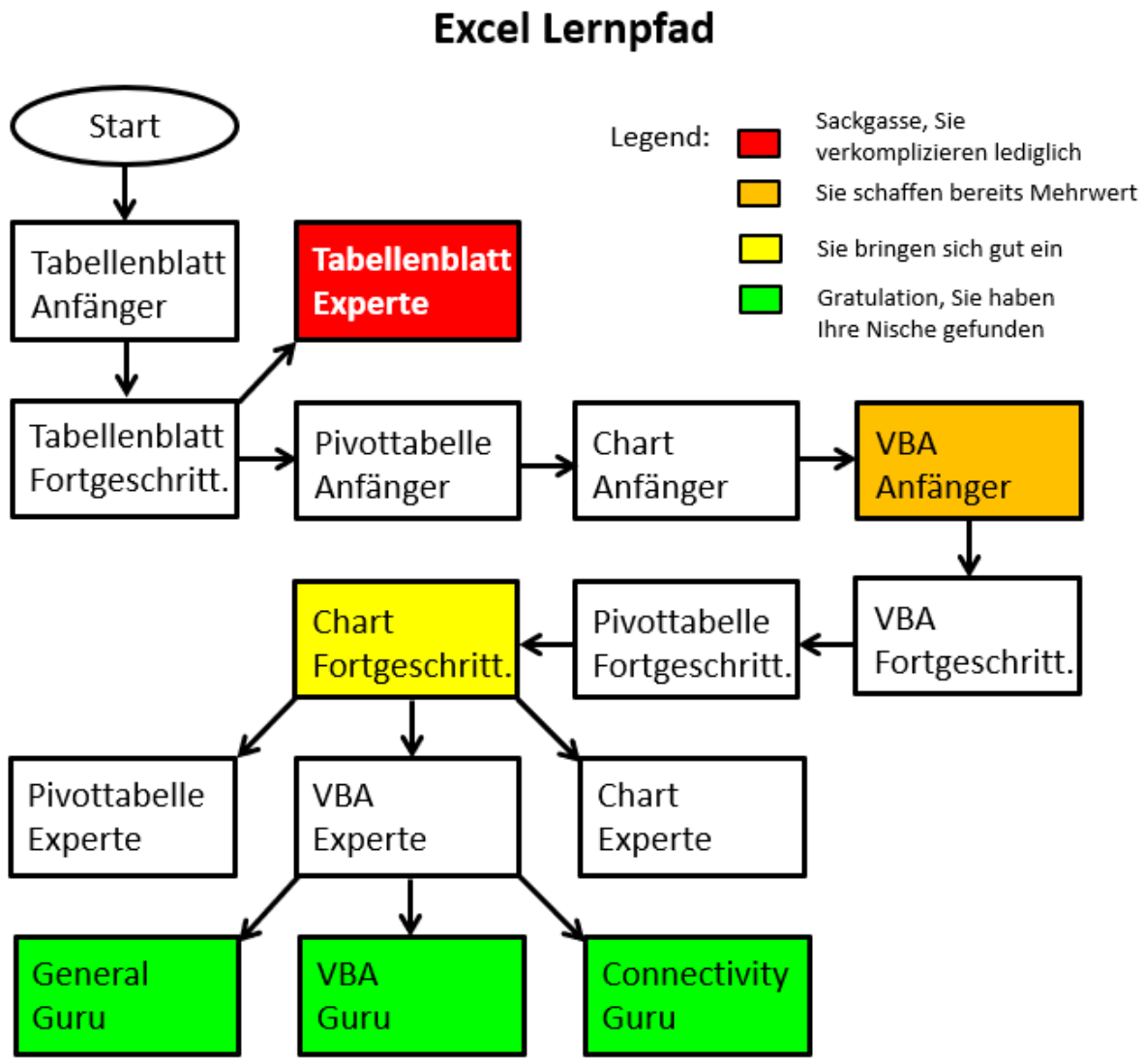
Excel / VBA – Eine Sammlung



 Stand: 28. Dezember 2024

Abstract

Dies ist eine Sammlung von Excel VBA Programmen und auch einigen Excelformeln, die mir sinnvoll erschienen.

Bernd Plumhoff, 28. Dezember 2024

Inhaltsverzeichnis

Die Excel / VBA Programmierumgebung

Abstract

Mit Visual Basic for Applications (VBA kann man in der Tabellenkalkulation Excel Aufgaben automatisieren und spezielle Funktionalitäten programmieren, die im Funktionsumfang von Excel nicht enthalten sind.

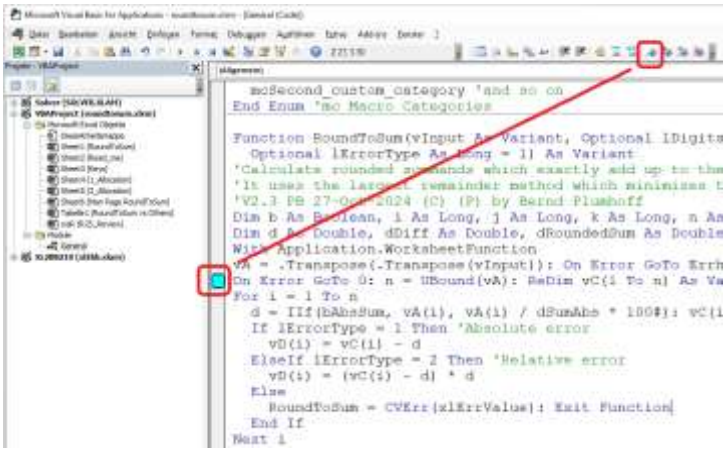
Hier zeige ich Programme, die ich nützlich fand, als ich Betriebsprozesse planen, umsetzen und ausführen musste.

Grundlegendes

Während des Editierens

Mit `Option Explicit` erzwingen Sie in einem VBA Modul die Deklaration von allen verwendeten Variablen. Wer dies nicht verwendet, sollte nicht programmieren
Wenn der Cursor in der VBA Programmierungebung auf einer Variablen oder einem Objeckt steht, springen Sie mit SHIFT und der Funktionstaste F2 (SHIFT + F2) direkt zur Deklaration dieser Variablen (nur Dim, nicht ReDim). Mit STRG + SHIFT + F2 springen Sie zurück.

Im VBA Editor können Sie mit dem Symbol „blaue Flagge“ Textstellen markieren, zu denen Sie mit den angrenzenden Zeigersymbolen schnell springen können:



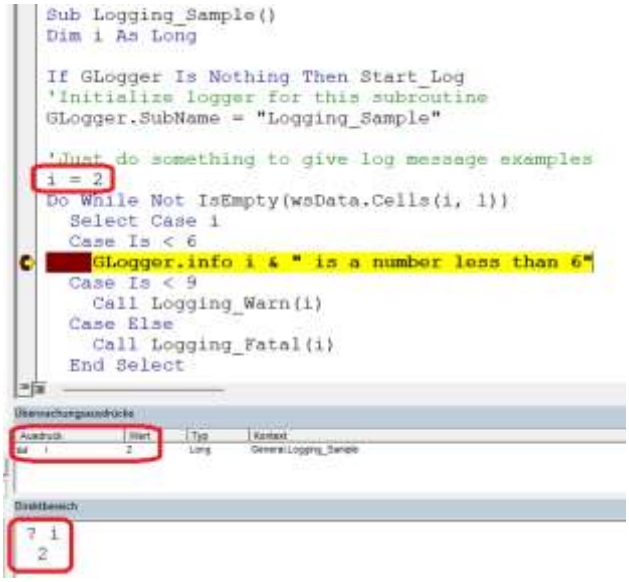
Mit dem Programmbefehl `Stop` oder wenn Sie auf den grauen Bereich gleich links neben einer Programmzeile klicken, können Sie einen Stopppunkt (engl. breakpoint) setzen:

```
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
```

Wenn das Programm später ausgeführt wird, dann stoppt es an dieser Stelle, und Sie können z. B. den Inhalt von Variablen oder Tabellenblättern prüfen.

Während der Programmausführung

Wenn ein VBA Programm ausgeführt wird, können Sie es durch Drücken der ESC Taste unterbrechen. Auch der Befehl `Stop` oder ein Stopppunkt unterbricht das Programm, wenn es an die entsprechende Stelle gelangt, und kennzeichnet die aktuelle Programmzeile gelb:



Sie können nun mit dem Cursor zur Variablen i an der Stelle „i = 2“ gehen (nicht klicken, lediglich darüber schweben, „hovern“). Es wird ein kleines Fenster i = 2 gezeigt. Sie können auch i auswählen, die rechte Maustaste drücken und eine Überwachung für i hinzufügen. Dann wird der Wert von i im Fenster Überwachungsausdrücke gezeigt.

Mit STRG + g kommen Sie nach einer Programmunterbrechung in den Direktbereich (engl. Immediate Window) . Hier können Sie einzelne Programmbefehle eingeben wie z. B. „Print i“ (oder einfach kurz „? I“ – das Fragezeichen ist das Kürzel für den Befehl Print).

Sie können nach einer Programmunterbrechung das Programm mit der Funktionstaste F5 weiterlaufen lassen. Sie können es aber mit F8 auch weiter schrittweise Befehl-für-Befehl ausführen. Mit SHIFT + F8 an Stelle von F8 verzweigt das Programm nicht in Unterprogramme, sondern führt diese als einen Befehl aus.

Gute Programmierpraxis

Seien Sie ein guter Programmierer

Das Wichtigste an einem guten VBA Programm ist, dass es ein gutes Programm ist, und nicht voller VBA Tricks. Wenn Sie keine assoziativen Arrays und keine Klassen kennen, dann müssen sie auf dem Boden herumkriechen und bekommen Ihre Anwendung nie zum Fliegen.

Gutes Excel und VBA Wissen

Sie sollten Excel and VBA gut beherrschen. Mit dem VBA Befehl *Enum* können Sie z. B. Tabellenspalten Namen geben. Dies vereinfacht Programmänderungen - oder wollen Sie alle hart kodierten Verweise auf Spalten rechts von einer neu eingefügten oder gelöschten Spalte anpassen?

Programmierkonventionen

Lernen Sie Namenskonventionen und Kodierkonventionen. Durch ihre Anwendung machen Sie Ihre Programme lesbarer, einfacher zu warten, und verlässlicher und effizienter.
Beispiellinks:
https://de.wikibooks.org/wiki/VBA_in_Excel/_Namenskonventionen
https://learn.microsoft.com/de-de/dotnet/visual-basic/programming-guide/program-structure/coding-conventions

Säubern Sie Makroaufzeichnungen

Selbstverständlich nehme ich ein Makro auf, wenn ich einen VBA Befehl vergessen habe. Aber wenn Sie aufgezeichneten ungesäuberten Spaghetticode verwenden, muss dies Ihr Nachfolger aufräumen.

Dokumentieren Sie Ihr Programm ausreichend

Erklären Sie was ein kundiger Dritter wissen muss, aber schreiben Sie keine Romane über Trivialitäten. Gute Dokumentation kommt mit dem Programm - nicht danach. Nur Dummköpfe und Menschen, die sich unentbehrlich machen wollen, dokumentieren nicht.

Testen Sie Ihr Programm gut

Anwendungen ab einer gewissen Größe benötigen Testprogramme oder sogar eine Serie von Regressionstests.

Protokollieren Sie Ihre Programmausführung

Mit einer Logger Klasse (Programmablauf dokumentieren – *Logging* Klasse) können Sie einem Revisor zeigen, wer Ihr Programm mit welchen Parametern ausführte und ob Ihr Programm problemlos durchlief.

Optimieren Sie Ihr Programm

Die Qualität Ihrer Anwendung wird maßgeblich durch ihr Design und ihre Struktur bestimmt. Je komplexer die Aufgabe, desto besser sollte Ihr Fachwissen und Ihre Erfahrung sein. Durch das Messen der Laufzeiten einzelner Programmteile (engl. Profiling) können Sie erkennen, wo noch Verbesserungen möglich sind.

Jan Karel Pieterse hat eine hilfreiche Klasse für das Profiling erstellt:
https://jkp-ads.com/Articles/performanceclass.asp

Systemstatus sichern und zurückschreiben – *SystemState* Klasse

Mein ehemaliger Kollege Jon T. entwickelte die kleinste mir bekannte sinnvolle VBA Klasse: Mit *SystemState* kann man Systemstatusvariablen leicht speichern und für eigene Zwecke optimieren.

Um die Programmausführung zu beschleunigen, schreibt man normalerweise zu Beginn eines VBA Makros

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

und am Ende des Makros

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

Mit dem Klassenmodul *SystemState* schreibt man am Start lediglich

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

und am Ende

```
Set state = Nothing 'Nicht einmal nötig - dies wird automatisch gemacht
```

Systemstatus Variablen

Die Klasse *SystemState* sichert und restauriert die folgenden Systemstatus Variablen:

Variable	Status	Kommentar / Zu optimieren durch …
Calculation	xlCalculationAutomatic, xlCalculationManual, xlCalculationSemiautomatic	Bestimmt ob nach jeder Zelländerung eine Neuberechnung durchgeführt wird. Auf xlCalculationManual setzen.
Cursor	xlDefault, xlBeam, xlNorthwestArrow, xlWait	Dies ist lediglich eine Anzeige. Am besten nicht anfassen - es sei denn, man möchte den Debug Modus z. B. mit einem Sanduhr Zeiger beginnen.
DisplayAlerts	Wahr, Falsch	Auf Falsch (engl. False)setzen um Systemrückfragen abzuschalten.
EnableAnimations	Wahr, Falsch	Ab Excel Version 2016 kann man hiermit Excel's Bildschirmanimationen abschalten.
EnableEvents	Wahr, Falsch	Auf Falsch (engl. False)setzen um Ereignisprozeduren an der Ausführung zu hindern.
Interactive	Wahr, Falsch	Am besten nicht anfassen - es sei denn, man möchte unbedingt alle Tastatureingaben verhindern.
PrintCommunication	Wahr, Falsch	Auf Falsch (engl. False)setzen um Seiteneinstellungen zu ändern ohne auf Rückantwort vom Drucker zu warten.
ScreenUpdating	Wahr, Falsch	Auf Falsch (engl. False) setzen um Bildschirmaktualisierungen während der Programmausführung abzuschalten.
StatusBar	Falsch, "Eine beliebige Benutzerinformation"	Der Text wird in der Statuszeile (unterste Bildschirmzeile) gezeigt. Auf Falsch (engl. False) setzen um die Anzeige zu löschen.

SystemState Programmcode

Bitte kopieren Sie den folgenden Programmcode in ein Klassenmodul mit dem Namen *SystemState*, nicht in ein normales Modul.

