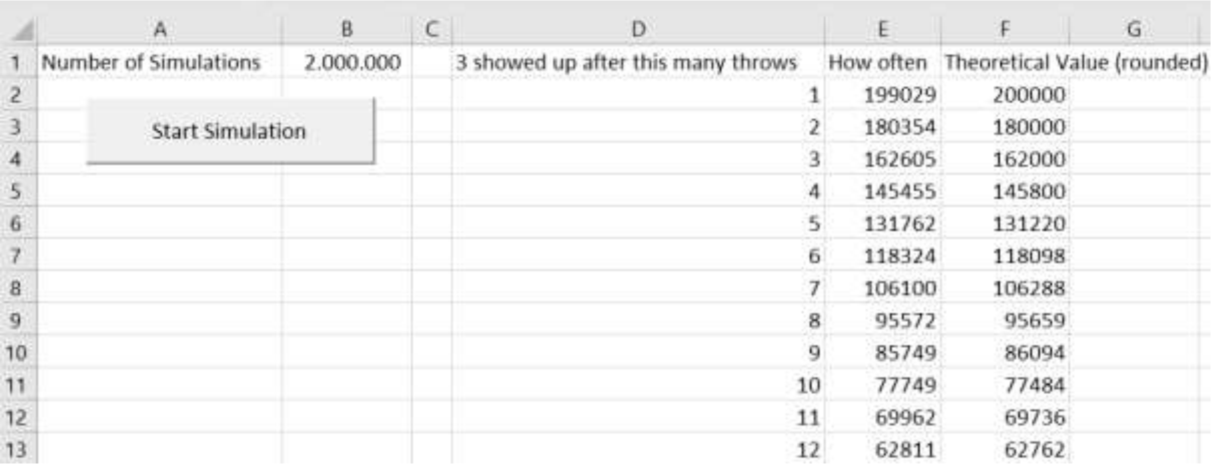
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	**Start age**				14										
2	**Maximum age**				101										
3	**Max during first 20 years**				71										
4															
5				**Apprentices and their age over time**											
6	**Mill Year**	**Real Year**	**Who dies?**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**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)

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				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

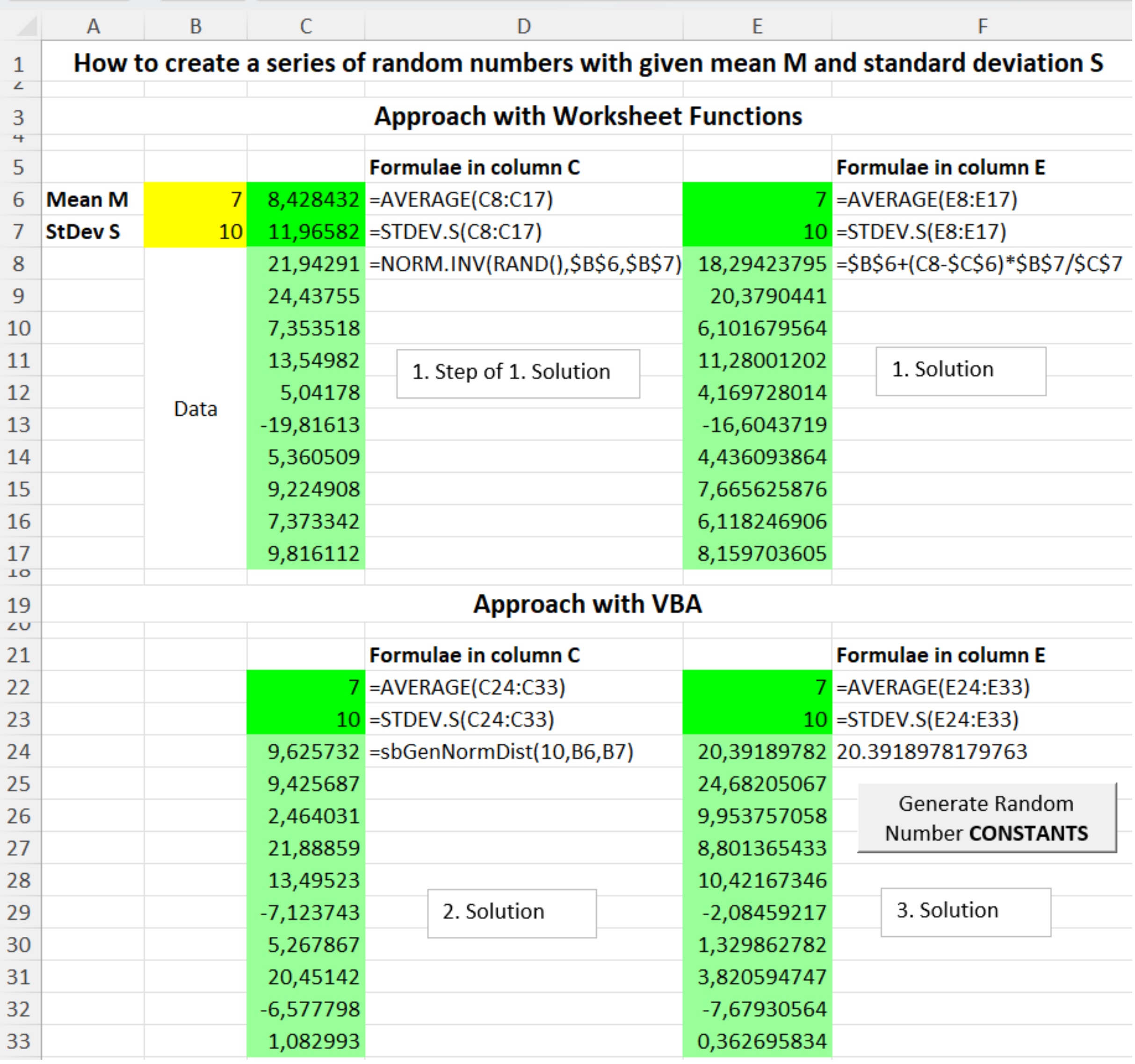
```
Sub Simulate()
'Creates a simple Monte Carlo simulation by counting how long
'it takes to throw a 3 with a die with 10 surfaces (likelihood
'for each to show is 1/10).
'(C) (P) by Bernd Plumhoff  23-Nov-2022 PB V0.2
Dim i                    As Long
Dim lSimulations         As Long
Dim lTries               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**					
2						
3				**Approach with Worksheet Functions**		
4						
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		Data	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			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	
18						
19				**Approach with VBA**		
20						
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	Generate Random Number **CONSTANTS**
26			2,464031		9,953757058	
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
```

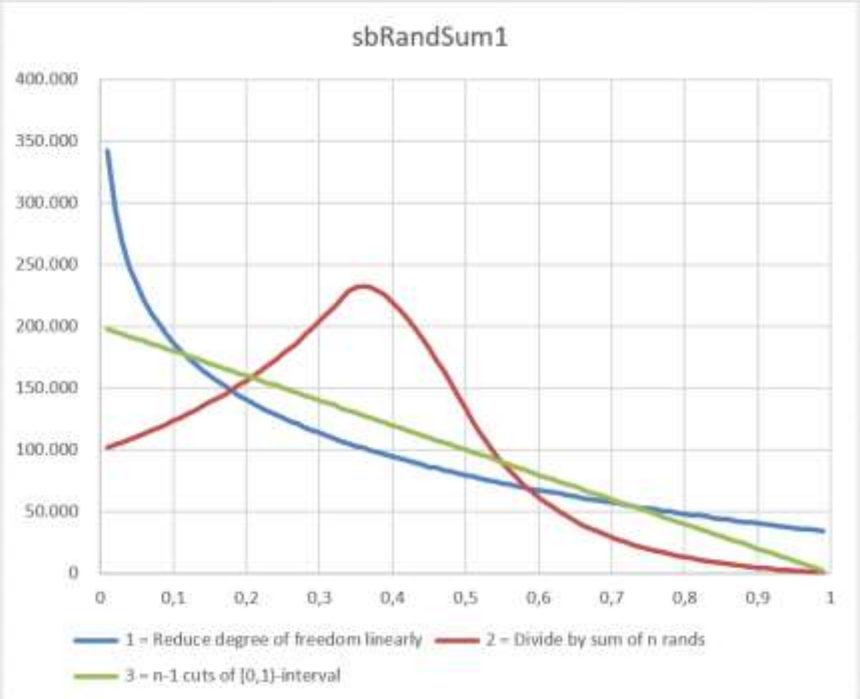
Generate Random Numbers with a Sum of 1 – *sbRandSum1*