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

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

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

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

'
```

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

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

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

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

Programmablauf dokumentieren – *Logging* Klasse

Diese Logger Klasse bietet Logging mit den Berichtsstufen INFO, WARN, FATAL und EVER an. Die Programminformationen werden sowohl in einem Tabellenblatt als auch in einer Datei festgehalten.

Die Anwendung dieser Logger Klasse ist nicht schwer: Einfach das allgemeine Modul Logger_Factory und das Klassenmodul Logger aus der unten bereitgestellten Beispieldatei in die eigene Anwendung kopieren, dann die Public Const AppVersion zum Beispiel mit dem Wert “Meine Anwendung Version 1.0” im Hauptmodul definieren, und dann kann man mit

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

die eigenen Logmeldungen erzeugen und automatisch im Tabellenblatt Workflow und in der Logdatei “Meine Anwendung Version 1.0_Logfile_yyyymmdd.log” im Unterverzeichnis Logs speichern.

Ich erhielt den ursprünglichen Programmcode 2009 von Cliff G. und erweiterte ihn später. Cliff verwendete dieses Programm hauptsächlich zum Debuggen. Ich finde es auch sehr sinnvoll, um Programmläufe zu Revisionszwecken zu protokollieren, oder damit ein Programm dem Benutzer ggf. detailliert seine einzelnen Ausführungsschritte erläutert. Weiter fügte ich Versionsinformationen und System- oder Excel-Parameter hinzu, um rasch wichtige Unterschiede zwischen verschiedenen Benutzerumgebungen zu ermitteln. Mit diesem Logger messe ich gewöhnlich auch einfache Laufzeiten von SQL Abfragen:

```
'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]"
```

Für und Wider

Dieses Logging Programm bietet meines Erachtens die sinnvollste sekundäre Funktionalität für jede VBA Anwendung. Man kann:

- nachvollziehbar testen
- ein Programm alle seine Ausführungsschritte nachvollziehbar erklären lassen
- einfach feststellen, ob mehrere Anwender eine Anwendung gleichzeitig nutzen
- leicht erkennen, ob ein Anwenderproblem auf einer unterschiedlichen Umgebung beruht
- auch sporadische Anwendungsfehler systematisch eingrenzen
- über einen längeren Zeitraum hinweg auch Revisoren überzeugend die korrekte fehlerfreie Nutzung nachweisen (einzelne Logdateien können zwar manipuliert werden, aber eine größe Menge von Logdateien wirkt dennoch hinreichend überzeugend)
- die Laufzeit von VBA (Unter-)Routinen grob bestimmen
- die Durchlaufzeiten von gesamten Prozessen messen

Der letzte obige Punkt wird Betriebs- und Personalräte hellhörig machen:

- wenn man Durchlaufzeiten von ganzen Prozessen misst, könnte man die Leistung einzelner Mitarbeiter ermitteln, vergleichen und ggf. gegen sie verwenden.

Dies wäre ein klarer Verstoß gegen die DSGVO (Datenschutz-Grundverordnung), siehe https://dsgvo-gesetz.de/

Ich habe dieses Logging nie gegen meine Mitarbeiter oder Anwender für eine Leistungsmessung verwendet, sondern lediglich für Nachschulungen genutzt, wenn ich fehlerhafte Nutzungen erkannte. Aber dies kann selbstverständlich nicht als Argument für eine unbedenkliche Nutzung dienen.

Eine Zustimmung von Betriebs- und Personalräten kann m. E. immer erreicht werden, wenn man auf die Freiwilligkeit dieser Selbstaufschreibung hinweist:

- jeder Anwender kann das Logging vor einem Programmlauf ein- oder ausschalten
- jeder Anwender kann die Log-Dateien zu jeder Zeit im Nachhinein löschen

Ich setzte und setze in Europa in mehreren Ländern (UK, Deutschland) bei mehreren Gesellschaften (Banken, Versicherungen, IT Providern) dieses Logging ohne jede Beanstandung erfolgreich ein.

Parameter

Public (öffentliche) Konstanten

AppVersion - Diese Zeichenkette sollte den Programmnamen und seine Version enthalten, z. B.:

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

Dann wird "… Version x" als Versionsinformation für dieses Programm protokolliert.

Compilerkonstanten

Separate_Log_Files_for_each_User - True erstellt für jeden Benutzer eigene Logdateien, False führt zu einer täglichen Logdatei für alle Benutzer

Use_Logger_auto_Open_Close - True verwendet die Subroutinen auto_open und auto_close im Modul LoggerFactory, False nicht.

Logging_on_Screen - In LoggerFactory und Logger auf True setzen um Nachrichten auch im Tabellenblatt Workflow zu zeigen.

Logging_cashed - In LoggerFactory und Logger auf True setzen um das Logging zu beschleunigen. Log Nachrichten werden dann erst am Ende des Programlaufs in eine Datei geschrieben. Hierfür muss auch Logging_on_Screen auf True gesetzt werden.

Log_WMI_Info - In LoggerFactory auf True setzen um interessante Windows Management Instrumentation (WMI) Informationen auszugeben wie z. B. Prozessor-, Speicher-, Laufwerks- und Betriebssystem-Angaben.

Show_Reference_Details - True zeigt alle Details, False zeigt lediglich die Beschreibung.

Logging Variablen

LogFilePath - Vollständiger Pfadname der Logdatei

SubName - Muss am Anfang jeder Subroutine gesetzt werden um den Sub Namen korrekt im Log zu protokollieren

LogLevel - Die Logging Berichtsstufen:

```
1 - Alle Log Nachrichten protokollieren: INFO, WARN, FATAL, and EVER
2 - Alle Log Nachrichten mit Ausnahme von Stufe INFO protokollieren
3 - Nur FATAL und EVER Log Nachrichten protokollieren
4 - Lediglich EVER Log Nachrichten protokollieren
5 - Kein Logging
```

LogScreenRow - Startzeile für das Logging in Tabellenblatt Workflow (gewöhnlich 3)

Beispielausgabe

Die Logs werden standardmäßig im Tabellenblatt Workflow und in der Logdatei im Unterverzeichnis Logs ausgegeben:

```
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
```

Module

Die Logs werden standardmäßig im Tabellenblatt Workflow und in der Logdatei im Unterverzeichnis Logs ausgegeben:

Normal

LoggerFactory enthält Konstanten, öffentliche Variablen, Standard Logger Einstellungen und optionale Auto-Open and Auto-Close Subroutinen.

Hinweis: Die Prozedur *Start_Log* benötigt (ruft auf) die Prozeduren *ApplicationVersion* und *getOperatingSystem* (https://www.devhut.net/vba-determine-the-installed-os/) sowie das Modul *LibFileTools* (https://github.com/cristianbuse/VBA-FileTools), um auch Verzeichnisse unter OneDrive und Sharepoint zu unterstützen.

```
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
```

Ein Beispielmodul General welches zeigt wie man den Logger nutzen kann:

```
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
```

Klassenmodule

Logger enthält die Logging Funktionalität:

```
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
```

Zahlensysteme, Formate und Umwandlungen

Abstract

Als Programmierer hat man häufig mit Zahlensystemen und deren Darstellung bzw. Umwandlung zu tun. Hier stelle ich einige Programme vor, die ich im Laufe der Zeit kennen- und nutzen lernte.

Umwandlungen und Berechnungen von Zahlen

Zahlen in Worten ausgeben – *sbInWorten*

Manchmal müssen Sie Zahlen in Worten ausgeben, z. B. in Euro/Cent oder in Dollars/Cents oder in britischen Pfund Sterling/Pence. 12,31 würde zum Beispiel als “Zwölf Euro und Einunddreißig Cent” ausgegeben werden.

Hinweis: Im Web existiert eine Vielzahl von *SpellNumber*- und *InWorten*-Versionen. Leider sind die meisten davon fehlerhaft. Testen Sie am besten mit den hier genannten Beispielzahlen:

	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 Programmcode

```
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
```

Umwandlungen zwischen dem Dezimalsystem und dem Binärsystem

Wie lautet die binäre Darstellung (Bitlänge = 256) der Dezimalzahl -872362346234627834628734627834627834628? Die eingebaute Excel Funktion DEZINBIN hilft hier nicht weiter. Sie kann nur Eingaben von -512 bis 511 verarbeiten. Wenn Sie die korrekte Antwort
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111101011011111011010100011111100111011110010111100100001000011101011001001010011001101000100
110011110101000010101010010111100111111100
erhalten wollen, benutzen Sie bitte die unten gezeigte Funktion *sbDec2Bin*.

Bitte bachten: Nachkommastellen werden lediglich für positive Zahlen unterstützt. Zum Beispiel lautet die Dezimalzahl 0.5 im Binärformat 0.1.

Funktion	Parameter 1	Parameter 2	Ergebnis	Kommentar
sbDec2Bin	2005		1111101010	
sbDec2Bin	2005	11	#Wert!	Mit 11 Bit lässt sich 2005 nicht darstellen
sbDec2Bin	11,5		1011,1	
sbDec2Bin	2,25		10,01	
sbBin2Dec	1111		15	
sbBin2Dec	1111	4	-1	4 Bit: das erste Bit ist das Vorzeichen
sbBIn2Dec	11,11		3,75	

sbDec2Bin, sbBin2Dec, sbDivBy2, sbBinNeg und sbDecAdd Programmcode

```
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
```

Feiertage ermitteln – *IstFeiertag*

Wollen Sie herausfinden, ob ein Datum ein deutscher Feiertag ist? Oder ob es ein Feiertag in einem Bundesland ist?

Hier sind alle Feiertage von 2023 bis 2100 gezeigt. Sie können die Liste manuell ändern, z. B. Feiertage hinzufügen oder entfernen. Diese Liste kann mit dem unten angebotenen Programm Feiertage_generieren.xlsm automatisch erzeugt werden, wobei aber historische Feiertagsänderungen (z. B. Abschaffung des Buß- und Bettages seit 1995 mit Ausnahme Sachsens) und Gebietsreformen (z. B. Wiedervereinigung 1990) nicht berücksichtigt sind.

Für Schulferien (besser: schulfreie Tage) können die Feiertage einfach erweitert werden: Man fügt alle einzelnen Ferientage zu den entsprechenden Bundeslandspalten hinzu und benennt die benutzerdefinierte Funktion *IstFeiertag* um in *IstSchulfreierTag*.

IstFeiertag Programmcode

```
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 Programmcode

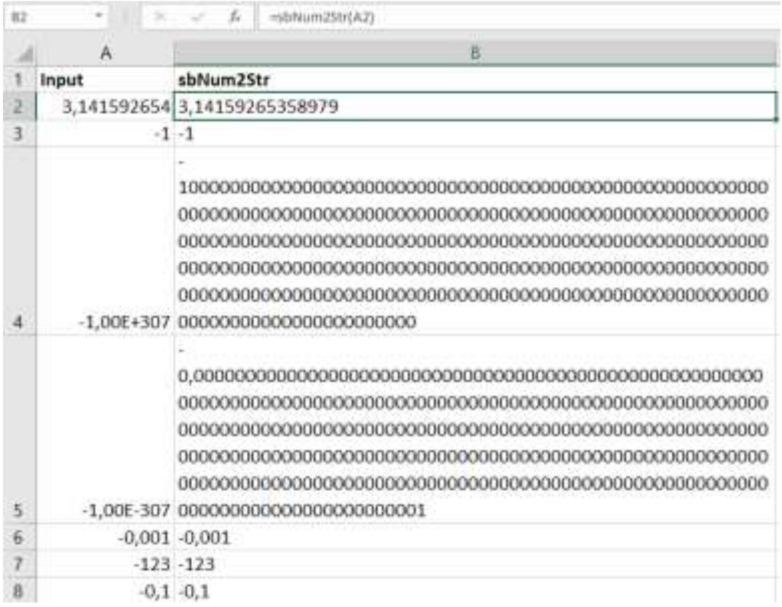
```
Const StartJahr = 2023
Const EndJahr = 2100

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
```

Zahl vollständig nicht-wissenschaftlich darstellen – *sbNum2Str*

Wenn Sie eine nicht-wissenschaftliche Zahlendarstellung mit allen signifikanten Ziffern und allen vorangestellten und nachstehenden Nullen benötigen, können Sie diese Funktion *sbNum2Str* verwenden:



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

sbNum2Str Code

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

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

Nummer eines Monatsnamens – *sbMonatsZahl*

Falls Sie die Nummer für einen Monatsnamen ermitteln wollen:

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

sbMonatsZahl Programmcode

```
Function sbMonatsZahl(sMonat As String) As Integer
'Gibt die Monatsnummer für einen Monatsnamen zurück.
'(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"
        sbMonatsZahl = 4 'Abrele, April, Aprilius
    Case "u"
        sbMonatsZahl = 8 'August, Augschte, Augusti
    Case Else
        sbMonatsZahl = CVErr(xlErrNum)
    End Select
Case "b"
    sbMonatsZahl = 6 'Brachet, Brachmond
Case "c", "d"
    sbMonatsZahl = 12 'Chrischtmonet, Dezember, December, Decembris, Christmond
Case "e"
    sbMonatsZahl = 8 'Erntemond
Case "f"
    sbMonatsZahl = 2 'Februar, February, Feber
Case "h"
    c2 = Mid(s, 2, 1)
    Select Case c2
    Case "a"
        sbMonatsZahl = 1 'Hartung
    Case "e"
        c3 = Mid(s, 3, 1)
        Select Case c3
        Case "r"
            sbMonatsZahl = 9 'Herbschtmonet, Herbstmond
        Case "u"
            sbMonatsZahl = 7 'Heuet, Heuert, Heumond
        Case Else
            sbMonatsZahl = CVErr(xlErrNum)
        End Select
    Case "o"
        sbMonatsZahl = 2 'Hornig, Hornung
    Case Else
        sbMonatsZahl = CVErr(xlErrNum)
    End Select
Case "j"
    c2 = Mid(s, 2, 1)
    Select Case c2
    Case "a", "ä", "e"
        sbMonatsZahl = 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"
                sbMonatsZahl = 7 'Juli, July, Juley
            Case "m"
                sbMonatsZahl = 12 'Julmond
            Case Else
                sbMonatsZahl = CVErr(xlErrNum)
            End Select
        Case "n"
            sbMonatsZahl = 6 'Juni, June, Juno
        Case Else
            sbMonatsZahl = CVErr(xlErrNum)
        End Select
    Case Else
        sbMonatsZahl = CVErr(xlErrNum)
    End Select
Case "l"
    sbMonatsZahl = 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"
            sbMonatsZahl = 5 'Mai, May
        Case "r"
            sbMonatsZahl = 3 'March, Marty, Martii
        Case Else
            sbMonatsZahl = CVErr(xlErrNum)
        End Select
    Case "ä"
        c3 = Mid(s, 3, 1)
        Select Case c3
        Case "i"
            sbMonatsZahl = 5 'Mäie
        Case "r"
            sbMonatsZahl = 3 'März
        Case Else
            sbMonatsZahl = CVErr(xlErrNum)
        End Select
    Case "e"
        sbMonatsZahl = 3 'Merze
    Case Else
        sbMonatsZahl = CVErr(xlErrNum)
    End Select
Case "n"
    sbMonatsZahl = 11 'November, Nebelmond
Case "o"
    c2 = Mid(s, 2, 1)
    Select Case c2
    Case "c", "k"
        sbMonatsZahl = 10 'Oktober, October, Oktobris
    Case "s"
        sbMonatsZahl = 4 'Ostermond
    Case Else
        sbMonatsZahl = CVErr(xlErrNum)
    End Select
Case "s"
    c2 = Mid(s, 2, 1)
    Select Case c2
    Case "a"
        sbMonatsZahl = 4 'Saating
    Case "c"
        c3 = Mid(s, 3, 1)
        Select Case c3
        Case "h"
            c4 = Mid(s, 4, 1)
            Select Case c4
            Case "e"
                sbMonatsZahl = 9 'Scheiding
            Case "n"
                sbMonatsZahl = 1 'Schneemond
            Case Else
                sbMonatsZahl = CVErr(xlErrNum)
            End Select
        Case Else
            sbMonatsZahl = CVErr(xlErrNum)
        End Select
    Case "e"
        sbMonatsZahl = 9 'September, Septembris
    Case Else
        sbMonatsZahl = CVErr(xlErrNum)
    End Select
Case "w"
    sbMonatsZahl = 9 'September
    c2 = Mid(s, 2, 1)
```

```
    Select Case c2
    Case "e"
        sbMonatsZahl = 10 'Weinmond
    Case "i"
        c3 = Mid(s, 3, 1)
        Select Case c3
        Case "i"
            sbMonatsZahl = 10 'Wiimonet
        Case "n"
            sbMonatsZahl = 11 'Wintermonet
        Case Else
            sbMonatsZahl = CVErr(xlErrNum)
        End Select
    Case "o"
        sbMonatsZahl = 5 'Wonnemond
    Case Else
        sbMonatsZahl = CVErr(xlErrNum)
    End Select
Case Else
    sbMonatsZahl = CVErr(xlErrNum)
End Select
End Function
```

Die Berechnung der Kreiszahl π

Wie viele Dezimalstellen der Kreiszahl π werden für die Praxis gebraucht?

Der Hamburger Mathematikprofessor Hermann Schubert zeigte 1889, wieviel Stellen nicht mehr benötigt werden: Man stelle sich eine Kugel vor, in deren Mitte die Erde liegt. Der Radius sei so groß wie die Entfernung der Erde vom Sirius: 8,8 Lichtjahre. Diese Kugel sei mit Mikroben gefüllt, von denen viele Millionen in einen Kubikmillimeter passen. Nun fädele man alle Mikroben in dieser Kugel an einem straffen Faden auf und lasse zwischen je zwei Mikroben einen Platz von 8,8 Lichtjahren. Diesen Faden nehme man als Durchmesser eines Kreises. Multipliziert man diese Länge mit π (auf 100 Dezimalstellen genau), so weicht das Ergebnis vom exakten Umfang des Kreises weniger als ein Millionstel Millimeter ab.

Dieses Beispiel zeigt, daß die Berechnung von π auf 100 Stellen oder mehr vollkommen nutzlos ist.

Warum wird dann versucht, π auf immer mehr Stellen zu berechnen?

In jüngster Zeit möchte man erfahren, ob die Folge der Dezimalziffern gewissen Gesetzen genügt oder nicht. Auch ist nicht bekannt, wie häufig alle Ziffern in der Dezimaldarstellung vorkommen.

Bis in die Mitte des 17. Jahrhunderts wurde π nach der Methode des Archimedes berechnet. Er berechnete die Fläche des Einheitskreises näherungsweise mit Polygonflächen. James Gregory entdeckte 1671 die Potenzreihe

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

für -1 <= x <= 1.

Mit dieser Reihenentwicklung und der Gleichung

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

wird π in der jüngsten Zeit berechnet.

Herleitung von (*):

Sei $-\frac{\pi}{2} < \alpha + \beta < \frac{\pi}{2}$.

Das Additionstheorem für den Tangens lautet $\tan(\alpha + \beta) = \frac{\tan\alpha + \tan\beta}{1 - \tan\alpha \tan\beta}$.

Wird $\alpha = \arctan x, \beta = \arctan y$ gesetzt, so ergibt sich für xy ≠ 1

$\tan(\alpha + \beta) = \frac{x+y}{1-xy}$

so dass $\alpha + \beta = \arctan x + \arctan y = \arctan \frac{x+y}{1-xy}$ ist.

Eine Tabelle zur Geschichte der Näherungen von π

Jahr	Stellen	Name
-225	2	Archimedes
1579	9	Viete
1593	15	Roomen
1610	35	Ceulen
1699	72	Sharp
1719	112	de Lagny
1794	140	von Vega
1794	200	Dahse
1873	527	Shanks
945-48	808	Smith & Wrench
1949	2.037	Reitwiesner
954-55	3.089	Nicholson & Jeenel
1958	10.000	Genuys
1959	16.167	Genuys
1961	100.000	Shanks & Wrench
1966	250.000	Gilloud & Fillatoire
1967	500.000	Gilloud & Dichampt

Pi Programmcode

```
Option Explicit

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
```

Pi Ausgabe auf 1000 Stellen

```
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
```

Die Berechnung der Eulerschen Zahl *e*

Zur Berechnung der Eulerschen Zahl *e* verwendet man die bekannte Summenformel

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

Also ist

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

Jede reelle Zahl 0 <= x < 1 kann man darstellen als (möglicherweise unendlichen) “variablen Basis”-Bruch $.a_1a_2a_3\ldots$, wobei für alle i gilt: 0 <= a(i) < 1. Die dargestellte Zahl ist

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

Die reelle Zahl *e* - 2 wird demnach dargestellt durch den unendlichen Bruch .111… . Das Problem der Berechnung von *e* reduziert sich damit auf die Umwandlung der obigen Darstellung in Dezimaldarstellung. Je mehr Stellen man für die Dezimaldarstellung ermitteln will, desto “länger” muss der “variable Basis”-Bruch sein, den man umwandelt.

Beispiel: Berechnung von e auf 1000 Stellen. Ab welchem Index n kann man den Reihenrest vernachlässigen? Sei y(n) die (n+1)-te Partialsumme der unendlichen Reihe

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

Es gilt (Quelle: Fichtenholz Band I, Nr. 37):

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

Für dieses Beispiel muss gelten:

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

Dafür wähle man n = 500.

e Programmcode

```
Option Explicit

Sub e()

'e 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
```

e Ausgabe auf 1000 Stellen

```
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
```

Literatur

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.

Zahlenfolge kürzer darstellen – *sbParseNumSeq*

Gliedere eine Zahlenfolge auf und gib eine kürzere Darstellung zurück: 1,2,3,5,6,7 wird zurückgegeben als 1-3,5-7. Falls *bWithSingleDouble* = TRUE, dann wird 1,3,5,6,8,10 zurückgegeben als 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 Programmcode

```
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
```

Rationale Zahlen = Brüche

Ermittle die nächstliegende rationale Zahl zu einer Gleitkommazahl – *sbNRN*

Welche rationale Zahl ist eine gute Näherung von π (3,1415926…)? Geben Sie ein: in Zelle A1 '=pi()', in Zelle B1 den maximal gewünschten Nenner (zum Beispiel 10), und als Matrixformel (mit STRG + SHIFT + ENTER) in den Zellen C1:D1 '=*sbNRN*(A1;B1)'. Als Ergebnis erscheint in C1:D1 22 und 7. Dies bedeutet: 22/7 ist die nächstliegende rationale Zahl (Bruch) zu π mit einem Nenner nicht größer als 10. Mit 1000 in B1 würde man 355/113 erhalten.

Dieser Algorithmus findet nicht immer die nächstliegende rationale Zahl zu einer gegebenen Gleitkommazahl mit einem gewünschten maximalen Nenner und der vordefinierten maximalen Fehlerschranke 1# / (2# * CDbl(lMaxDen) ^ 2#). Die gute Nachricht ist jedoch, dass er dann einen #ZAHL! Fehler zurückgeben würde. In einem solchen Fall geben Sie bitte eine größere individuelle maximale Fehlerschranke vor.

Die ursprüngliche Absicht des Autoren Oliver Aberth bestand in der Unterstützung exakter Berechnungen von Brüchen, zum Beispiel bei der Lösung von linearen Gleichungssystemen mit rationalen Koeffizienten.

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

Bemerkung: Die letzte Zeile in der obigen Grafik sagt uns nicht, dass wir den Kreis erfolgreich quadriert haben. Wir haben lediglich (meines) Excel's Genauigkeitsgrenze erreicht.

Die Bruchdarstellungen der TEXT Funktion wurden zum Vergleich angezeigt. Beispiel: =TEXT(PI();"?/?") = "22/7". Microsoft hat diese Darstellung nicht für die 64-Bit Version erweitert. Genauer als PI() = "5419351/1725033" kann nicht gezeigt werden. Genauer wäre mit 64-Bit PI() = "245850922/78256779", aber dann ist der absolute Fehler ist natürlich längst kleiner als 1e-15.

<u>Grenzen der Berechnung</u>

Excel kann Dezimalzahlen von -9,99999999999999E+307 bis 9,99999999999999E+307 darstellen. Die 64-Bit Version von Excel kann ganze Zahlen des Typs LongLong von -9223372036854775808 bis 9223372036854775807 darstellen, was in etwa dem Umfang -1E+10 bis 1E+10 entspricht. Es ist offensichtlich, dass Aberth's Algorithmus in Excel nicht alle verfügbaren Dezimalzahlen hinreichend genau als Bruch ermitteln kann.

Name

sbNRN - Berechne die nächstliegende rationale Zahl zu einer gegebenen Gleitkommazahl mit einem gegebenen maximalen Nenner

Synopsis

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

Beschreibung

sbNRN berechnet die nächstliegende rationale Zahl zu der gegebenen Gleitkommazahl *dFloat* mit dem maximalen Nenner *lMaxDen* und der optionalen maximalen absoluten Fehlerschranke *dMaxErr*.

Parameter

dFloat - Die Gleitkommazahl für die die nächstliegende rationale Zahl gefunden werden soll

lMaxDen - Die Obergrenze für den Nenner

dMaxErr - Optional - Die Obergrenze für den absoluten Fehler (absolute Differenz zwischen der eingegebenen Gleitkommazahl und der auszugebenden rationalen Zahl)

Literatur

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

Exkurs

Falls Sie lediglich die Relation zur Zehnerpotenz benötigen, verwenden Sie die folgende Formel:

```
=WENNFEHLER(-A2*10^(LÄNGE(-A2)-SUCHEN(",";-A2))&":"&10^(LÄNGE(-A2)-SUCHEN(",";-A2));-A2&":1")
```

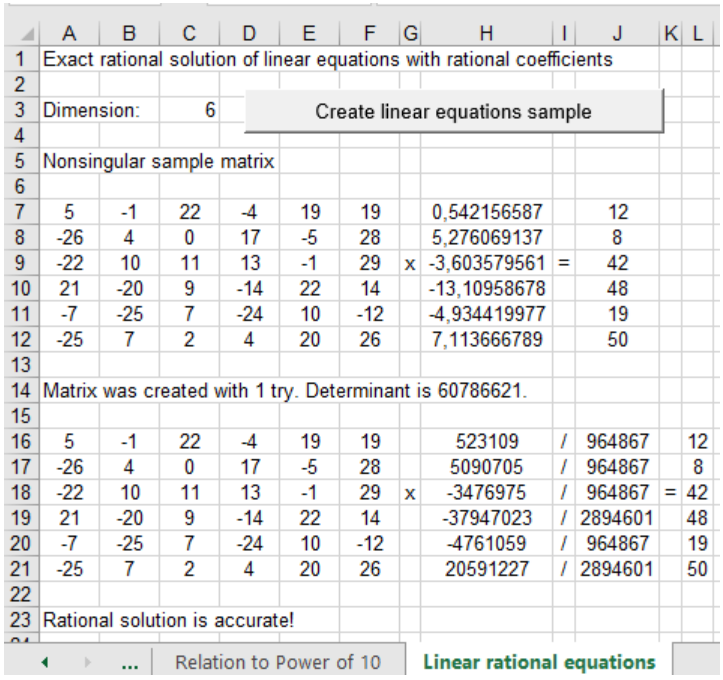
	A	B	C
1	Eingabe	Ausgabe	Formel in B
2	1E-14	1:100000000000000	=WENNFEHLER(--A2*10^(LÄNGE(--A2)-SUCHEN(",";--A2))&":"&10^(LÄNGE(--A2)-SUCHEN(",";--A2));--A2&":1")
3	0,00001	1:100000	=WENNFEHLER(--A3*10^(LÄNGE(--A3)-SUCHEN(",";--A3))&":"&10^(LÄNGE(--A3)-SUCHEN(",";--A3));--A3&":1")
4	0,1	1:10	=WENNFEHLER(--A4*10^(LÄNGE(--A4)-SUCHEN(",";--A4))&":"&10^(LÄNGE(--A4)-S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

```
Option Explicit

#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
```

Lineare Gleichungssysteme mit rationalen Koeffizienten

Lineare Gleichungssysteme der Form A * x = b mit der regulären quadratischen Matrix A und dem Ergebnisvektor b haben eine eindeutige Lösung, da die Determinante von A ungleich Null ist. Sind die Koeffizienten von A und b rationale Zahlen, so ist auch die Lösung rational.



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

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

Beispiel Programmcode

```
Option Explicit

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
```

Anteilsveränderung als Bruch

Manchmal müssen Sie Anteilsveränderungen einfach darstellen. Das können Sie mit Brüchen zeigen.