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

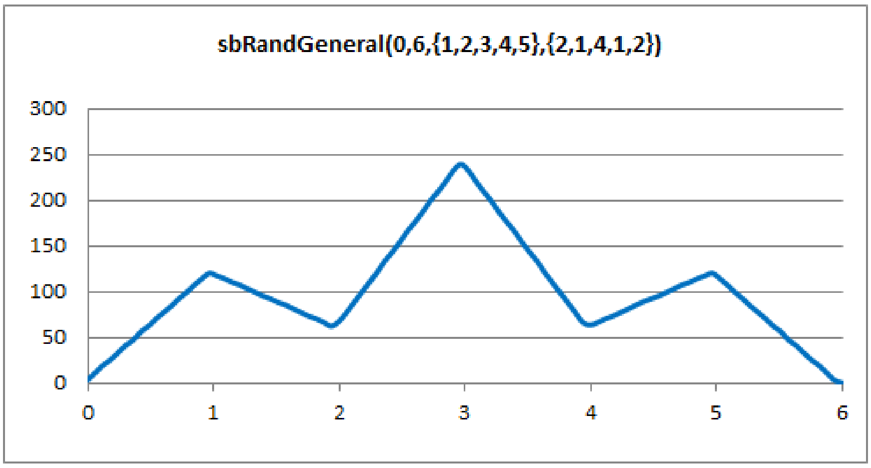
```
Function sbRandSum1(ByVal lDist As Long, Optional ByVal lCount As Long, _
    Optional bVolatile As Boolean = False) As Variant
'sbRandSum1 produces lCount (or the number of selected cells if
'called from a worksheet range) random numbers which sum up to 1.
'Possible values of lDist to specify desired 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

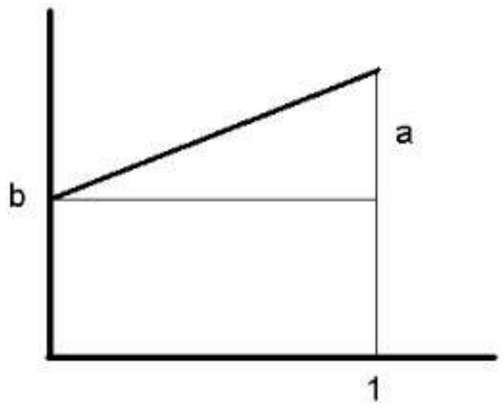
sbRandGeneral

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

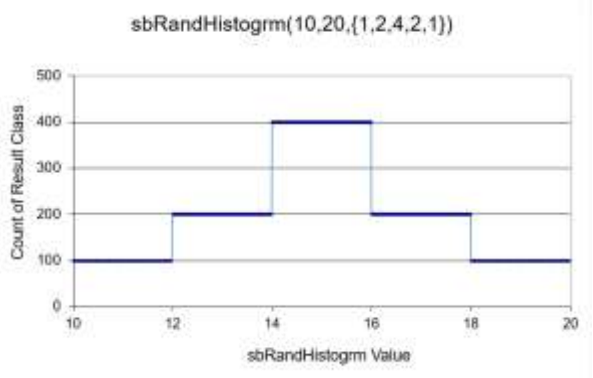
$$G(x) = -\frac{w_i(x_{i+1} - x_i)}{w_{i+1} - w_i} \pm \sqrt{\frac{2(x_{i+1} - x_i)}{w_{i+1} - w_i}x + \left(\frac{w_i(x_{i+1} - x_i)}{w_{i+1} - w_i}\right)^2}$$

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:



sbRandHistogram 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

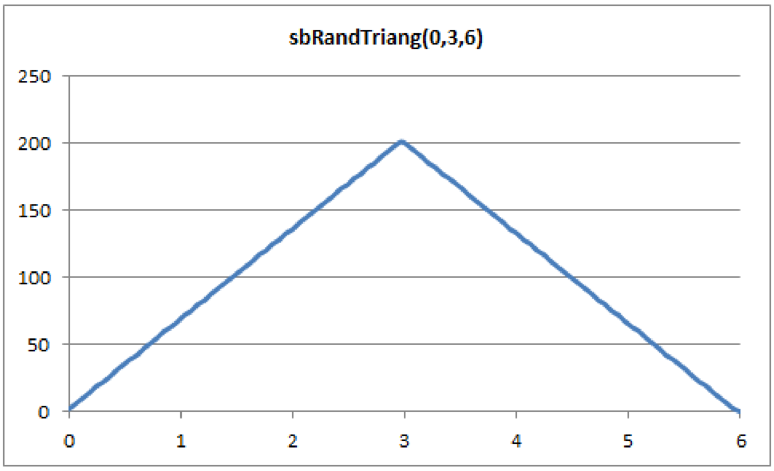
```
Function rww(ParamArray w() As Variant) As Double
'Produces random numbers with defined widths & weights
'Rww expects a vector of n random widths and weightings
'of type double and returns a random number of type double.
'This random number will lie in the given n-width-range of the
'(0,1)-interval 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:



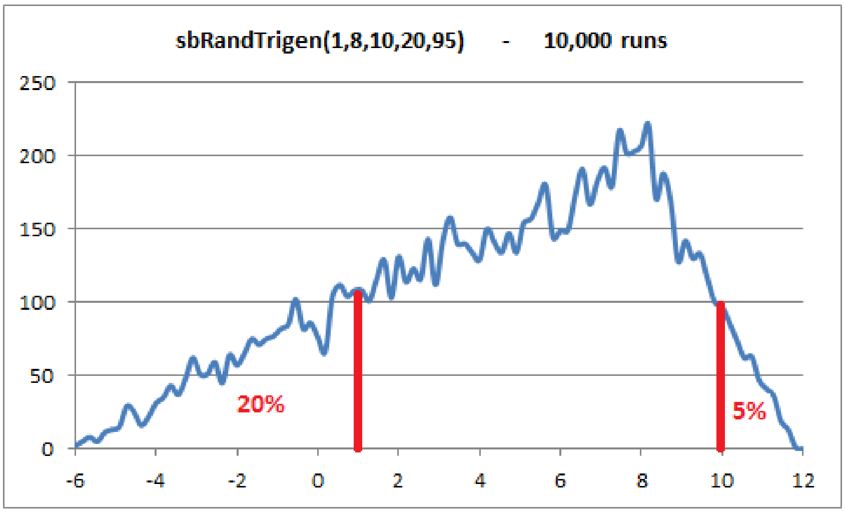
In the English-speaking world, this distribution is also referred to as the Distribution of Missing Data, because determining it requires only a minimal amount of information. This distribution is often used to simulate expert knowledge or when more accurate data collection is deemed too difficult or too expensive.

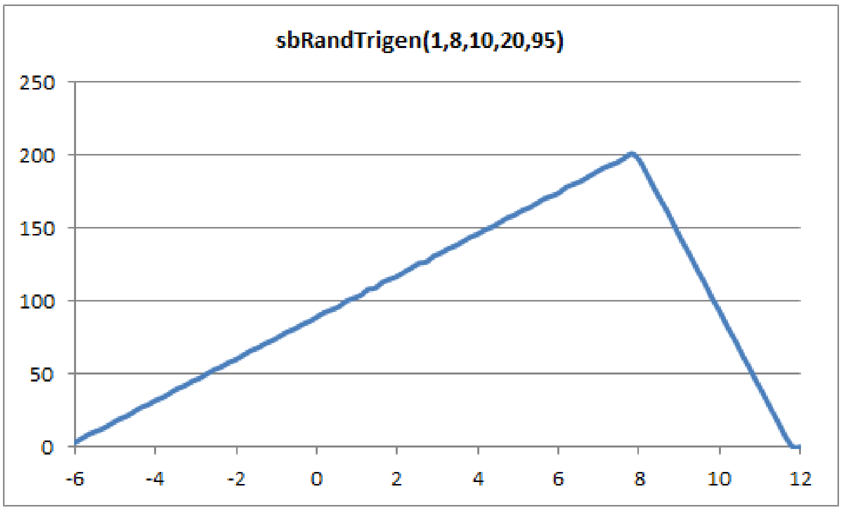
sbRandTriang Program Code

```
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
```

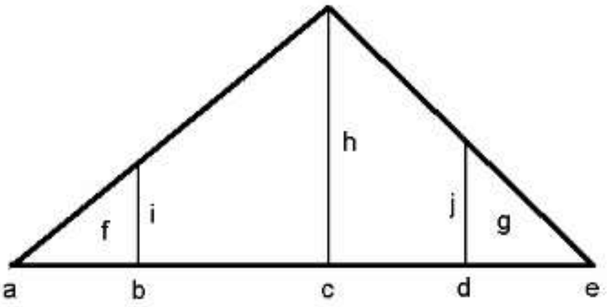
sbRandTrigen

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] $h(e-a)=2 \Leftrightarrow h=\dfrac{2}{e-a}$

[ii] $\dfrac{h}{c-a}=\dfrac{i}{b-a} \Leftrightarrow i=h\dfrac{b-a}{c-a}=\dfrac{2(b-a)}{(e-a)(c-a)}$

[iii] $\dfrac{h}{e-c}=\dfrac{j}{e-d} \Leftrightarrow j=h\dfrac{e-d}{e-c}=\dfrac{2(e-d)}{(e-a)(e-c)}$

[iv] $2g = j(e-d)=\dfrac{2(e-d)^2}{(e-a)(e-c)} \Rightarrow a=e-\dfrac{(e-d)^2}{g(e-c)}$

[v] $2f = i(b-a)=\dfrac{2(b-a)^2}{(e-a)(c-a)}$

Substituting a in [v] and putting everything on one side gives:

$$\left\{b-e+\frac{(e-d)^2}{g(e-c)}\right\}^2-f\left\{e-e+\frac{(e-d)^2}{g(e-c)}\right\}\left\{c-e+\frac{(e-d)^2}{g(e-c)}\right\}=0$$

$\Leftrightarrow$

$$\frac{((b-e)g(e-c)+(e-d)^2)^2}{(g(e-c))^2}-f\frac{(e-d)^2}{g(e-c)}\frac{(c-e)(g(e-c)+(e-d)^2}{g(e-c)}=0$$

$\Leftrightarrow$ [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.

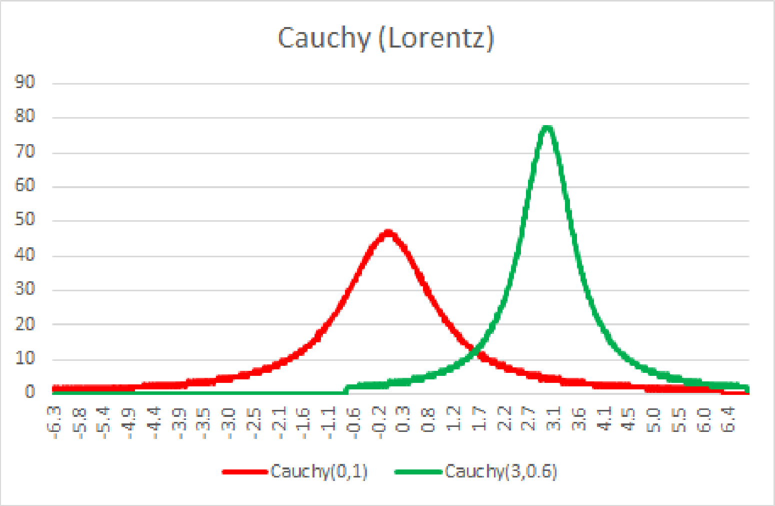
sbRandTrigen Program Code

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.



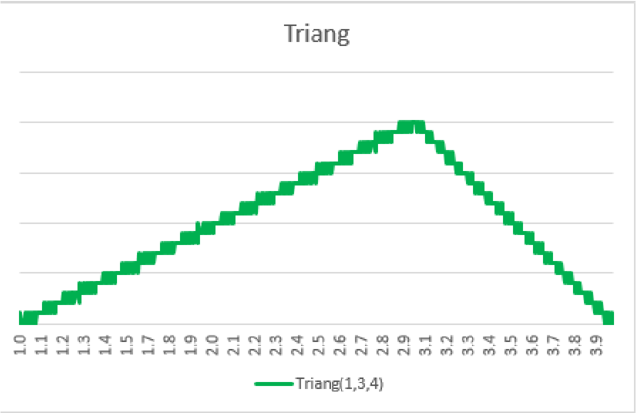
sbRandCauchy Program Code

```
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*.

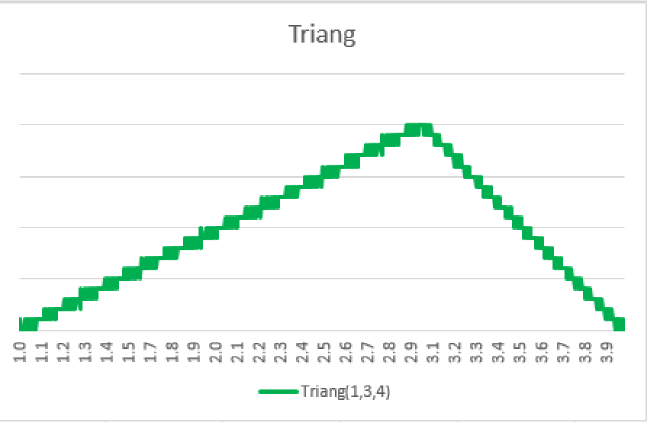
sbRandCFDInv Program Code

```
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:

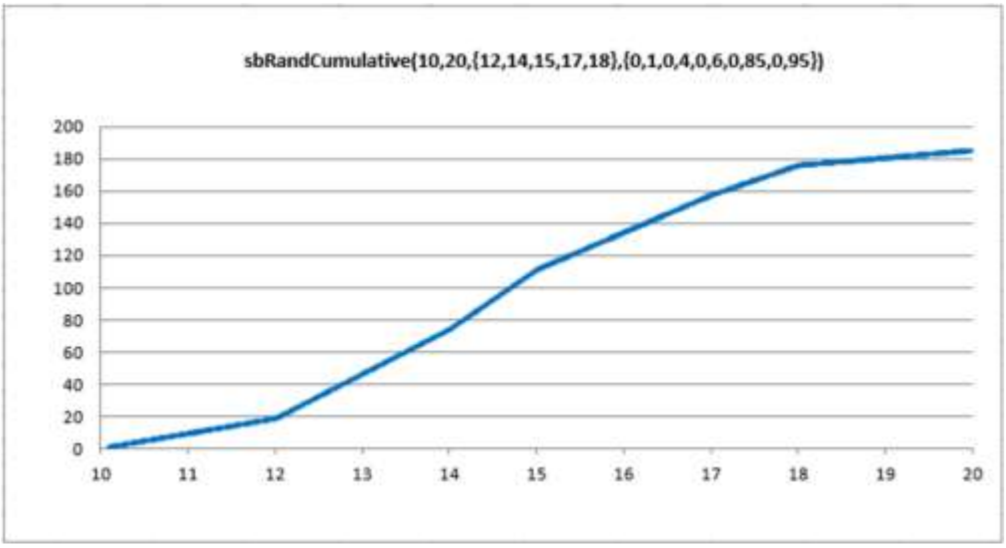


sbRandPDF Program Code

```
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
```

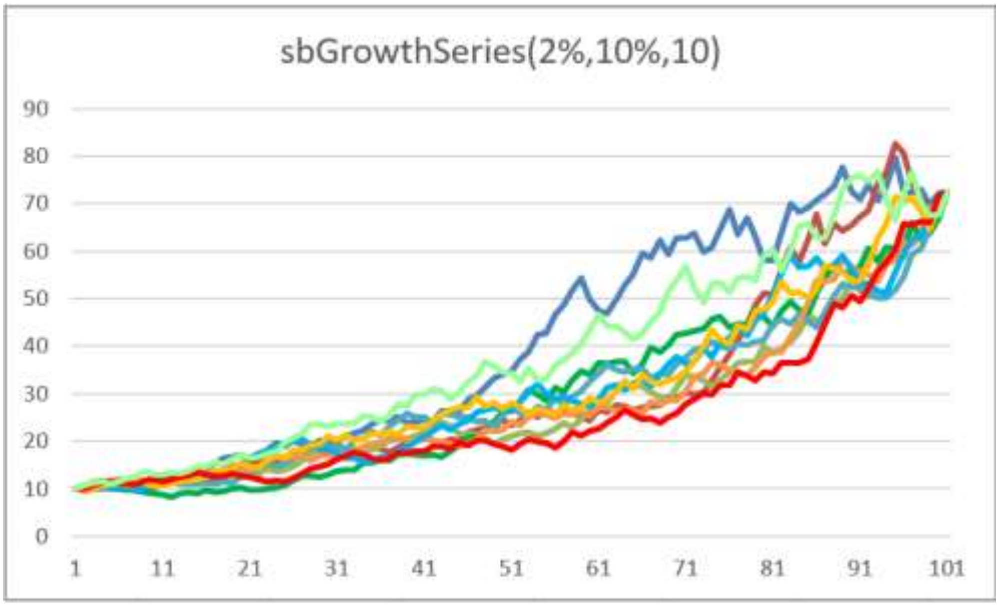
Brownian Bridges

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.