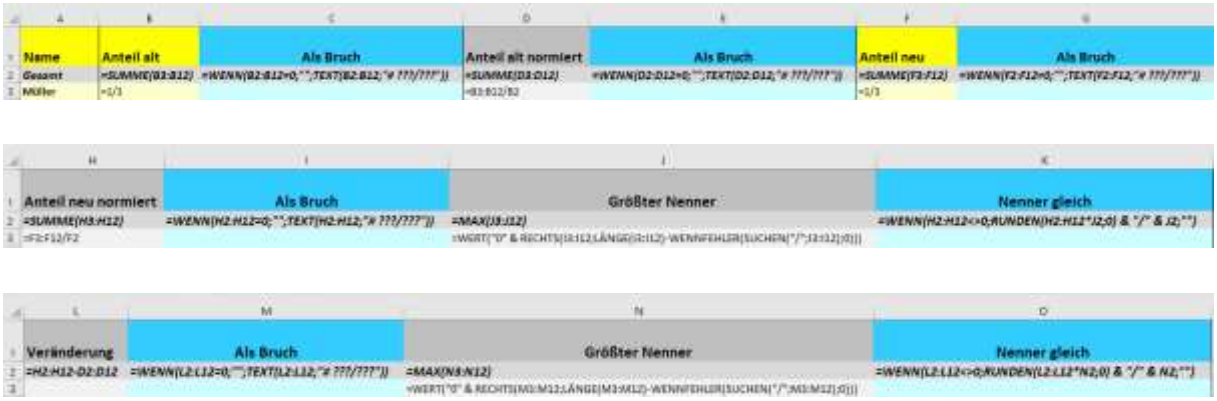
Beispiel: Eine Erbengemeinschaft besteht aus den Herren Müller, Meier und Schulz. Meier verstirbt ohne Erben, sein Anteil wird aufgeteilt. Schulz stirbt auch, sein Anteil geht zu 2/3 an seine Witwe und zu 1/3 an sein einziges Kind. Bitte beachten Sie, dass Sie diese Anteile mit =1/3 * 2/3 respektive =1/3 * 1/3 eingeben müssen, wobei das erste 1/3 den Originalanteil von Schulz darstellt.

Name	Anteil alt	Als Bruch	Anteil alt normiert	Als Bruch	Anteil neu	Als Bruch	Anteil neu normiert	Als Bruch	Größter Nenner	Nenner gleich	Veränderung	Als Bruch	Größter Nenner	Nenner gleich
Gesamt	*100,00%*	*1*	*100,00%*	*1*	*66,67%*	*2/3*	*100,00%*	*1*	*6*	*6/6*			*6*	
Müller	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
Meier	33,33%	1/3	33,33%	1/3							-33,33%	- 1/3	3	-2/6
Schulz	33,33%	1/3	33,33%	1/3							-33,33%	- 1/3	3	-2/6
Schulz Witwe					22,22%	2/9	33,33%	1/3	3	2/6	33,33%	1/3	3	2/6
Schulz Kind					11,11%	1/9	16,67%	1/6	6	1/6	16,67%	1/6	6	1/6

Hinweis: Nur in diese gefärbten Zellen Werte eingeben!

Die Formeln

Diese Aufgabe lässt sich einfach mit Excel Formeln lösen. indem Sie Excel's interne Bruchdarstellung als Textformat verwenden:



Name	Anteil alt	Als Bruch	Anteil alt normiert	Als Bruch	Anteil neu	Als Bruch
Gesamt	=SUMME(B3:B12)	=WENN(B2:B12=0;"";TEXT(B2:B12;"# ???/???"))	=SUMME(D3:D12)	=WENN(D2:D12=0;"";TEXT(D2:D12;"# ???/???"))	=SUMME(F3:F12)	=WENN(F2:F12=0;"";TEXT(F2:F12;"# ???/???"))
Müller	=1/3		=B3:B12/B2		=1/3	

Anteil neu normiert	Als Bruch	Größter Nenner	Nenner gleich
=SUMME(H3:H12)	=WENN(H2:H12=0;"";TEXT(H2:H12;"# ???/???"))	=MAX(J3:J12)	=WENN(H2:H12<>0;RUNDEN(H2:H12*J2;0) & "/" & J2;"")
=F2:F12/F2		=WERT("0" & RECHTS(J3:J12;LÄNGE(J3:J12)-WENNFEHLER(SUCHEN("/";J3:J12);0)))	

Veränderung	Als Bruch	Größter Nenner	Nenner gleich
=H2:H12-D2:D12	=WENN(L2:L12=0;"";TEXT(L2:L12;"# ???/???"))	=MAX(N3:N12)	=WENN(L2:L12<>0;RUNDEN(L2:L12*N2;0) & "/" & N2;"")
		=WERT("0" & RECHTS(M3:M12;LÄNGE(M3:M12)-WENNFEHLER(SUCHEN("/";M3:M12);0)))	

Sie können aber auch die benutzerdefinierte Funktion sbNRN verwenden.

Der Trick mit den 17 Kamelen

Ein Vater hinterließ seinen drei Söhnen 17 Kamele, sein letzter Wille besagte, dass der älteste Sohn die Hälfte der Kamele erhalten solle, der mittlere Sohn ein Drittel und der jüngste ein Neuntel. Dies war nicht einfach, aber ein weiser Mann half den Söhnen: er fügte sein Kamel

hinzu, der älteste Sohn nahm 18/2 = 9 Kamele, der zweite 18/3 = 6, der jüngste 18/9 = 2 und der weise Mann nahm sein eigenes zurück und entschwand.

Nun können wir einfach erkennen, was an dem letzten Willen des Vaters falsch war:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Name	Anteil alt	Als Bruch	Anteil alt normiert	Als Bruch	Anteil neu	Als Bruch	Anteil neu normiert	Als Bruch	Größter Nenner	Nenner gleich	Veränderung	Als Bruch	Größter Nenner	Nenner gleich
2	*Gesamt*	*94,44%*	*17/18*	*100,00%*	*1/1*	*100,00%*	*1/1*	*100,00%*	*1/1*	*18*	*18/18*			*153*	
3	Ältester Sohn	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	Zweiter Sohn	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	Jüngster Sohn	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	Weiser Mann					5,56%	1/18	5,56%	1/18	18	1/18	5,56%	1/18	18	9/153

Die Anteile der Söhne ergeben zusammen keine 100%, sondern lediglich 94,44%. Damit erhalten sie am Ende 1/34, 1/51 und 1/153 zu wenig. Der Vater hätte dem jüngsten Sohn besser ein Sechstel der Kamele vermacht, dann hätten die Anteile zusammen 100% ausgemacht. Die Anteile hätten zwar keine ganzen Kamele ergeben, aber die Söhne hätten Ausgleichszahlungen vereinbaren können.

Monatsanteil

Wenn Sie die Anzahl ganzer Monate zwischen zwei Daten benötigen, können Sie die Funktion *DATEDIF* verwenden, aber was machen Sie, wenn Sie den korrekten Anteil von Monaten und zusätzlich Tagen brauchen?

Auf diese Weise kann man es korrekt machen:

```
=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

Hinweis: Mit der Funktion *MONATSENDE* kann man die Formel abkürzen zu

```
=DATEDIF(A2;B2;"m")+WENN(TAG(B2)>=TAG(A2);(TAG(B2)-TAG(A2))/TAG(MONATSENDE(B2;0));(TAG(MONATSENDE(A2;0))-
TAG(A2))/TAG(MONATSENDE(A2;0))+TAG(B2)/TAG(MONATSENDE(B2;0)))
```

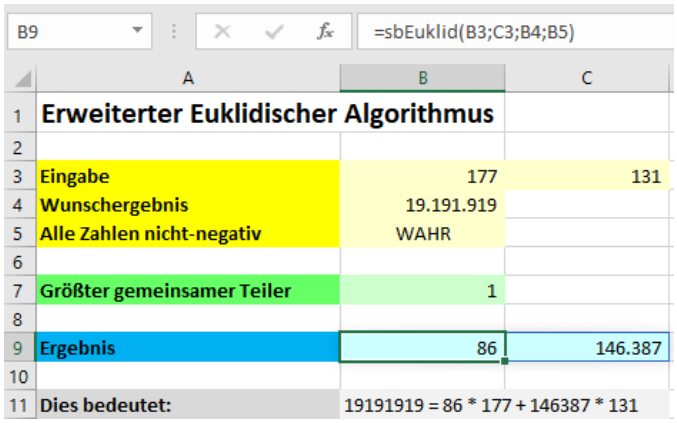
Falls Sie das Ergebnis als rationale Zahl darstellen müssen, nutzen Sie bitte die Funktion *sbNRN*.

Linearkombination Ganzer Zahlen

Erweiterter Euklidischer Algorithmus – *sbEuklid*

Sie wollen eine Zahl als nicht-negative Linearkombination zweier natürlicher Zahlen darstellen?

Dies kann man mit dem erweiterten Euklidischen Algorithmus erreichen:



	A	B	C
1	**Erweiterter Euklidischer Algorithmus**		
2			
3	**Eingabe**	177	131
4	**Wunschergebnis**	19.191.919	
5	**Alle Zahlen nicht-negativ**	WAHR	
6			
7	**Größter gemeinsamer Teiler**	1	
8			
9	**Ergebnis**	86	146.387
10			
11	**Dies bedeutet:**	19191919 = 86 * 177 + 146387 * 131	

Hinweis: Wenn das Wunschergebnis ein Vielfaches des größten gemeinsamen Teilers der eingegebenen Zahlen ist, dann existiert immer eine ganzzahlige Lösung, aber nicht notwendigerweise eine nicht-negative. Beispiel: Sie können mit den Eingaben 5 und 3 das Wunschergebnis 1 ganzzahlig darstellen, weil 1 der GGT von 5 und 3 ist: 1 = 2 * 5 + (-3) * 3. Aber es geht nicht mit ausschließlich nicht-negativen Faktoren.

Ein weiterer Hinweis: Es können mehrere nicht-negative Lösungen existieren, mit *bAllNonNegative* = WAHR gibt der u. g. Algorithmus die kleinste Summe der Ausgabewerte zurück. Wenn eine ganzzahlige Lösung existiert, dann gibt es abzählbar viele Lösungen.

sbEuklid Programmcode

Die möglichen Fehlercodes des Programms bedeuten:

- #NV! - Es existiert keine Lösung
- #WERT! - *bAllNonNegative* = WAHR aber nicht alle Eingaben sind nicht-negativ
- #ZAHL! - *bAllNonNegative* = WAHR aber es existiert keine nicht-negative Lösung

```
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
```

Uhrzeiten

Arbeitszeit zwischen 2 Zeitpunkten – *sbTimeDiff*

Name

sbTimeDiff() - Berechne die Zeit zwischen zwei Zeitpunkten, aber zähle lediglich die spezifizierten Zeiten pro Wochentag oder Feiertag minus Pausen, falls die tägliche Arbeitszeit definierte Grenzwerte überschreitet.

Synopsis

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

Beschreibung

Berechnet die Zeit zwischen zwei Zeitpunkten, aber zähle lediglich die spezifizierten Zeiten pro Wochentag oder Feiertag minus Pausen, falls die tägliche Arbeitszeit definierte Grenzwerte überschreitet.

Optionen

dtFrom - Datum und Uhrzeit ab wann zu zählen ist

dtTo - Datum und Uhrzeit bis wann zu zählen ist

vwh - 8 mal 2 Matrix, definiert die Startzeiten und Endzeiten pro Wochentag und für Feiertage, die erste Zeile für Montage, die achte für Feiertage

vHolidays - Optional. Liste der Feiertage (ganzzahlige Datumswerte). Für die Tage in dieser Liste werden nicht die Wochentagszeiten von vwh genommen, sondern die Feiertagszeiten in der achten Zeile

vBreaks - Optional. N x 2 Matrix, die die aggregierten täglichen Arbeitszeiten aufsteigend mit den zugehörigen Pausenzeiten darstellt, die zu subtrahieren sind, wenn die entsprechende Zeit an einem Tag gearbeitet wurde

Beispiel

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 Programmcode

```
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
```

Arbeitszeit zu einem Zeitpunkt addieren – *sbTimeAdd*

Name

sbTimeAdd() - Addiere eine positive Zeit zu einem Datum mit Uhrzeit, wobei lediglich spezifizierte Zeitintervalle pro Wochentag und Feiertag berücksichtigt werden und auch eine definierte tägliche Pausenzeit abgezogen wird, falls die entsprechende tägliche definierte Arbeitszeit überschritten wird.

Synopsis

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

Beschreibung

Addiere eine positive Zeit zu einem Datum mit Uhrzeit, wobei lediglich spezifizierte Zeitintervalle pro Wochentag und Feiertag berücksichtigt werden und auch eine definierte tägliche Pausenzeit abgezogen wird, falls die entsprechende tägliche definierte Arbeitszeit überschritten wird.

Optionen

dt - Datum und Uhrzeit auf die die Zeit dh addiert werden soll

dh - Zeit als Datentyp Double die auf *dt* addiert werden soll

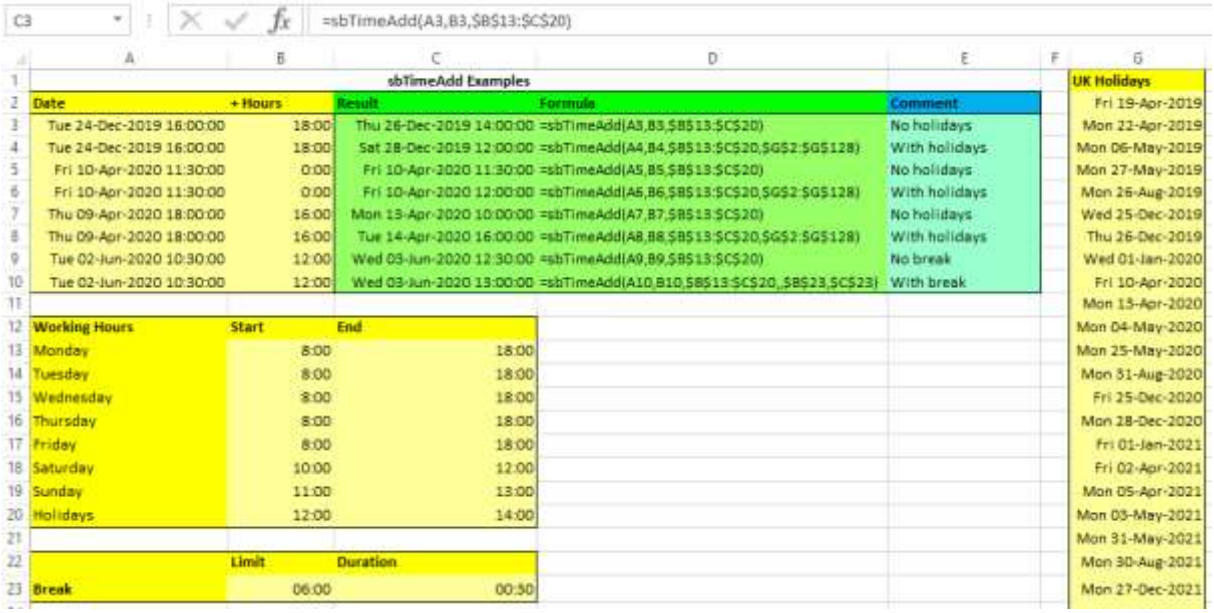
vwh - 8 mal 2 Matrix, definiert die Startzeiten und Endzeiten pro Wochentag und für Feiertage, die erste Zeile für Montage, die achte für Feiertage

vHolidays - Optional. Liste der Feiertage (ganzzahlige Datumswerte). Für die Tage in dieser Liste werden nicht die Wochentagszeiten von *vwh* genommen, sondern die Feiertagszeiten in der achten Zeile

dtBreakLimit - Optional. Tägliche Arbeitszeit ab der die Pausenzeit *dtBreakDuration* abgezogen werden muss

dtBreakDuration - Optional. Pausenzeit die von der aggregierten täglichen Arbeitszeit *dtBreakLimit* abgezogen muss, falls diese erreicht wird

<u>Beispiel</u>



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

sbTimeAdd Examples

Date	+ Hours	Result	Formula	Comment
Tue 24-Dec-2019 16:00:00	18:00	Thu 26-Dec-2019 14:00:00	=sbTimeAdd(A3,B3,$B$13:$C$20)	No holidays
Tue 24-Dec-2019 16:00:00	18:00	Sat 28-Dec-2019 12:00:00	=sbTimeAdd(A4,B4,$B$13:$C$20,$G$2:$G$128)	With holidays
Fri 10-Apr-2020 11:30:00	0:00	Fri 10-Apr-2020 11:30:00	=sbTimeAdd(A5,B5,$B$13:$C$20)	No holidays
Fri 10-Apr-2020 11:30:00	0:00	Fri 10-Apr-2020 12:00:00	=sbTimeAdd(A6,B6,$B$13:$C$20,$G$2:$G$128)	With holidays
Thu 09-Apr-2020 18:00:00	16:00	Mon 13-Apr-2020 10:00:00	=sbTimeAdd(A7,B7,$B$13:$C$20)	No holidays
Thu 09-Apr-2020 18:00:00	16:00	Tue 14-Apr-2020 16:00:00	=sbTimeAdd(A8,B8,$B$13:$C$20,$G$2:$G$128)	With holidays
Tue 02-Jun-2020 10:30:00	12:00	Wed 03-Jun-2020 12:30:00	=sbTimeAdd(A9,B9,$B$13:$C$20)	No break
Tue 02-Jun-2020 10:30:00	12:00	Wed 03-Jun-2020 13:00:00	=sbTimeAdd(A10,B10,$B$13:$C$20,,$B$23,$C$23)	With break

Working Hours	Start	End
Monday	8:00	18:00
Tuesday	8:00	18:00
Wednesday	8:00	18:00
Thursday	8:00	18:00
Friday	8:00	18:00
Saturday	10:00	12:00
Sunday	11:00	13:00
Holidays	12:00	14:00

	Limit	Duration
Break	06:00	00:30

UK Holidays
Fri 19-Apr-2019
Mon 22-Apr-2019
Mon 06-May-2019
Mon 27-May-2019
Mon 26-Aug-2019
Wed 25-Dec-2019
Thu 26-Dec-2019
Wed 01-Jan-2020
Fri 10-Apr-2020
Mon 13-Apr-2020
Mon 04-May-2020
Mon 25-May-2020
Mon 31-Aug-2020
Fri 25-Dec-2020
Mon 28-Dec-2020
Fri 01-Jan-2021
Fri 02-Apr-2021
Mon 05-Apr-2021
Mon 03-May-2021
Mon 31-May-2021
Mon 30-Aug-2021
Mon 27-Dec-2021

sbTimeAdd Programmcode

```
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
```

Uhrzeit für eine andere Zeitzone umwandeln – *ConvertTime*

Julian Hess und Patrick Honorez entwickelten ein sehr gutes Konvertierprogramm für MS Access und für MS Excel, um die Zeit einer Zeitzone in eine andere zu übersetzen:

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

Bitte beachten Sie, dass hierfür ein ordentlich installiertes MS Outlook Voraussetzung ist.

Prüfziffern

Prüfziffern sollen die manuelle Fehleingabe oder die fehlerhafte Übertragung von numerischen Daten erschweren.

Berechne oder prüfe eine Europäische Artikelnummer – *sbEAN*

Falls Sie eine Europäische Artikel Nummer (EAN) berechnen oder prüfen müssen:

	A	B	C
1	**Eingabe**	**Ausgabe**	**Kommentar**
2	1234567	0	Nur die EAN-8 Prüfziffer
3	1234567	12345670	Ganze EAN-8
4	12345670	WAHR	EAN-8 ist korrekt
5	12345678	FALSCH	EAN-8 ist falsch
6	123456789012	8	Nur die EAN-13 Prüfziffer
7	123456789012	1234567890128	Ganze EAN-13
8	1234567890128	WAHR	EAN-13 ist korrekt
9	1234567890129	FALSCH	EAN-13 ist falsch
10	12345678901234567	5	Nur die EAN-18 / NVE / SSCC Prüfziffer
11	12345678901234567	123456789012345675	Ganze EAN-18 / NVE / SSCC
12	123456789012345675	WAHR	EAN-18 ist korrekt
13	123456789012345676	FALSCH	EAN-18 ist falsch
14	9023671254823	0	Nur die EAN-14 / GTIN Prüfziffer
15	9023671254823	90236712548230	Ganze EAN-14 / GTIN
16	90236712548230	WAHR	EAN-14 / GTIN ist korrekt
17	90236712548231	FALSCH	EAN-14 / GTIN ist falsch
18	1234567890123456789	#WERT!	Falsche Eingabelänge
19	123456789O12	#ZAHL!	Ist keine Zahl (Zeichen 'O')

sbEAN Programmcode

```
Option Explicit

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
```

Summenerhaltendes Runden mit *RoundToSum* (Excel / VBA)

Abstract

Gerundete Werte ergeben zusammen nicht immer ihre gerundete Summe. Wie stelle ich sicher, dass meine Aufstellung von gerundeten Prozentzahlen genau 100% ergibt? Kann ich für meine Buchhaltung sicherstellen, dass meine Gemeinkostenverrechnung genau die originale Kostensumme verteilt? Diese Fragen sind seit langem bekannt und wurden oft analysiert.

In diesem Dokument wird eine einfach nutzbare Lösung mit Excel / VBA vorgestellt. Sie kann relative Werte (Prozentzahlen) auf 100% runden oder absolute Werte (z. B. Kostenrechnungsergebnisse) runden, ohne deren gerundete Summe zu verändern. Dabei kann je nach Parameter im Vergleich zur üblichen kaufmännischen Rundung der absolute Fehler oder der relative Fehler minimal gehalten werden.

Summenerhaltendes Runden

Beim summenerhaltenden Runden werden die Summanden so gerundet, dass deren Summe gleich der gerundeten Summe der Summanden ist. Dabei kann es notwendig sein, einen oder mehrere Summanden zum entfernteren gerundeten Wert zu runden.

Beispiel für Prozentzahlen

Die Werte 11, 45 und 555 mit der Summe 611 zeigen als Prozentsumme auf 2 Nachkommastellen gerundet nicht 100,00, sondern 99,99. Die **fett** markierten Werte innerhalb der Tabelle zeigen die von der Funktion RoundToSum veränderten gerundeten Werte:

	Werte	Prozent auf 2 Stellen gerundet	Minimiere absoluten Fehler	Minimiere relativen Fehler
	11	1,80	1,80	1,80
	45	7,36	**7,37**	7,36
	555	90,83	90,83	**90,84**
Summe	611	99,99	100,00	100,00

Die hier vorgestellte Excel / VBA Funktion würde jedoch mit dem Aufruf *RoundToSum({11;45;555};2;FALSCH)* die Ergebniswerte *{1,80;7,37;90,83}* liefern. Der Prozentwert *7,364975* wurde hier zur "falschen" Seite hin gerundet, um die Prozentsumme 100,00 zu erhalten und dabei den absoluten Fehler gegenüber der kaufmännischen Rundung zu minimieren. Mit dem Aufruf *RoundToSum({11;45;555};2;FALSCH;2)* hätten wir die Ergebniswerte *{1,80;7,36;90,84}* erhalten, weil hier der Fehlertypparameter 2 den relativen Fehler minimal gehalten hätte.

Beispiel für absolute Zahlen

Die Summe der kaufmännisch gerundeten Werte in Spalte 2 weicht um +2.000 von der gerundeten Summe ab. Die **fett** markierten Werte innerhalb der Tabelle zeigen die von der Funktion *RoundToSum* veränderten gerundeten Werte:

Werte	Absolut gerundet auf 1,000	Minimiere absoluten Fehler	Minimiere relativen Fehler
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
Summe	681,709	684,000	682,000	682,000

Die benutzerdefinierte VBA Funktion *RoundToSum*

Name

RoundToSum – Summanden runden ohne Veränderung der gerundeten Summe

Synopsis

RoundToSum(vInput; [lDigits]; [bAbsSum]; [lErrorType]; [bDontAmend])

Beschreibung

RoundToSum rundet die Summanden, ohne deren gerundete Summe zu verändern. Es verwendet die Largest Remainder Methode (auch Hare-Niemeyer Verfahren genannt), um den Fehler gegenüber der üblichen kaufmännischen Rundung zu minimieren. Falls dieser Fehler für mehrere Summanden identisch ist, wird der erste oder die ersten (von oben oder von links gesehen) Summanden angepasst.

Anmerkung: Die hier vorgestellte Lösung ist auf eindimensionale Tabellen ohne Teilsummen beschränkt. Für zweidimensionale Tabellen oder Tabellen mit Teilsummen existiert keine allgemeingültige Lösung.

Parameter

vInput Bereich oder Array, das die nicht gerundeten Eingabewerte enthält.

lDigits Optional, der Standardwert ist 2. Anzahl der Stellen, auf die gerundet werden soll. Zum Beispiel: 0 rundet auf ganze Zahlen, 2 rundet auf den Cent, -3 rundet auf Tausender.

bAbsSum Optional, der Standardwert ist WAHR. WAHR nimmt die Summanden (Eingabewerte) als unveränderte absolute Werte, was für buchhalterische Rechnungen oft notwendig ist. FALSCH verwendet die Prozentzahlen der Summanden, um genau auf die Summe 100% zu kommen. Dies wird meist für Präsentationen von prozentualen Verteilungen verwendet.

lErrorType Optional, der Standardwert ist 1. Fehlertyp, der minimal gehalten werden soll: 1 – absoluter Fehler, 2 – relativer Fehler. Den absoluten Fehler minimieren Sie üblicherweise bei zu buchenden Beträgen. Der relative Fehler wird normalerweise bei Verteilungen minimiert, um Anpassungen in flachen Außenbereichen zu vermeiden.

RoundToSum Programmcode

```
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-Ausdruck

Mithilfe dreier Lambda-Ausdrücke können wir die VBA Funktion *RandToSum* durch diesen *Round2Sum* Lambda-Ausdruck ersetzen:

```
=LAMBDA(vI;lD;bA;lE;
    LET(
        i;WENN(bA;vI;vI/SUMME(vI)%);
        r;RUNDEN(i;lD);
        _C;RUNDEN(SUMME(i);lD)-SUMME(r);
        _E;WAHL(lE;r-i;(r-i)*i);
        _R;UniqRank(_E;WENN(_C>0;1;0));
        _D;WENN(_R<=RUNDEN(ABS(_C*10^lD);0);VORZEICHEN(_C)*10^-lD;0);
        r+WENN(ZEILEN(r)=1;MTRANS(_D);_D)
    )
)
```

UniqRank wird definiert als:

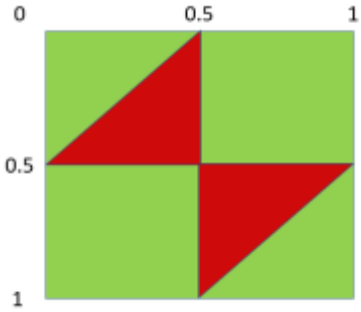
```
=LAMBDA(Ref;[Order];
    LET(
        _ord;WENN(WURDEAUSGELASSEN(Order);-1;WENN(Order=0;-1;1));
        _r;INDEX(WENN(ZEILEN(Ref)=1;MTRANS(Ref);Ref);;1);
        _c;ZEILEN(_r);
        _i;SEQUENZ(ZEILEN(_r));
        INDEX(SORTIEREN(HSTAPELN2(_i;INDEX(SORTIEREN(HSTAPELN2(_r;_i);;_ord);;2));2;1);;1)
    )
 )
```

Und – da Excel's Tabellenblattfunktion HSTAPELN nur Bereiche, aber keine Arrays annimt – *HSTAPELN2* als:

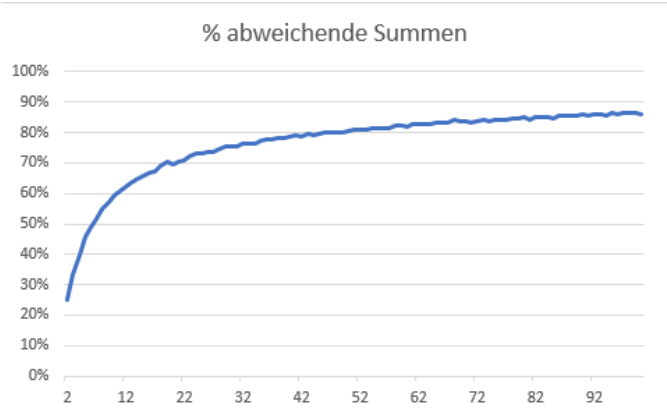
```
=LAMBDA(a;b;
    MATRIXERSTELLEN(
        ZEILEN(a);
        2;
        LAMBDA(r;c;
            WENN(c=1;INDEX(a;r);INDEX(b;r))
        )
    )
)
```

Werte runden ändert ihre Summe

Wie wahrscheinlich ist es, dass die Summe von gerundeten Werten nicht gleich ihrer gerundeten Summe ist? Für zwei zufällige Gleitkommazahlen ist dies klar: die Wahrscheinlichkeit liegt bei etwa 25% - das ist der **Rot**-Anteil an der Gesamtfläche der gezeigten Grafik:



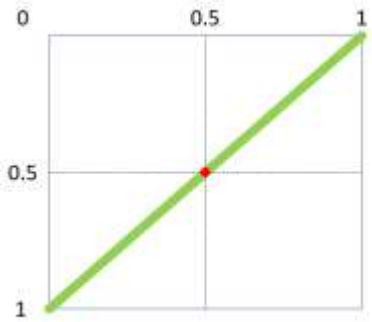
Aber etwas überraschend mag sein, dass die Wahrscheinlichkeit sich 90% nähert, je mehr Zahlen gerundet und addiert werden:



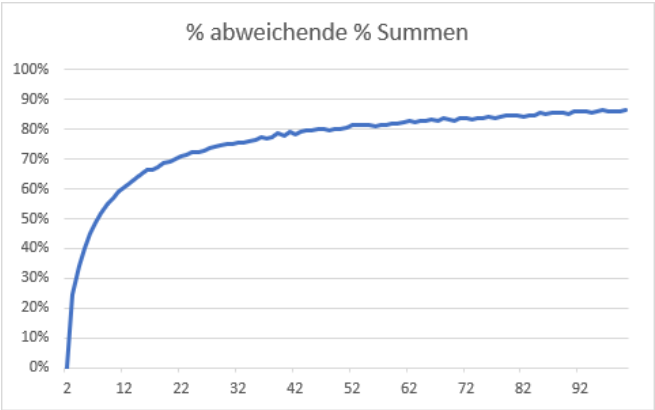
Bei sieben zufälligen Gleitkommazahlen ist die Wahrscheinlichkeit bereits größer als 50%, dass die Summe ihrer gerundeten Werten nicht gleich ihrer gerundeten Summe ist.