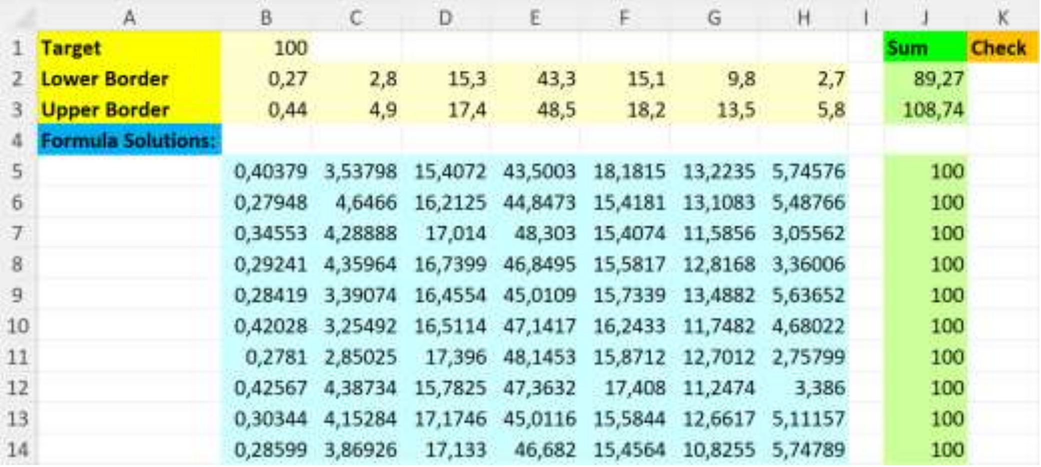


sbGrowthSeries Program Code

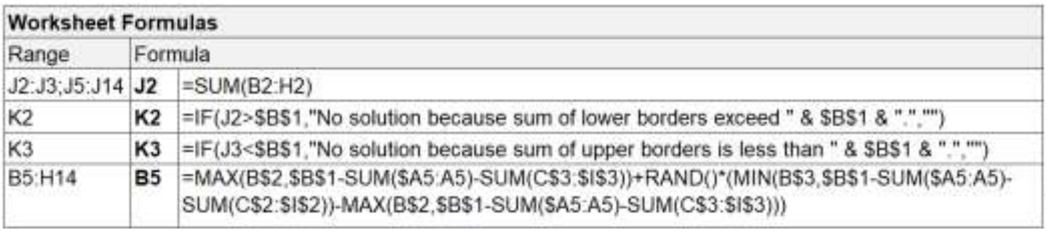
```
Function sbGrowthSeries(dblRate As Double, _
       dblMaxRatePerStep As Double, _
       Optional dblStartVal As Double = 1#) As Variant
'Returns random data with a compound growth rate dblRate, with
'a maximal relative change rate per step of dblMaxRatePerStep
'and with a start value dblStartVal. 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	

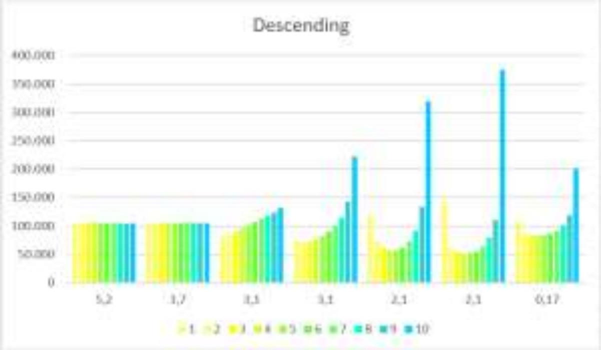


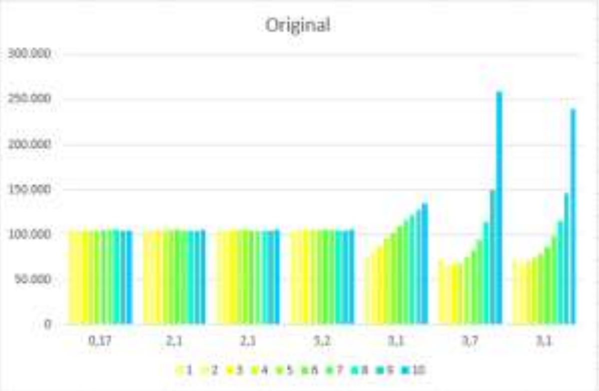
Worksheet Formulas		
Range	Formula	
J2:J3;J5:J14	**J2**	=SUM(B2:H2)
K2	**K2**	=IF(J2>$B$1,"No solution because sum of lower borders exceed " & $B$1 & ".","")
K3	**K3**	=IF(J3<$B$1,"No solution because sum of upper borders is less than " & $B$1 & ".","")
B5:H14	**B5**	=MAX(B$2,$B$1-SUM($A5:A5)-SUM(C$3:$I$3))+RAND()*(MIN(B$3,$B$1-SUM($A5:A5)-SUM(C$2:$I$2))-MAX(B$2,$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

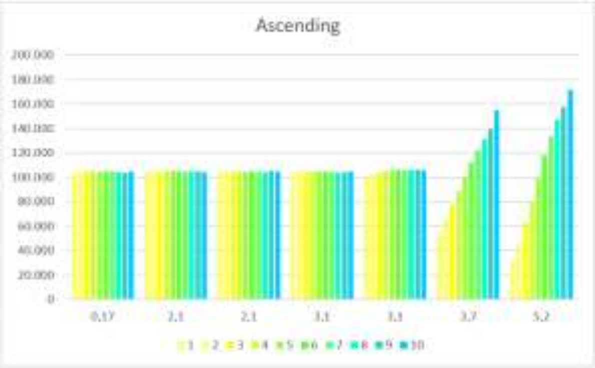
The generated random numbers in the example above are distributed fairly equally. With 1.048.572 generated rows of 7 numbers each you get for the original corridor width sort order (see on the right):





With descending corridor width sort order you get (see left):

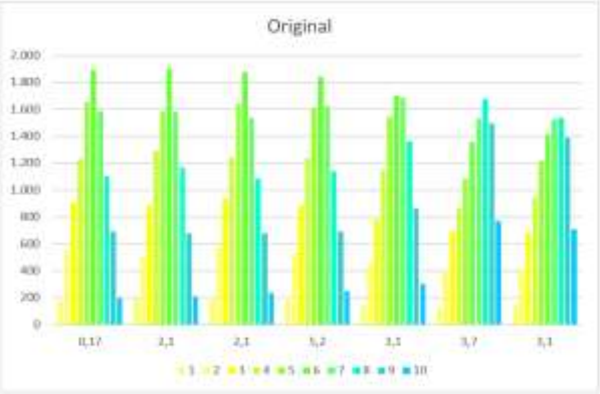
When the corridor widths are sorted ascending:

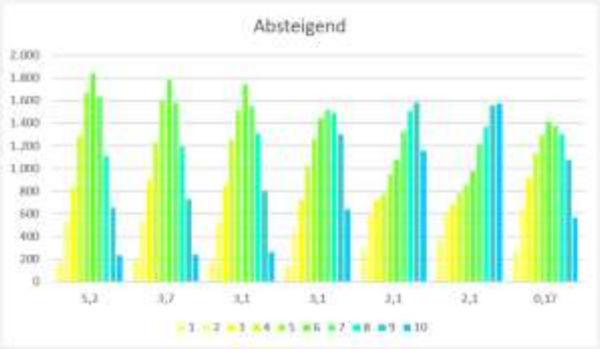


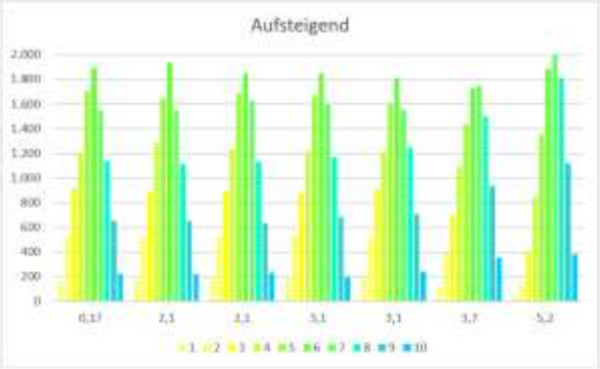
Conclusion: To achieve more equally distributed random numbers you should sort the columns ascending, because the generating formulas reduce the degree of freedom from left to right. If – for whatever reason – you have descending sorted corridor widths, you will have to expect far more extreme distributions.

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).

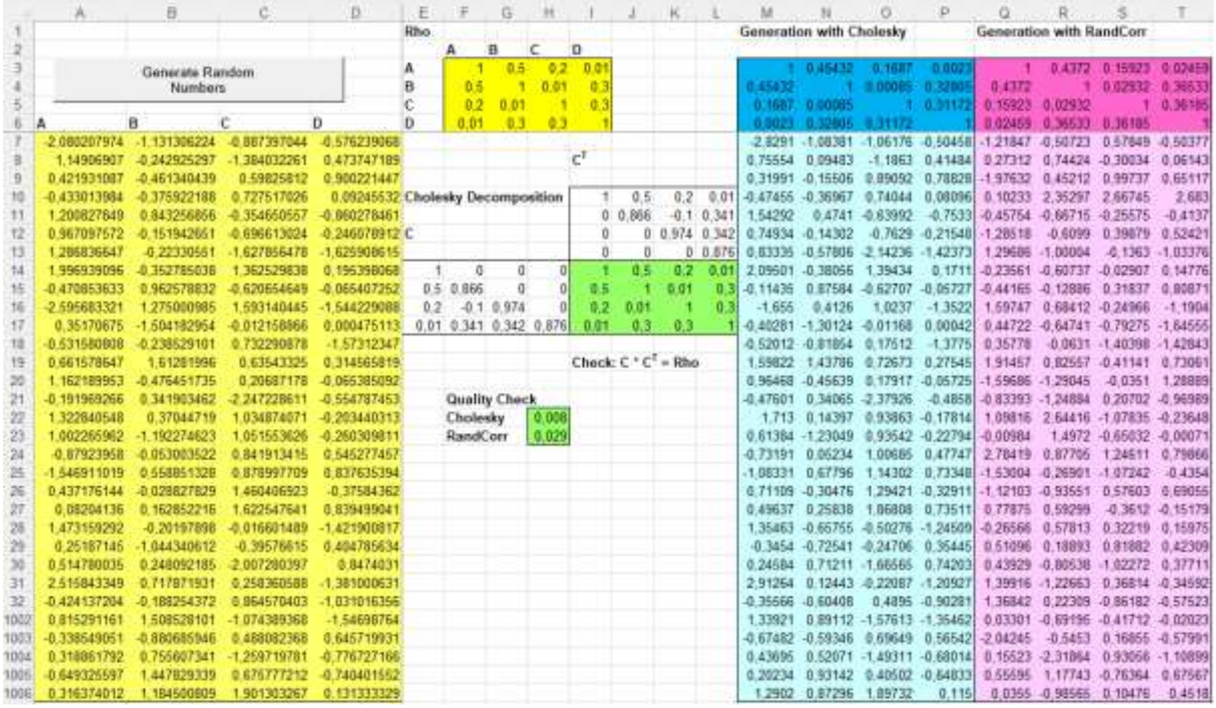
But keep in mind that you need to round at least to the maximal number of digits used in the corridor widths to ensure

- 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.

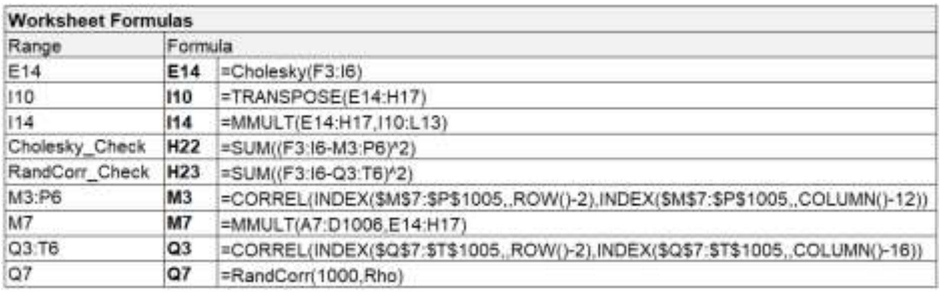
Correlated Random Numbers

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	0,89052	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	-0,151942661	-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	-0,22330561	-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	-0,11436	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	-0,81864	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	0,06234	1,00686	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	0,25187146	-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	-0,69196	-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

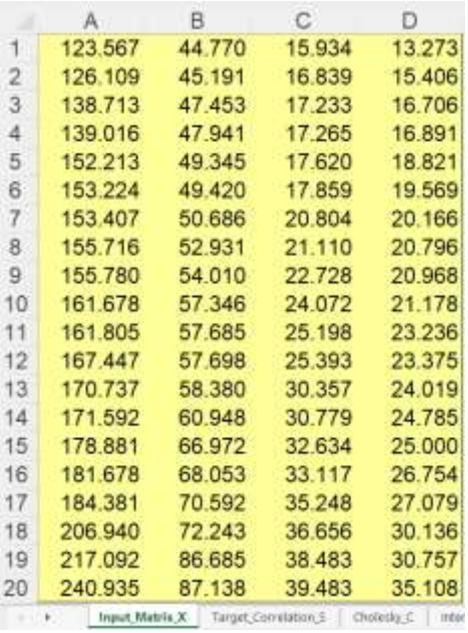
In 2021, Rick Wicklin wrote “Simulate correlated variables by using the Iman-Conover transformation”. His article includes an SAS implementation of the Iman-Conover method:

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”:

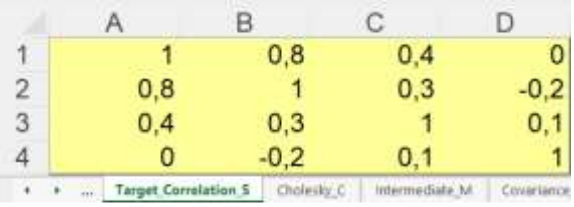
https://www.mynl.com/old/wp/ic.pdf

I implemented the example from Mildenhall’s paper both using Excel spreadsheet functions and with Excel / VBA: The input matrix X (on the right):

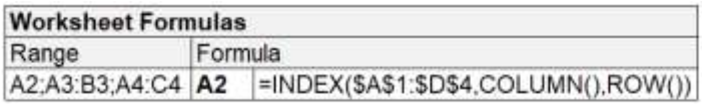


	A	B	C	D
1	123.567	44.770	15.934	13.273
2	126.109	45.191	16.839	15.406
3	138.713	47.453	17.233	16.706
4	139.016	47.941	17.265	16.891
5	152.213	49.345	17.620	18.821
6	153.224	49.420	17.859	19.569
7	153.407	50.686	20.804	20.166
8	155.716	52.931	21.110	20.796
9	155.780	54.010	22.728	20.968
10	161.678	57.346	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

The target correlation S:

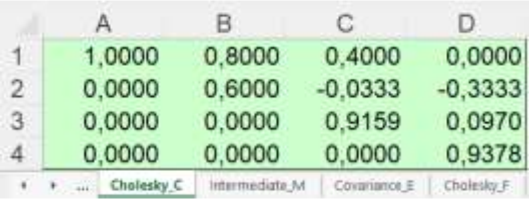


	A	B	C	D
1	1	0,8	0,4	0
2	0,8	1	0,3	-0,2
3	0,4	0,3	1	0,1
4	0	-0,2	0,1	1

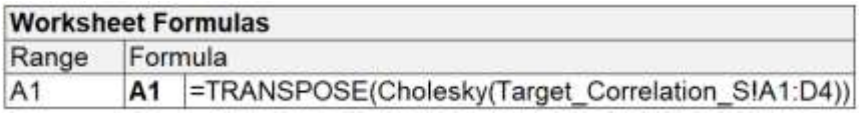


Worksheet Formulas		
Range	Formula	
A2;A3:B3;A4:C4	A2	=INDEX($A$1:$D$4,COLUMN(),ROW())

The Cholesky decomposition C of S:

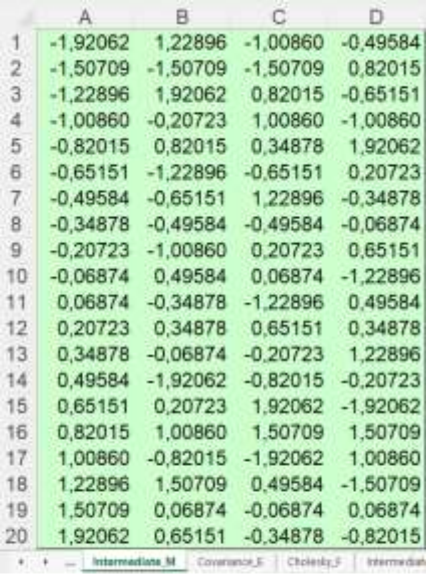


	A	B	C	D
1	1,0000	0,8000	0,4000	0,0000
2	0,0000	0,6000	-0,0333	-0,3333
3	0,0000	0,0000	0,9159	0,0970
4	0,0000	0,0000	0,0000	0,9378



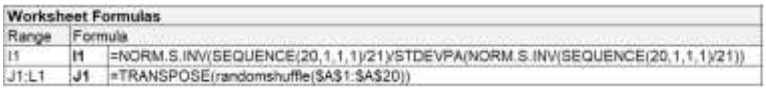
Worksheet Formulas		
Range	Formula	
A1	A1	=TRANSPOSE(Cholesky(Target_Correlation_S!A1:D4))

The intermediate matrix M (constant values, identical to Mildenhall’s data):



	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

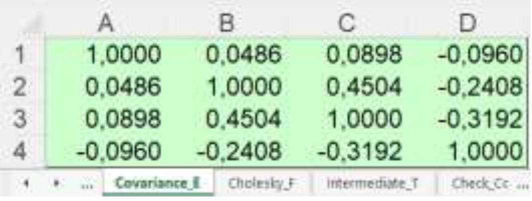


Worksheet Formulas

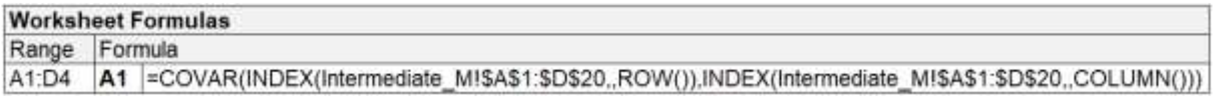
Range	Formula	
I1	I1	=NORM.S.INV(SEQUENCE(20,1,1,1)/21)/STDEVPA(NORM.S.INV(SEQUENCE(20,1,1,1)/21))
J1:L1	J1	=TRANSPOSE(randomshuffle($A$1:$A$20))

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:



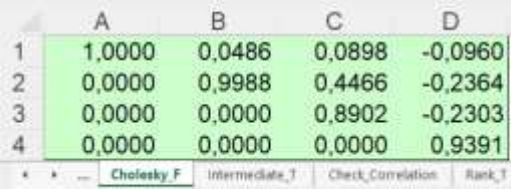
	A	B	C	D
1	1,0000	0,0486	0,0898	-0,0960
2	0,0486	1,0000	0,4504	-0,2408
3	0,0898	0,4504	1,0000	-0,3192
4	-0,0960	-0,2408	-0,3192	1,0000



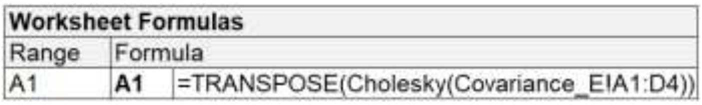
Worksheet Formulas

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:



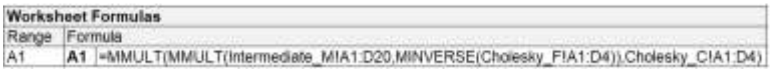
	A	B	C	D
1	1,0000	0,0486	0,0898	-0,0960
2	0,0000	0,9988	0,4466	-0,2364
3	0,0000	0,0000	0,8902	-0,2303
4	0,0000	0,0000	0,0000	0,9391



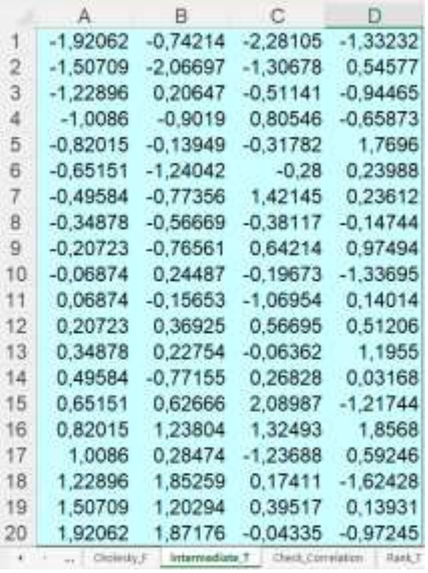
Worksheet Formulas

Range	Formula	
A1	A1	=TRANSPOSE(Cholesky(Covariance_E!A1:D4))

The intermediate matrix T:

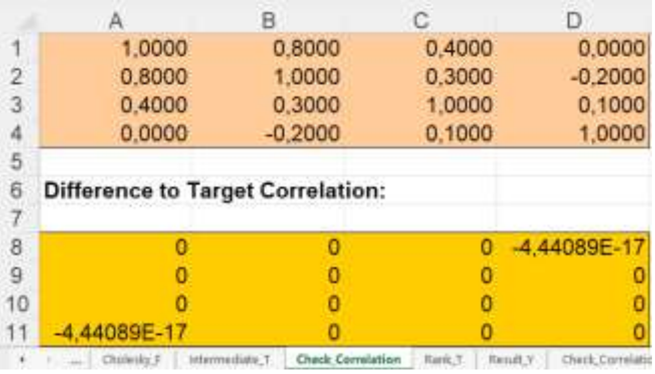


Worksheet Formulas		
Range	Formula	
A1	A1	=MMULT(MMULT(Intermediate_M!A1:D20,MINVERSE(Cholesky_F!A1:D4)),Cholesky_C!A1:D4)

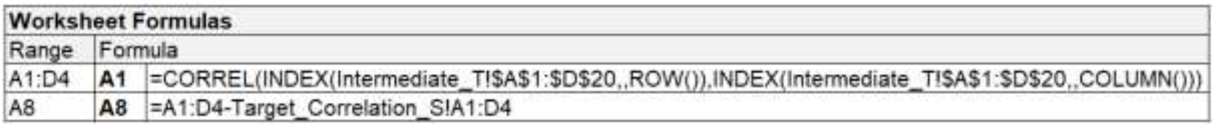


	A	B	C	D
1	-1,92062	-0,74214	-2,28105	-1,33232
2	-1,50709	-2,06697	-1,30678	0,54577
3	-1,22896	0,20647	-0,51141	-0,94465
4	-1,0086	-0,9019	0,80546	-0,65873
5	-0,82015	-0,13949	-0,31782	1,7696
6	-0,65151	-1,24042	-0,28	0,23988
7	-0,49584	-0,77356	1,42145	0,23612
8	-0,34878	-0,56669	-0,38117	-0,14744
9	-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

You can check the generated correlations:

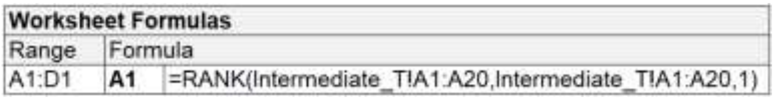


	A	B	C	D
1	1,0000	0,8000	0,4000	0,0000
2	0,8000	1,0000	0,3000	-0,2000
3	0,4000	0,3000	1,0000	0,1000
4	0,0000	-0,2000	0,1000	1,0000
5				
6	Difference to Target Correlation:			
7				
8	0	0	0	-4,44089E-17
9	0	0	0	0
10	0	0	0	0
11	-4,44089E-17	0	0	0

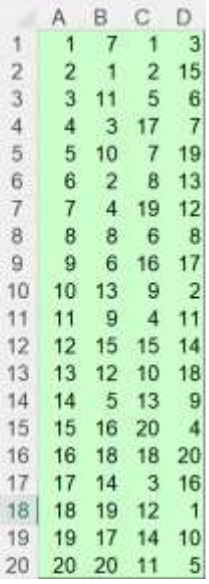


Worksheet Formulas		
Range	Formula	
A1:D4	A1	=CORREL(INDEX(Intermediate_T!$A$1:$D$20,,ROW()),INDEX(Intermediate_T!$A$1:$D$20,,COLUMN()))
A8	A8	=A1:D4-Target_Correlation_S!A1:D4

Calculate the ranks of numbers in columns of T in sheet Rank_T:

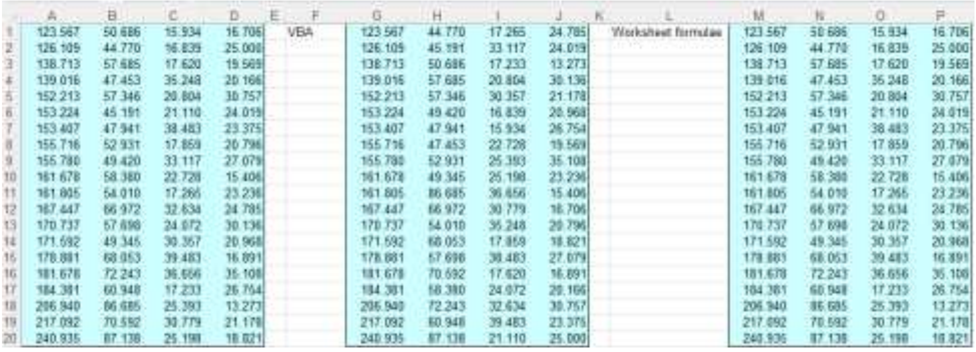


Worksheet Formulas		
Range	Formula	
A1:D1	A1	=RANK(Intermediate_T!A1:A20,Intermediate_T!A1:A20,1)

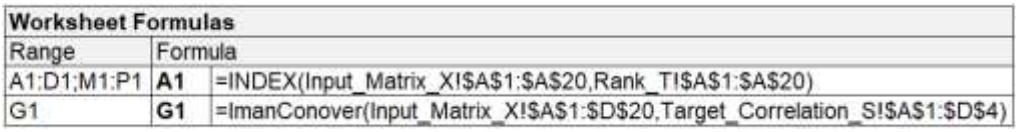


	A	B	C	D
1	1	7	1	3
2	2	1	2	15
3	3	11	5	6
4	4	3	17	7
5	5	10	7	19
6	6	2	8	13
7	7	4	19	12
8	8	8	6	8
9	9	6	16	17
10	10	13	9	2
11	11	9	4	11
12	12	15	15	14
13	13	12	10	18
14	14	5	13	9
15	15	16	20	4
16	16	18	18	20
17	17	14	3	16
18	18	19	12	1
19	19	17	14	10
20	20	20	11	5

Finally you will get the results in sheet Result_Y:

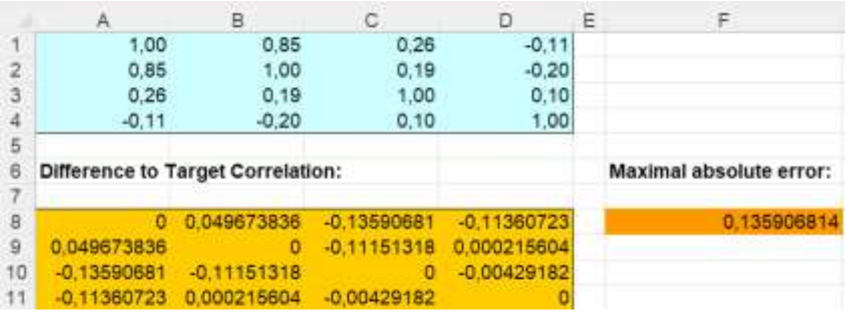


	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	123.567	50.686	15.934	16.706		VBA	123.567	44.770	17.265	24.785		Worksheet formulas	123.567	50.686	15.934	16.706
2	126.109	44.770	16.839	25.000			126.109	45.191	33.117	24.019			126.109	44.770	16.839	25.000
3	138.713	57.685	17.620	19.569			138.713	50.686	17.233	13.273			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

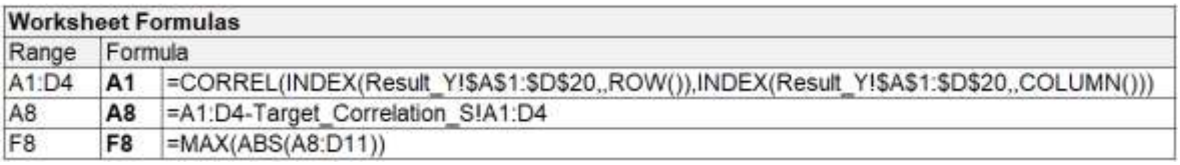


Worksheet Formulas		
Range	Formula	
A1:D1;M1:P1	A1	=INDEX(Input_Matrix_X!$A$1:$A$20,Rank_T!$A$1:$A$20)
G1	G1	=ImanConover(Input_Matrix_X!$A$1:$D$20,Target_Correlation_S!$A$1:$D$4)

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