Gerundete Prozentanteile

Gerundete Prozentanteile ergeben addiert auch häufig nicht 100%. Bei zwei zufälligen Zahlen tritt das Problem nur auf, wenn beide Zahlen genau gleich 0,5 sind:



Aber bei mehr Zahlen stellt sich dasselbe Problem wie eingangs erwähnt, nur mit etwa einer Zahl mehr. Gerundete Prozentanteile von drei zufälligen Zahlen ergeben in etwa 25% der Fälle keine Summe von 1:



Programmcode Monte Carlo

```
Const n = 100
Const runs = 20000
Const bOnlyPositive = True 'Without loss of generality

Sub monte_carlo_add_rounded_values()
'Calculates for 2 to n how likely it is
'that rounding would not alter their sum.
'Example: for 2 numbers there is a 25% chance
'that the sum of their rounded values is not
'equal to their rounded sum.
'(C) (P) by Bernd Plumhoff 16-Dec-2023 PB V0.3
Dim i                As Long
Dim j                As Long
Dim k                As Long
Dim m                As Long
Dim d                As Double
Dim s1               As Double
Dim s2               As Double

With Application.WorksheetFunction
Randomize
For i = 2 To n
  m = 0
  For j = 1 To runs
    s1 = 0#
    s2 = 0#
    For k = 1 To i
      If bOnlyPositive Then
        d = Rnd()
      Else
        d = 2# * Rnd() - 1#
      End If
      s1 = s1 + d
      s2 = s2 + .Round(d, 0)
    Next k
    s1 = .Round(s1, 0)
    If s1 <> s2 Then
      m = m + 1
    End If
  Next j
  Cells(i, 1) = i
  Cells(i, 2) = m / runs
Next i
End With
End Sub

Sub monte_carlo_percentage_sum_of_rounded_values()
'Calculates for 2 to n how likely it is that
'rounding would not alter their percentage sum.
'Example: for 2 numbers there is a 25% chance
'that the sum of their rounded values is not
'equal to their rounded sum.
'(C) (P) by Bernd Plumhoff 16-Dec-2023 PB V0.2
Dim i                As Long
Dim j                As Long
Dim k                As Long
Dim m                As Long
Dim s1               As Double
Dim s2               As Double

With Application.WorksheetFunction
Randomize
For i = 2 To n
  m = 0
  ReDim e(1 To i) As Double
  For j = 1 To runs
    s1 = 0#
    For k = 1 To i
      If bOnlyPositive Then
        e(k) = Rnd()
      Else
        e(k) = 2# * Rnd() - 1#
      End If
      s1 = s1 + e(k)
    Next k
    s2 = 0#
    For k = 1 To i
      e(k) = .Round(1000# * e(k) / s1, 0)
      s2 = s2 + e(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
```

Anwendungsbeispiele für *RoundToSum*

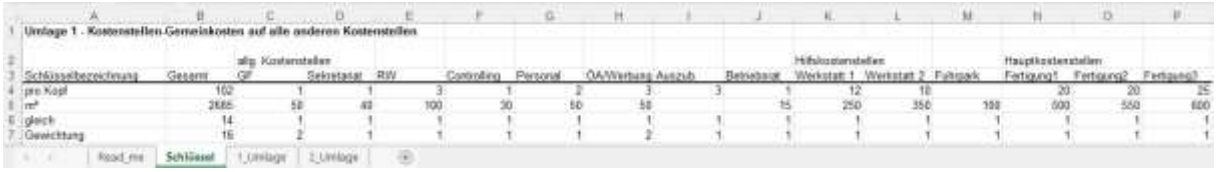
Gemeinkostenumlage

Wenn Sie Gemeinkosten auf Kostenträger verrechnen müssen, ergibt sich häufig das Problem, dass die Summe der umgelegten Kosten nicht cent-genau mit der Ursprungssumme übereinstimmt. Hier hilft die benutzerdefinierte Excel Funktion *RoundToSum*.

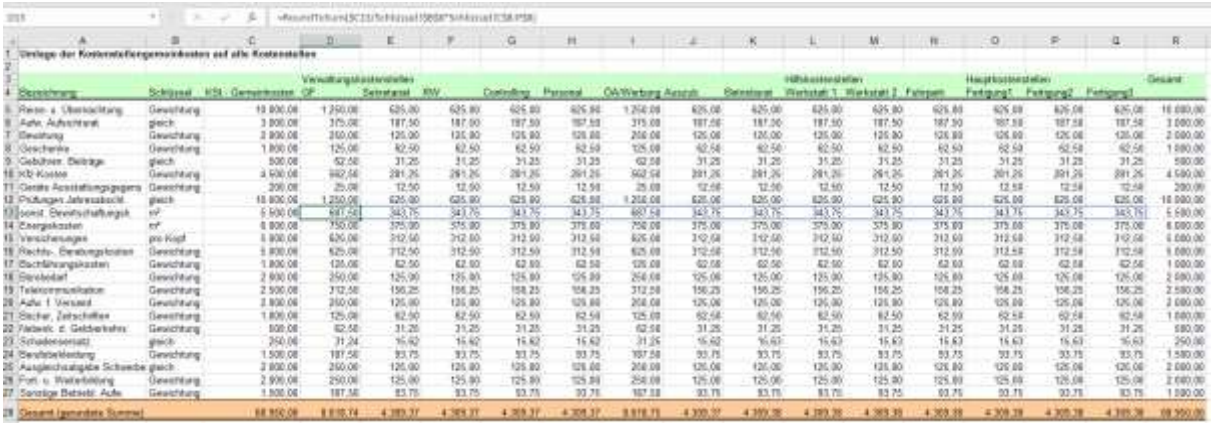
Ein praktisches Beispiel

Vorgestellt wird eine Gemeinkostenumlage, bei der sich die einzelnen Summanden cent-genau zu den jeweiligen Gesamtsummen addieren.

Zunächst legen Sie die Gemeinkosten auf alle anderen Kostenstellen um (Tabellenblatt ‚Schlüssel'):



Diese erste Gemeinkostenumlage erfolgt mit dem Aufruf einer Rundungskorrektur, damit die einzelnen Summanden cent-genau die Spaltensummen pro Kostenstelle ergeben (Tabellenblatt ‚1_Umlage':



Tabellenblattformeln		
Bereich	Formel	
D5:D27	**D5**	=RoundToSum($C5/Schlüssel!$B$6*Schlüssel!C$6:P$6)
C28:Q28	**C28**	=SUMME(C5:C27)
R5:R28	**R5**	=SUMME(D5:Q5)
R30	**R30**	=R28-C28

Die zweite Gemeinkostenumlage (Tabellenblatt ‚Schlüssel') verteilt die allgemeinen Kostenstellen und die Hilfskostenstellen mit dem Aufruf einer Rundungskorrektur centgenau auf die Hauptkostenstellen:

	A	B	C	D	E	F
10	**Umlage 2 - allgemeine Kostenstellen und Hilfskostenstellen auf Hauptkostenstellen**					
11						
12						
13	umzulegende KSt.	Schlüssel	Fertigung1	Fertigung2	Fertigung3	Gesamt
14						
15	GF	Gewichtung	30%	40%	30%	100%
16	Sekretariat	Gewichtung	40%	50%	10%	100%
17	RW	Gewichtung	30%	13%	57%	100%
18	Controlling	gleich	1	1	1	3
19	Personal	pro Kopf	20	20	25	65
20	ÖA/Werbung	Gewichtung	30%	42%	28%	100%
21	Auszub.	gleich	1	1	1	3
22	Betriebsrat	pro Kopf	20	20	25	65
23	Werkstatt 1	Gewichtung	25%	20%	55%	100%
24	Werkstatt 2	Gewichtung	20%	20%	60%	100%
25	Fuhrpark	Gewichtung	40%	30%	30%	100%

Read_me | **Schlüssel** | 1_Umlage | 2_Umlage

Das Endergebnis (Tabellenblatt ‚2_Umlage'):

	A	B	C	D	E	F	G	H
1	**Umlage der allgemeinen Kostenstellen und Hilfskostenstellen auf die Hauptkostenstellen**							
2								
3	**umzulegende KSt.**	**KST- Einzel-kosten**	**Umlage 1**	**Gesamt**	**Fertigung1**	**Fertigung2**	**Fertigung3**	**Gesamt**
4								
5	GF	111.666,00	4.924,95	116.590,95	34.977,28	46.636,38	34.977,29	116.590,95
6	Sekretariat	34.627,00	4.924,95	39.551,95	15.820,78	19.775,97	3.955,20	39.551,95
7	RW	96.834,00	4.924,97	101.758,97	30.527,69	13.228,67	58.002,61	101.758,97
8	Controlling	83.875,00	4.924,97	88.799,97	29.599,99	29.599,99	29.599,99	88.799,97
9	Personal	53.765,00	4.925,00	58.690,00	18.058,46	18.058,46	22.573,08	58.690,00
10	ÖA/Werbung	239.170,00	4.925,00	244.095,00	73.228,50	102.519,90	68.346,60	244.095,00
11	Auszub.	147.397,00	4.925,02	152.322,02	50.774,00	50.774,01	50.774,01	152.322,02
12	Betriebsrat	471,00	4.925,02	5.396,02	1.660,32	1.660,31	2.075,39	5.396,02
13	Werkstatt 1	125.225,00	4.925,02	130.150,02	32.537,51	26.030,00	71.582,51	130.150,02
14	Werkstatt 2	2.398.512,00	4.925,02	2.403.437,02	480.687,41	480.687,40	1.442.062,21	2.403.437,02
15	Fuhrpark	26.992,00	4.925,02	31.917,02	12.766,81	9.575,10	9.575,11	31.917,02
16	1. Umlage				4.925,02	4.925,02	4.925,02	14.775,06
17	2. Umlage	3.318.534,00	54.174,94	3.372.708,94	780.638,75	798.546,19	1.793.524,00	3.372.708,94
18	KSt. - Einzelkosten				738.060,00	854.000,00	650.360,00	2.242.420,00
19	Gesamt Haupkostenstellen				1.523.623,77	1.657.471,21	2.448.809,02	5.629.904,00
20								
21	Gemeinkostenzuschlagssatz				106,4%	94,1%	276,5%	151,1%
22								
23							Kontrolle	0,00

Tabellenblattformeln		
Bereich	Formel	
C5:C15	**C5**	=MTRANS('1_Umlage'!D28:N28)
D5:D15	**D5**	=SUMME(B5:C5)
E5:E15	**E5**	=RoundToSum($D5/Schlüssel!$F15*Schlüssel!C15:E15)
H5:H16	**H5**	=SUMME(E5:G5)
E16	**E16**	='1_Umlage'!O28:Q28
B17:H17	**B17**	=SUMME(B5:B15)
E19:H19	**E19**	=SUMME(E16:E18)
E21:H21	**E21**	=(E17+E16)/E18
H23	**H23**	=H19-SUMME(E18:G18)-SUMME(B5:B15)-SUMME('1_Umlage'!C5:C27)

Diese korrekte Gemeinkostenumlage können Sie in einem Buchungssystem ohne Cent-Differenzen buchen.

Beispiel für ein exaktes Verhältnis von Zufallszahlen

Es ist recht leicht, einen “unfairen” Würfel zu simulieren. Wenn wir z. B. im Mittel die 6 doppelt so häufig wie alle anderen Zahlen 1 bis 5 erhalten wollen, geben Sie in Zelle A1 ein:
*=MIN(GANZZAHL(ZUFALLSZAHL()*7+1);6)*

Aber wenn Sie einen Würfel 7 mal würfeln wollen und alle Zahlen von 1 bis 5 genau einmal und die 6 genau zweimal erscheinen soll?

Eine allgemeine Lösung:

	A	B	C	D	E	F	G	H	I	J	K	L
1				Nur statistische Wahrscheinlichkeit						Total		
2	Farbe	Häufigkeit	Pos / Iteration	Eins	Zwei	Drei	Vier	Fünf	Sechs	Grün	Gelb	Rot
3	Grün	50,00%	1	Grün	Grün	Grün	Grün	Grün	Rot	5	0	1
4	Gelb	33,33%	2	Gelb	Gelb	Grün	Grün	Gelb	Gelb	2	4	0
5	Rot	16,67%	3	Rot	Rot	Rot	Grün	Grün	Gelb	2	1	3
6			4	Gelb	Gelb	Grün	Grün	Gelb	Grün	3	3	0
7			5	Grün	Grün	Gelb	Grün	Gelb	Grün	4	2	0
8			6	Rot	Rot	Grün	Grün	Gelb	Grün	3	1	2
9			7	Gelb	Grün	Grün	Gelb	Grün	Rot	3	2	1
10			8	Grün	Gelb	Grün	Gelb	Grün	Gelb	3	3	0
11			9	Rot	Gelb	Gelb	Gelb	Gelb	Gelb	0	5	1
12			10	Grün	Grün	Grün	Gelb	Gelb	Rot	3	2	1
13									Total:	28	23	9
14									Sollte stochastisch ergeben:	30	20	10
16				Genaue Häufigkeit						Total		
17			Pos / Iteration	Eins	Zwei	Drei	Vier	Fünf	Sechs	Grün	Gelb	Rot
18			1	Grün	Grün	Gelb	Grün	Rot	Gelb	3	2	1
19			2	Gelb	Grün	Gelb	Grün	Grün	Rot	3	2	1
20			3	Grün	Grün	Rot	Gelb	Grün	Gelb	3	2	1
21			4	Rot	Grün	Grün	Grün	Gelb	Gelb	3	2	1
22			5	Gelb	Grün	Rot	Grün	Grün	Gelb	3	2	1
23			6	Rot	Grün	Grün	Grün	Gelb	Gelb	3	2	1
24			7	Grün	Grün	Gelb	Gelb	Grün	Rot	3	2	1
25			8	Grün	Grün	Rot	Gelb	Gelb	Grün	3	2	1
26			9	Gelb	Rot	Gelb	Grün	Grün	Grün	3	2	1
27			10	Grün	Rot	Grün	Gelb	Gelb	Grün	3	2	1
28									Total:	30	20	10
29									Sollte stochastisch ergeben:	30	20	10

Tabellenblattformeln		
Bereich	Formel	
D3:I12	**D3**	=INDEX($A$3:$A$5;GANZZAHL(sbRandHistogrm(1;4;$B$3:$B$5)))
J3:L12;J18:L27	**J3**	=ZÄHLENWENN($D3:$I3;J$2)
J13:L13;J28:L28	**J13**	=SUMME(J3:J12)
J14;J29	**J14**	=ANZAHL2($D$3:$I$12)*MTRANS($B$3:$B$5)
D18:D27	**D18**	=INDEX($A$3:$A$5;GANZZAHL(sbExactRandHistogrm(6;1;4;$B$3:$B$5)))

Die benutzerdefinierte VBA Funktion sbExactRandHistogrm

Name

sbExactRandHistogrm – Erzeuge eine exakte Histogramm-Verteilung des Datentyps Double.