```vba
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
```

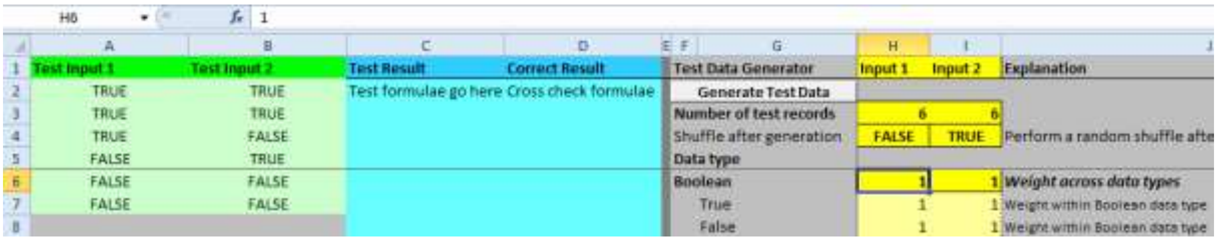
```
  vM(i, 1) = vMV(i)
Next i

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*

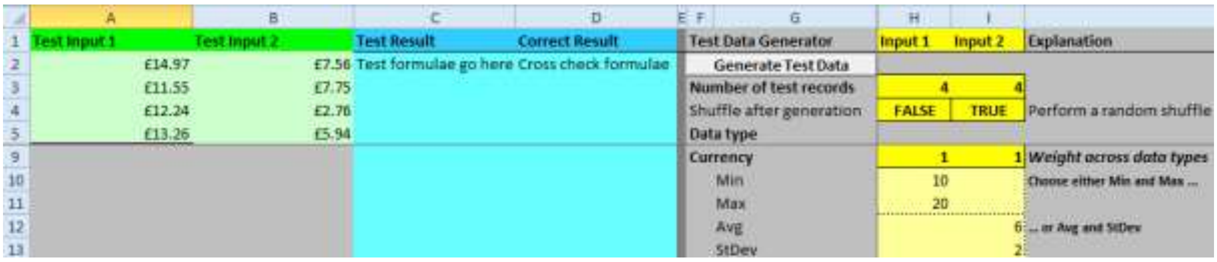
When you want to thoroughly test an application or program, you often need test data. The *sbGenerateTestData* application is designed to help you generate random test data in numerical form or as text.

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::



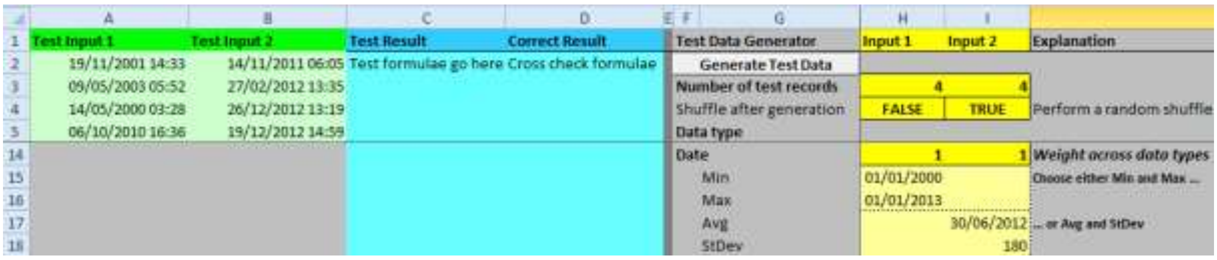
	A	B	C	D	G	H	I	J
1	Test Input 1	Test Input 2	Test Result	Correct Result	Test Data Generator	Input 1	Input 2	Explanation
2	TRUE	TRUE	Test formulae go here	Cross check formulae	Generate Test Data			
3	TRUE	TRUE			Number of test records	6	6	
4	TRUE	FALSE			Shuffle after generation	FALSE	TRUE	Perform a random 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:



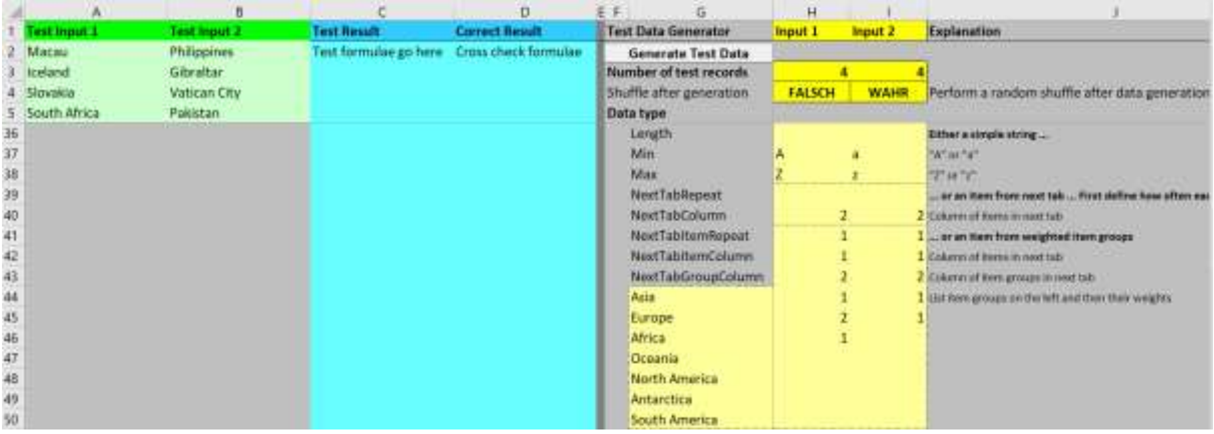
	A	B	C	D	G	H	I	Explanation
1	Test Input 1	Test Input 2	Test Result	Correct Result	Test Data Generator	Input 1	Input 2	Explanation
2	£14.97	£7.56	Test formulae go here	Cross check formulae	Generate Test Data			
3	£11.55	£7.75			Number of test records	4	4	
4	£12.24	£2.76			Shuffle after generation	FALSE	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:



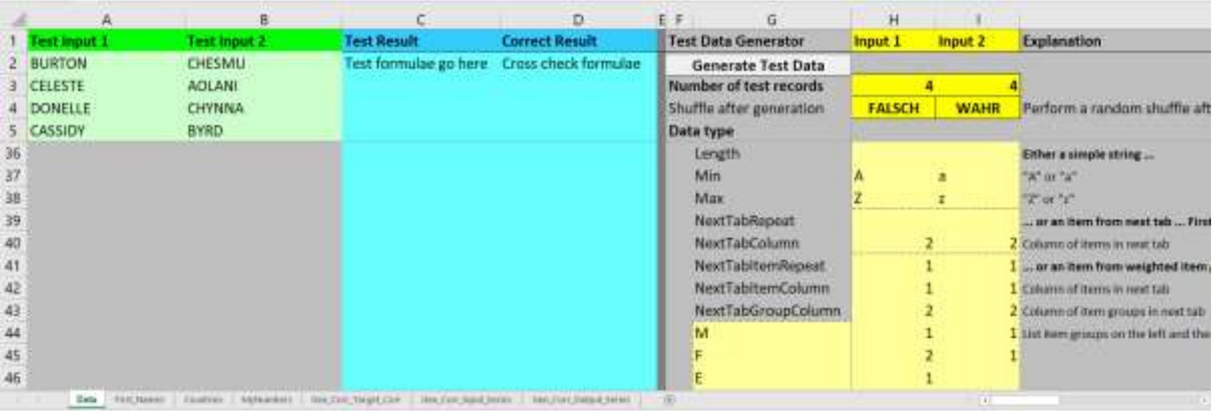
	A	B	C	D	G	H	I	Explanation
1	Test Input 1	Test Input 2	Test Result	Correct Result	Test Data Generator	Input 1	Input 2	Explanation
2	19/11/2001 14:33	14/11/2011 06:05	Test formulae go here	Cross check formulae	Generate Test Data			
3	09/05/2003 05:52	27/02/2012 13:35			Number of test records	4	4	
4	14/05/2000 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	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 ea
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":

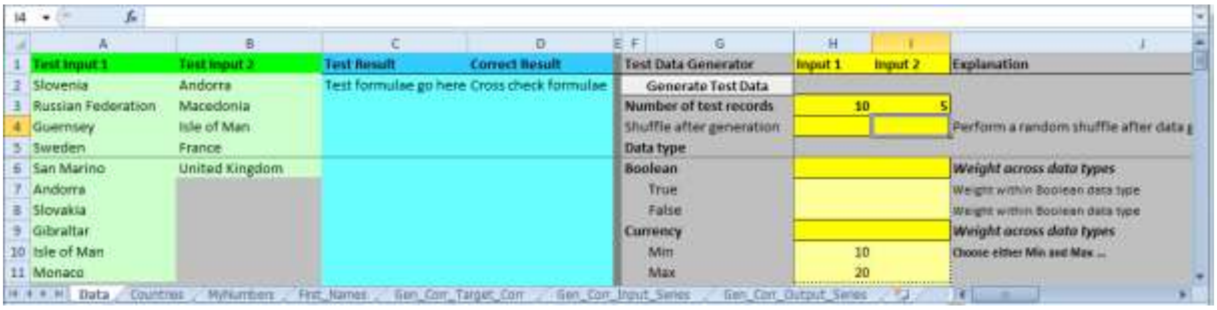


	A	B	C	D	G	H	I	J
1	Test Input 1	Test Input 2	Test Result	Correct Result	Test Data Generator	Input 1	Input 2	Explanation
2	BURTON	CHESMU	Test formulae go here	Cross check formulae	Generate Test Data			
3	CELESTE	AOLANI			Number of test records	4	4	
4	DONELLE	CHYNNA			Shuffle after generation	FALSCH	WAHR	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		

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.

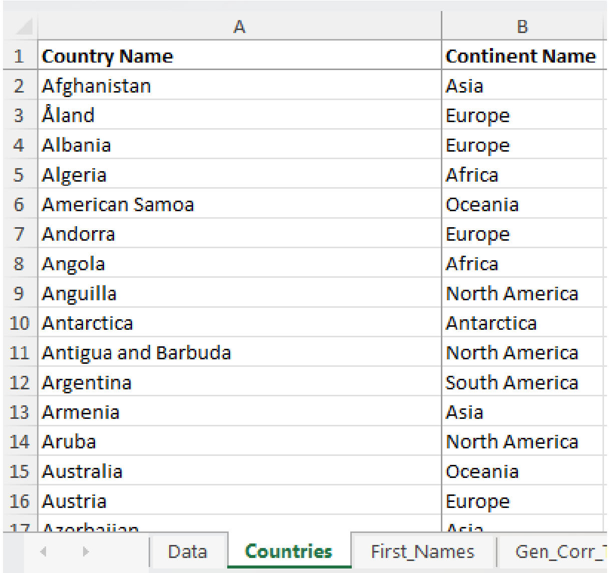
The sheets:



	A	B	C	D	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		

Sheet ‚Countries‘:

Source: https://raw.github.com/wikimedia/limn-data/master/geo/country-codes.csv

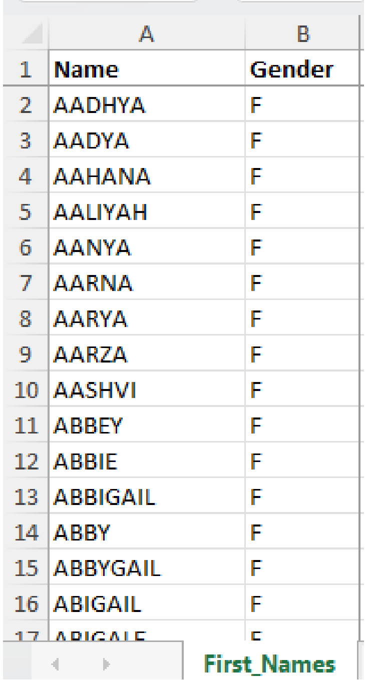


	A	B
1	**Country Name**	**Continent Name**
2	Afghanistan	Asia
3	Åland	Europe
4	Albania	Europe
5	Algeria	Africa
6	American Samoa	Oceania
7	Andorra	Europe
8	Angola	Africa
9	Anguilla	North America
10	Antarctica	Antarctica
11	Antigua and Barbuda	North America
12	Argentina	South America
13	Armenia	Asia
14	Aruba	North America
15	Australia	Oceania
16	Austria	Europe
17	Azerbaijan	Asia

Sheet ‚First_Names‘:

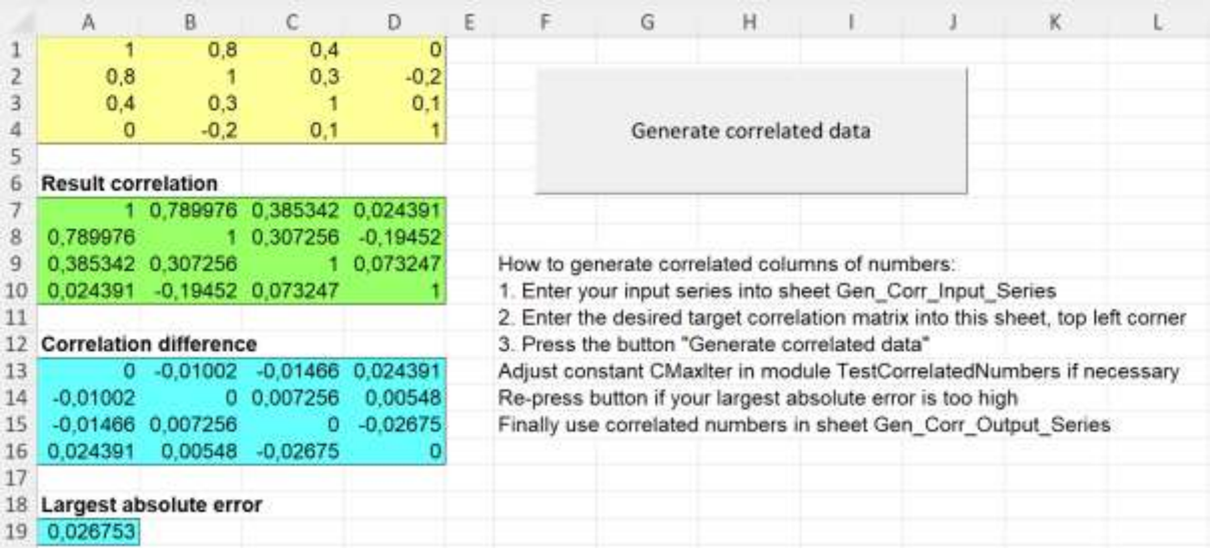
Sources: https://www2.gov.bc.ca/assets/gov/birth-adoption-death-marriage-and-divorce/statistics-reports/baby-names-trends-m-2023.csv

https://www2.gov.bc.ca/assets/gov/birth-adoption-death-marriage-and-divorce/statistics-reports/baby-names-trends-m-2023.csv