Synopsis

sbExactRandHistogrm(ldraw; dmin; dmax; vWeight)

Beschreibung

sbExactRandHistogrm erzeugt eine exakte Histogramm-Verteilung für ldraw Ziehungen von Gleitkommazahlen mit doppelter Genauigkeit im Intervall dmin:dmax. Dieses Intervall ist in vWeight.count Klassen unterteilt. Jede Klasse besitzt das Gewicht vWeight(i), welches die Wahrscheinlichkeit des Erscheinens eines Wertes innerhalb dieser Klasse darstellt. Wenn die Gewichte nicht genau für ldraw Ziehungen erreicht werden können, dann wird das Hare-Niemeyer-Verfahren (“Quotenverfahren mit Restausgleich nach größten Bruchteilen”) angewandt, um den absoluten Fehler zu minimieren. Diese Funktion ruft *RoundToSum* auf.

Parameter

ldraw Anzahl der Ziehungen

dmin Minimum = Untergrenze für die zu ziehenden Zahlen

dmax Maximum = Obergrenze für die zu ziehenden Zahlen

vWeight Array von Gewichten. Die Array Größe bestimmt die Anzahl der Klassen, in die das Intervall dmin : dmax aufgeteilt wird. Die Array-Werte bestimmen die Wahrscheinlichkeit, mit der Werte innerhalb dieser Klasse gezogen werden.

sbRandHistogrm Programmcode

```
Function sbExactRandHistogrm(ldraw As Long, _
            dmin As Double, _
            dmax As Double, _
            vWeight As Variant) As Variant
'Creates an exact histogram distribution for ldraw draws within range dmin:dmax.
'This range is divided into vWeight.count classes. Each class has weight vWeight(i)
'reflecting the probability of occurrence of a value within the class.
'If weights can't be achieved exactly for ldraw draws the largest remainder method will
'be applied to minimize the absolute error. This function calls (needs) RoundToSum.
'(C) (P) by Bernd Plumhoff 01-May-2021 PB V0.9

Dim i As Long, j As Long, n As Long
Dim vW As Variant
Dim dSumWeight As Double, dR As Double

Randomize
With Application.WorksheetFunction
vW = .Transpose(vWeight)
On Error GoTo Errhdl
i = vW(1) 'Throw error in case of horizontal array
On Error GoTo 0

n = UBound(vW)
ReDim dWeight(1 To n) As Double
ReDim dSumWeightI(0 To n) As Double
ReDim vR(1 To ldraw) As Variant

For i = 1 To n
    If vW(i) < 0# Then 'A negative weight is an error
        sbExactRandHistogrm = CVErr(xlErrValue)
        Exit Function
    End If
    'Calculate sum of all weights
    dSumWeight = dSumWeight + vW(i)
Next i

If dSumWeight = 0# Then
    'Sum of weights has to be greater zero
    sbExactRandHistogrm = CVErr(xlErrValue)
    Exit Function
End If

For i = 1 To n
    'Align weights to number of draws
    dWeight(i) = CDbl(ldraw) * vW(i) / dSumWeight
Next i

vW = RoundToSum(dWeight, 0)
On Error GoTo Errhdl
i = vW(1) 'Throw error in case of horizontal array
On Error GoTo 0

For j = 1 To ldraw

    dSumWeight = 0#
    dSumWeightI(0) = 0#
    For i = 1 To n
        'Calculate sum of all weights
        dSumWeight = dSumWeight + vW(i)
        'Calculate sum of weights till i
        dSumWeightI(i) = dSumWeight
    Next i

    dR = dSumWeight * Rnd

    i = n
    Do While dR < dSumWeightI(i)
        i = i - 1
    Loop

    vR(j) = dmin + (dmax - dmin) * (CDbl(i) + _
            (dR - dSumWeightI(i)) / vW(i + 1)) / CDbl(n)
    vW(i + 1) = vW(i + 1) - 1#

Next j

sbExactRandHistogrm = vR

Exit Function

Errhdl:
'Transpose variants to be able to address
'them with vW(i), not vW(i,1)
vW = .Transpose(vW)
Resume Next
End With

End Function
```

Faire Mitarbeiterauswahl nach Teamgröße

Nehmen Sie an, in Ihrem Unternehmen müssen Sonderaufgaben durchgeführt werden, die von jedem Mitarbeiter getätigt werden können. Sie wollen alle Teams fairerweise auf Basis ihrer Mitarbeiteranzahl belasten. Für diese Auswahl bietet sich die benutzerdefinierte Excel Funktion *RoundToSum* an. Da nicht davon ausgegangen werden kann, dass für jede Sonderaufgabe jedes Team prozentual zu seiner Teamgröße belastet werden kann, sollte der Aufruf von *RoundToSum* eine Rückschau über vergangene Mitarbeiterabstellungen enthalten.

RoundToSum verwendet das Hare-Niemeyer Verfahren, welches das Mandatszuwachsparadoxon aufweist. Wenn ein Mitarbeiter mehr ausgewählt wird, kann es vorkommen, dass ein Team weniger Mitarbeiter als zuvor abstellen muss. Da dies nicht rückwirkend geschehen kann, muss dieser Effekt ausgeglichen werden, sobald er auftritt.

Beispiel

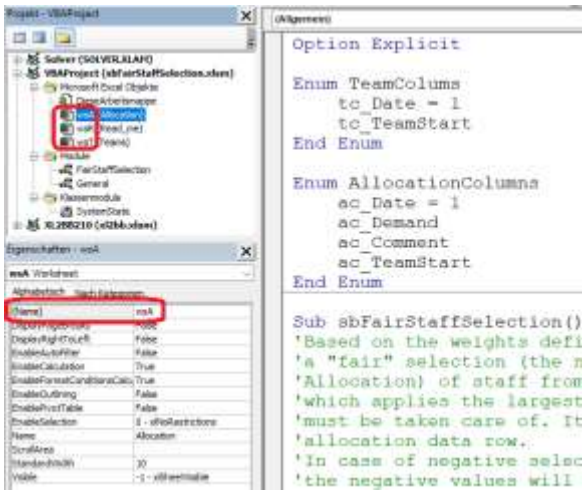
Am 1. Januar 2023 existieren die folgenden Teams (Tabellenblatt ‚Teams', VBA Name ‚wsT'):

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

Über drei Monate hinweg werden die folgenden Mitarbeiter für Sonderaufgaben benötigt und ausgewählt (Tabellenblatt ‚Allocation', VBA Name ‚wsA'):

	A	B	C	D	E	F	G
		Calculate Allocation					
1	**Date**	**Demand**	**Comment**	**Team A**	**Team B**	**Team C**	**Team D**
2	01.01.2023	323		183	124	14	2
3	01.02.2023	1	Recalc 11.03.2023 10:52:24. Allocation for 1 amended to 0. Allocation for 3 set to 0.	0	1	0	0
4	01.03.2023	9676	Recalc 11.03.2023 10:52:24.	5487	3725	406	58

Am 1. Februar 2023 hätte das Hare-Niemeyer Verfahren insgesamt 184, 125, 13 und 2 Mitarbeiter für die Teams A, B, C und D ausgewählt. Da aber am 1. Januar Team C bereits 14 Mitarbeiter bereitgestellt hatte und keine rückwirkenden Anpassungen möglich sind, muss Team A oder B einen Mitarbeiter weniger geben. Der implementierte Algorithmus schaut von links nach rechts, ob angepasst werden kann, also trifft es hier Team A. Am 1. März 2023 werden genau alle restlichen Mitarbeiter aller Teams benötigt. Der Algorithmus wählt insgesamt für jedes Team genau die Gesamtzahl der Mitarbeiter aus, weil die Rückschau alle Anforderungs-Datensätze umfasst.



Anmerkung: Der VBA Name eines Tabellenblattes kann direkt mit VBA angesprochen werden. Er kann von dem Tabellenblattnanmen abweichen, den der Anwender sieht. Leider kann man ihn nur manuell ändern, nicht via VBA.

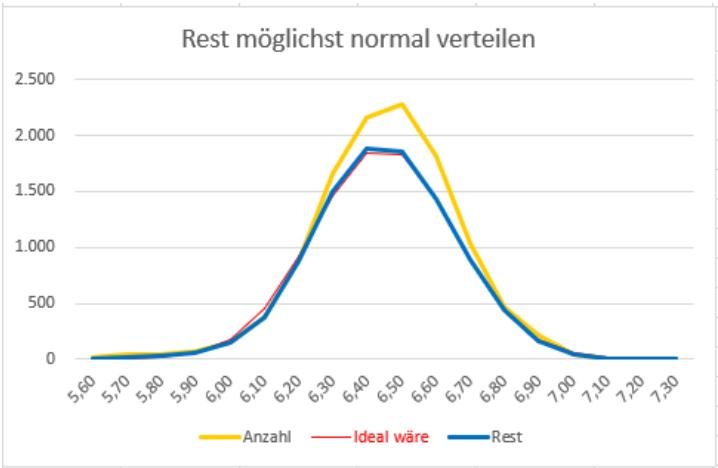
sbFairStaffSelection Programmcode

```
Enum TeamColums
    tc_Date = 1
    tc_TeamStart
End Enum

Enum AllocationColumns
    ac_Date = 1
    ac_Demand
    ac_Comment
    ac_TeamStart
End Enum

Sub sbFairStaffSelection()
'Based on the weights defined in tab Teams this program allocates
'a "fair" selection (the number given in column Demand of tab
'Allocation) of staff from these teams. This program uses (calls) RoundToSum
'which applies the largest remainder method, so the Alabama paradoxon
'must be taken care of. It also applies a lookback up to the topmost
'allocation data row.
'In case of negative selection counts (i. e. the Alabama paradoxon)
'the negative values will be set to zero and the necessary amendments
'(reductions) will be applied from left to right. Please order your
'teams with ascending sizes or descending sizes to account for this.
'(C) (P) by Bernd Plumhoff 09-Mar-2023 PB V0.1

Dim bLookBack               As Boolean
Dim bReCalc                 As Boolean

Dim i                       As Long
Dim j                       As Long
Dim k                       As Long
Dim m                       As Long
Dim lAmend                  As Long
Dim lCellResult             As Long
Dim lDemand                 As Long
Dim lRowSum                 As Long
Dim lSum                    As Long
Dim lTotal                  As Long 'Most recent total number of staff in all teams

Dim sComment                As String

Dim vAlloc                  As Variant
Dim vTeams                  As Variant

Dim state                   As SystemState

Set state = New SystemState

With Application.WorksheetFunction

vTeams = .Transpose(.Transpose(Range(wsT.Cells(1, 1).End(xlDown).Offset(0, tc_TeamStart - 1), _
          wsT.Cells(1, 1).End(xlDown).End(xlToRight))))
j = UBound(vTeams)
ReDim dAlloc(1 To j) As Double
lTotal = .Sum(vTeams)

bReCalc = False
i = 2
lDemand = wsA.Cells(i, ac_Demand)
Do While lDemand > 0

    lRowSum = .Sum(Range(wsA.Cells(i, ac_TeamStart), wsA.Cells(i, ac_TeamStart + j)))

    If lDemand <> lRowSum Then bReCalc = True

    If bReCalc Or wsA.Cells(i + 1, ac_Demand) = 0 Then

        sComment = "Recalc " & Format(Now(), "DD.MM.YYYY HH:nn:ss") & ". "
        bLookBack = False
        k = i - 1
        If k > 1 Then
            bLookBack = True
            lDemand = 0
            lSum = 0
            ReDim lTeamSum(1 To j) As Long
            Do While k > 1
                lSum = lSum + wsA.Cells(k, ac_Demand)
                lDemand = wsA.Cells(i, ac_Demand) + lSum
                For m = 1 To j
                    lTeamSum(m) = lTeamSum(m) + wsA.Cells(k, m + ac_TeamStart - 1)
                Next m
                'If lSum >= lTotal Then Exit Do 'Uncomment if lookback should be restricted
                                               'to total staff number
                k = k - 1
            Loop
        End If

        For m = 1 To j
            dAlloc(m) = lDemand * vTeams(m) / lTotal
        Next m

        vAlloc = RoundToSum(vInput:=dAlloc, lDigits:=0)

        If bLookBack Then
            For m = 1 To j
                lCellResult = vAlloc(m) - lTeamSum(m)
                If lCellResult < 0 Then
                    'The Alabama Paradoxon: we have to reduce other parties'
                    'allocations because we cannot have negative allocations
                    lAmend = lAmend - lCellResult
```

```
                End If
                vAlloc(m) = lCellResult
            Next m
            If lAmend > 0 Then
                For m = 1 To j
                    lCellResult = vAlloc(m)
                    If lCellResult < 0 Then
                        vAlloc(m) = 0
                        sComment = sComment & "Allocation for " & m & " set to 0. "
                    ElseIf lCellResult > 0 And lAmend > 0 Then
                        If lCellResult > lAmend Then
                            vAlloc(m) = lCellResult - lAmend
                            lAmend = 0
                        Else
                            vAlloc(m) = 0
                            lAmend = lAmend - lCellResult
                        End If
                        sComment = sComment & "Allocation for " & m & " amended to " & _
                                   vAlloc(m) & ". "
                    End If
                Next m
            End If
        End If
        wsA.Cells(i, ac_Comment) = sComment
        For m = 1 To j
            wsA.Cells(i, ac_TeamStart + m - 1) = vAlloc(m)
        Next m

    End If

    i = i + 1
    lDemand = wsA.Cells(i, ac_Demand)

Loop

Range(wsT.Cells(1, tc_TeamStart), wsT.Cells(1, 250)).Copy Destination:=wsA.Cells(1, ac_TeamStart)

End With

End Sub
```

Stichprobe normalverteilen

Sie haben 11.256 Weihnachtsbäume. Ein Kunde möchte Ihnen 1.500 davon abkaufen. Die einzige Bedingung: die durchschnittliche Länge soll 6,50 Meter betragen. Sie würden nun gern die Restmenge Ihrer Weihnachtsbäume möglichst normalverteilt haben:



Wie können Sie dies erreichen?