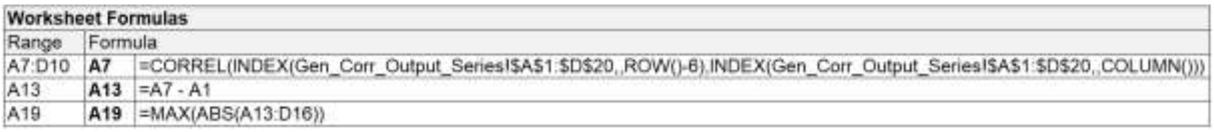
	A	B
1	**Name**	**Gender**
2	AADHYA	F
3	AADYA	F
4	AAHANA	F
5	AALIYAH	F
6	AANYA	F
7	AARNA	F
8	AARYA	F
9	AARZA	F
10	AASHVI	F
11	ABBEY	F
12	ABBIE	F
13	ABBIGAIL	F
14	ABBY	F
15	ABBYGAIL	F
16	ABIGAIL	F
17	ABIGALE	F

Correlated random numbers you generate with this application as follows:



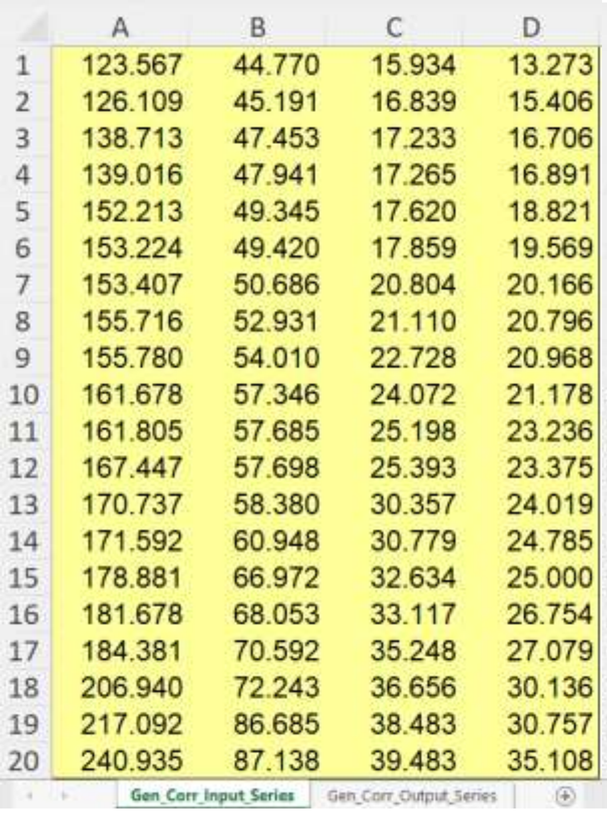
	A	B	C	D
1	1	0,8	0,4	0
2	0,8	1	0,3	-0,2
3	0,4	0,3	1	0,1
4	0	-0,2	0,1	1
5				
6	**Result correlation**			
7	1	0,789976	0,385342	0,024391
8	0,789976	1	0,307256	-0,19452
9	0,385342	0,307256	1	0,073247
10	0,024391	-0,19452	0,073247	1
11				
12	**Correlation difference**			
13	0	-0,01002	-0,01466	0,024391
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			

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

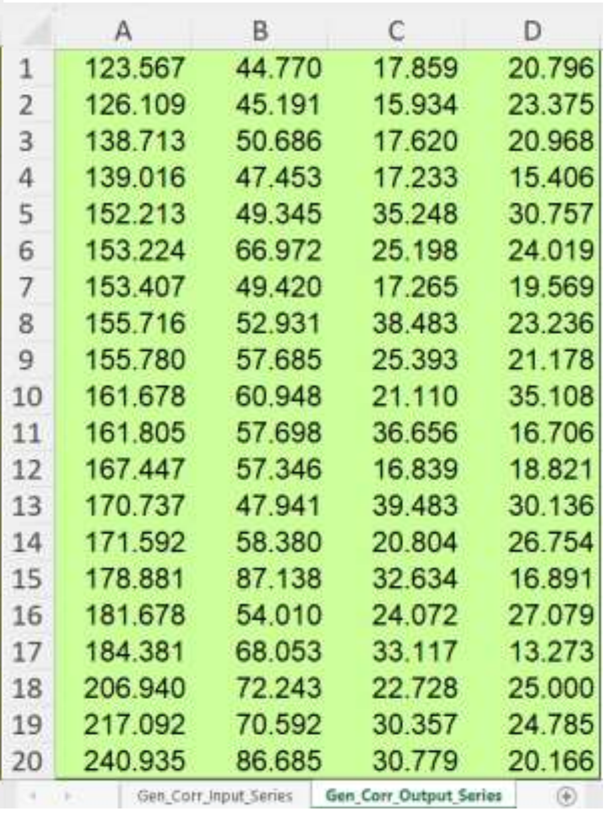


Worksheet Formulas

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



	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

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
```

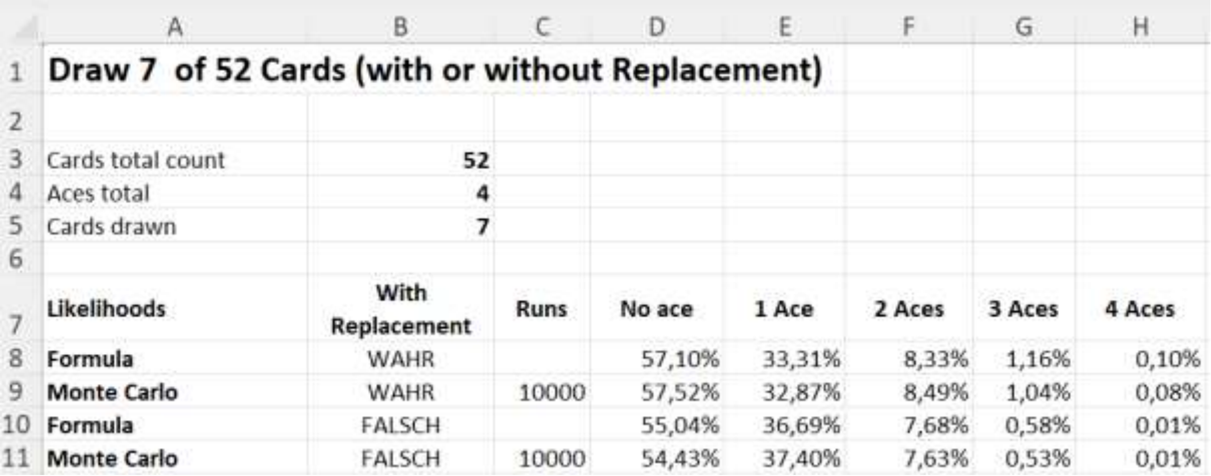
```
          End If
          dSumWeights = 0#
          j = 0
          Do While Not IsEmpty(wsD.Cells(pr_sNextTabGroupWeights + j, pc_ItemGroups))
            dSumWeights = dSumWeights + wsD.Cells(pr_sNextTabGroupWeights + j, lCol)
            j = j + 1
          Loop
          ReDim dGroupWeights(1 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
```

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%.



Draw 7 of 52 Cards (with or without Replacement)

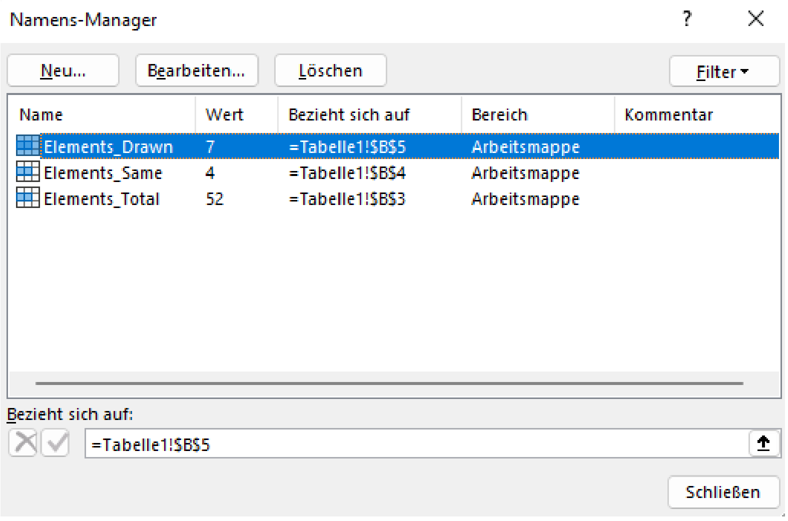
Cards total count	**52**
Aces total	**4**
Cards drawn	**7**

Likelihoods	**With Replacement**	**Runs**	**No ace**	**1 Ace**	**2 Aces**	**3 Aces**	**4 Aces**
Formula	WAHR		57,10%	33,31%	8,33%	1,16%	0,10%
Monte Carlo	WAHR	10000	57,52%	32,87%	8,49%	1,04%	0,08%
Formula	FALSCH		55,04%	36,69%	7,68%	0,58%	0,01%
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:



Name	Wert	Bezieht sich auf	Bereich	Kommentar
Elements_Drawn	7	=Tabelle1!$B$5	Arbeitsmappe	
Elements_Same	4	=Tabelle1!$B$4	Arbeitsmappe	
Elements_Total	52	=Tabelle1!$B$3	Arbeitsmappe	

Bezieht sich auf: =Tabelle1!$B$5

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
```

Index