Eine Beispielrechnung

	A	B	C	D	E	F	G	H
1		Vorgabe Gesamtzahl Entnahme:				1500		
2		Vorgabe Mittelwert Länge:				6,5		
3	**Länge**	**Anzahl**	Ideal wäre	Auto-Entnahme	Zwischen-ergebnis	**Entnahme**	**Rest**	Ideal wäre
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**	**1664**	**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

Tabellenblattformeln		
Bereich	Formel	
C4	**C4**	=RoundToSUm(NORM.VERT(A4:A21;B24;B25;FALSCH)*B22/10;0;;2)
D4	**D4**	=WENN(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.VERT(A4:A21;(B24*B22-F2*F1)/(B22-F1);B25;FALSCH)*(B22-F1)/10;0;;2)
B22:H22	**B22**	=SUMME(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)

Angenommen, Sie haben die oben gezeigte Menge an Bäumen mit den angegebenen Längen. Eine erste interessante Rechnung ist sicherlich, inwieweit Ihre Ausgangsmenge bereits normalverteilt ist. Die Schiefe ermitteln wir mittels der Funktion sbSWV: sie beträgt gerundet *=sbSWV("SKEW.P";$A$4:$A$21;B$4:B$21) = -0,35*. Der Exzess (Maß für die Wölbung) beträgt gerundet *=sbSWV("KURT";$A$4:$A$21;B$4:B$21) = 0,95*. Wie wir am oben gezeigten Diagramm am gelb-orangen Graphen sehen können, ist diese Ausgangsmenge bereits "einigermaßen" normalverteilt.

Idealerweise wäre diese Stichprobe allerdings verteilt wie in Spalte C gezeigt mit der Formel *=MTRANS(RoundToSum(NORM.VERT(A4:A21;B24;B25;FALSCH)*B22/10;0))*: die Schiefe und der Exzess wären gleich Null (die vorgenommenen Rundungen führen hier zu leichten Abweichungen). Spalte H zeigt die ideale Restmengenverteilung nach Entnahme.

Mit der in Spalte D gezeigten automatischen Entnahme versuchen wir, diese ideale Restmenge möglichst zu erreichen. Dies kann natürlich nur gelingen, wenn wir ausreichend viele Bäume in den jeweiligen Längen zur Verfügung haben. Wo dies nicht der Fall ist können wir leider keine Bäume hinzufügen und müssen als Entnahmemenge 0 eintragen - im Diagramm sehen Sie z. B., dass die ideale Verteilung bei Länge 6,10 m höher ist als die tatsächliche Restverteilung.
Die ursprünglichen Formeln in Spalte F sollten =D4 bis =D21 lauten.

Diese überschreiben wir nun mit manuellen Werten, um

- auf eine Gesamtentnahme von genau 1.500 Bäumen,
- auf einen Mittelwert von 6,5 Länge der Bäume,
- auf etwa die gleiche Standardabweichung (Streuung) der Restmenge wie der Originalmenge,
- auf eine betragsmäßig kleinere Schiefe als bei der Originalmenge
- und auf einen betragsmäßig kleineren Exzess als bei der Originalmenge

zu kommen.

In der unten angebotenen Beispieldatei werden erhöhte Abweichungen durch bedingte Formatierungen angezeigt.

Anmerkung: Es ist nicht immer möglich, eine “hinreichend” normal verteilte Restmenge zu erhalten. Wie man einfach sehen kann, ist manchmal nicht einmal eine gewünschte Durchschnittslänge zu erreichen - fragen Sie bei der oben gezeigten Menge z. B. nach 21 Bäumen mit der Durchschnittslänge 5,60 m.

Hilfsfunktionen

Excel verfügt zwar über viele statistische Grundfunktionen. Diese sind jedoch nicht in der Lage, gewichtete Werte zu verarbeiten. Die hier verwendete benutzerdefinierte Funktion *sbSWV* (engl. statistics for weighted values) ermöglicht eine einfache und schnelle Anzeige, wie gut die Stichproben normalverteilt sind.

Damit die Summen der ganzzahligen Idealverteilungen der Stichproben genau mit den Summen der originalen Stichproben übereinstimmen, wurde die benutzerdefinierte Funktion *RoundToSum* verwendet. Man beachte, dass hierbei der Parameter 2 für den Fehlertyp zur Minimierung des relativen Fehlers gewählt wurde. Dies verhindert künstliche Rundungen zur “falschen” Seite in den Außenbereichen der Verteilungen.

sbSWV Programmcode

```
#Const SORTED = False

Function sbSWV(sStat As String, _
        ParamArray vInput() As Variant) As Variant
'Calculate some statistical measures of weighted values
'(C) (P) by Bernd Plumhoff 20-Aug-2024 PB V0.81
Dim d As Double, d2 As Double, dSum As Double
Dim i As Long, j As Long, k As Long, m As Long, n As Long
Dim vV, vV2, vV3, vW 'Variants

With Application.WorksheetFunction
vV = .Transpose(vInput(0))
Select Case sStat
Case "COVAR", "CORREL"
    vV2 = .Transpose(vInput(1))
    vW = .Transpose(vInput(2))
Case Else
    vW = .Transpose(vInput(1))
End Select
On Error GoTo errhdl
i = vV(1) 'Force error in case of vertical arrays
On Error GoTo 0
If UBound(vV) <> UBound(vW) Then
    'Arrays of values and of weights must have same dimension
    sbSWV = CVErr(xlErrNum)
    Exit Function
End If
Select Case UCase(sStat)
Case "AVERAGE"
    sbSWV = .SumProduct(vV, vW) / .Sum(vW)
Case "CORREL"
    vV3 = vV
    dSum = .Sum(vW)
    d = .SumProduct(vV, vW) / dSum
    d2 = .SumProduct(vV2, vW) / dSum
    For i = LBound(vV) To UBound(vV)
        vV3(i) = vW(i) * (vV(i) - d) * (vV2(i) - d2)
        vV(i) = vW(i) * (vV(i) - d) ^ 2#
        vV2(i) = vW(i) * (vV2(i) - d2) ^ 2#
    Next i
    sbSWV = .Sum(vV3) / Sqr(.Sum(vV) * .Sum(vV2))
Case "COVAR"
    dSum = .Sum(vW)
    d = .SumProduct(vV, vW) / dSum
    d2 = .SumProduct(vV2, vW) / dSum
    For i = LBound(vV) To UBound(vV)
        vV(i) = vW(i) * (vV(i) - d) * (vV2(i) - d2)
    Next i
    sbSWV = .Sum(vV) / dSum
Case "KURT"
    n = .Sum(vW)
    ReDim dV(1 To n) As Double
    k = 1
    For i = 1 To UBound(vW)
      For j = 1 To vW(i)
        dV(k) = vV(i)
        k = k + 1
      Next j
    Next i
    sbSWV = .Kurt(dV)
Case "MODE"
    k = .Max(vW)
    If k < 2 Then
        sbSWV = CVErr(xlErrNA)
        Exit Function
    End If
    sbSWV = vV(.Match(.Max(vW), vW, False))
Case "MEDIAN"
    If .Min(vW) < 1 Then
        sbSWV = CVErr(xlErrNA)
        Exit Function
    End If
    k = 0
    j = .Sum(vW)
    m = j Mod 2
    For i = LBound(vW) To UBound(vW)
        If vW(i) Mod 1 <> 0 Then
            sbSWV = CVErr(xlErrNum)
            Exit Function
        End If
        #If Not SORTED Then
            'Ensure ascending values in case input is unsorted.
            'This simple bubble sort leads to a quadratic runtime
            'but it's still quicker on 50 input values or more than
            'Lorimer Miller's nifty worksheet function approach
            '=LOOKUP(2,1/FREQUENCY(SUM(B1:B50)/2,SUMIF(A1:A50,"<="&A1:A50,B1:B50)),A1:A50)
            'BTW: Lorimer's approach is different from Excel's MEDIAN
            '(see below); and his other elegant array formula
            '=MEDIAN(IF(TRANSPOSE(ROW(A1:A1000))<=B1:B50,A1:A50))
            'calculates like Excel's MEDIAN but IMHO it's way too slow
            For n = i + 1 To UBound(vW)
                If vV(n) < vV(i) Then
                    d = vV(i)
                    vV(i) = vV(n)
                    vV(n) = d
                    d = vW(i)
                    vW(i) = vW(n)
                    vW(n) = d
                End If
            Next n
```

```
        #End If
        k = k + vW(i)
        Select Case 2 * k
        Case j + m
            If m = 0 Then
                #If Not SORTED Then
                    'Ensure vV(i + 1) is next greater value
                    For n = i + 2 To UBound(vW)
                        If vV(n) < vV(i + 1) Then
                            vV(i + 1) = vV(n)
                        End If
                    Next n
                #End If
                'Here Lorimer's function mentioned above would
                'return vV(i), the lower value
                sbSWV = (vV(i) + vV(i + 1)) / 2#
            Else
                sbSWV = vV(i)
            End If
            Exit Function
        Case Is > j + m
            sbSWV = vV(i)
            Exit Function
        End Select
    Next i
Case "SKEW.P"
    n = .Sum(vW)
    ReDim dV(1 To n) As Double
    k = 1
    For i = 1 To UBound(vW)
      For j = 1 To vW(i)
        dV(k) = vV(i)
        k = k + 1
      Next j
    Next i
    sbSWV = .Skew_p(dV)
Case "STDEV"
    dSum = .Sum(vW)
    d = .SumProduct(vV, vW) / dSum
    For i = LBound(vV) To UBound(vV)
        vV(i) = Abs(vV(i) - d) ^ 2#
    Next i
    sbSWV = Sqr(.SumProduct(vV, vW) / (dSum - 1#))
Case "STDEV.P"
    dSum = .Sum(vW)
    d = .SumProduct(vV, vW) / dSum
    For i = LBound(vV) To UBound(vV)
        vV(i) = Abs(vV(i) - d) ^ 2#
    Next i
    sbSWV = Sqr(.SumProduct(vV, vW) / dSum)
Case "VAR"
    dSum = .Sum(vW)
    d = .SumProduct(vV, vW) / dSum
    For i = LBound(vV) To UBound(vV)
        vV(i) = vW(i) * (vV(i) - d) ^ 2#
    Next i
    sbSWV = .Sum(vV) / (dSum - 1#)
Case Else
    sbSWV = CVErr(xlErrValue)
End Select
Exit Function
errhdl:
'Transpose variants to be able to address them
'with vV(i), not vV(i,1)
vV = .Transpose(vV)
vW = .Transpose(vW)
Select Case sStat
Case "COVAR", "CORREL"
    vV2 = .Transpose(vV2)
End Select
Resume Next
End With
End Function
```

Verteilung nach Restmenge

Die Budgets ausscheidender Mitarbeiter werden auf die verbliebenen gemäß deren bisherigen Budgets verteilt. Wie können Sie dies korrekt durchführen?

Ein einfacher Ansatz

Eine einfache Formel ist *=RUNDEN(C3*$B$2/$C$2;2)*, die Sie von D3 nach D12 hinunterkopieren können.

Sie können die Budgets der ausscheidenden Mitarbeiter einfach in Spalte C löschen. Die Reihenfolge der Löschungen ist egal. Der offensichtliche Nachteil besteht in einem möglichen Rundungsfehler, weil die Summe gerundeter Summanden nicht notwendig der gerundeten Summe der nicht gerundeten Summanden entspricht. Dieses Beispiel zeigt eine Differenz von 0,02.

	A	B	C	D
1	**Name**	**Betrag**	**Löschung**	**Neuer Betrag**
2	**Summe**	**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

Tabellenblattformeln		
Bereich	Formel	
B2:D2	**B2**	=SUMME(B3:B12)
D3	**D3**	=RUNDEN(C3:C12*$B$2/$C$2;2)

Eine korrekte Rechnung

Mit der benutzerdefinierten Funktion *RoundToSum* können Sie die Spillformel *=RoundToSum(C4:C13*$B$3/$C$3;D1)* verwenden.

RoundToSum muss manchmal in die ‘falsche’ Richtung runden, aber dann wird dies wenigstens mit dem kleinstmöglichen Fehler getan.

	A	B	C	D
1	Runden auf Nachkommastellen:			-1
2	**Name**	**Betrag**	**Löschung**	**Neuer Betrag**
3	**Summe**	**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

Tabellenblattformeln		
Bereich	Formel	
B3:D3	**D3**	=SUMME(D4:D13)
D4	**D4**	=RoundToSum(C4:C13*$B$3/$C$3;D1)

Urlaub nehmen wenn weniger los ist

Wenn Ihr Geschäft saisonal stark schwankt, können Sie den Urlaub Ihrer Belegschaft entsprechend planen und ggf. Saisonarbeitskräfte anstellen:

Hinweis: Selbstverständlich kann man keinem Mitarbeiter vorschreiben, wieviel Urlaub wann genommen werden muss. Diese Rechnungen sind lediglich Vorschläge, die als vernünftige Indikatoren dienen sollen.

Einfaches Beispiel

Wenn sie den maximalen Umsatzmonat (hier: 24.000) als Basis nehmen wollen, in dem kein Urlaub genommen werden sollte, und die restlichen Urlaubstage linear gemäß der Umsätze verteilen wollen:

	A	B	C	D	E	F	G	H
1			**Umsatzgrenze (kein Urlaub):**		**Höhere Umsatzgrenze (etwas Urlaub):**	Erhöht um auch Urlaub bei Umsatzmaximum zuzulassen.		
2			24.000		28.000			
3		**Umsatz**	**Urlaub (streng)**	**Ganzzahlig**	**Urlaub (moderat)**	**Ganzzahlig**		
4	**Total**	**230.000**	83		83			
5	**Januar**	20.000	5,7	6	6,3	6		
6	**Februar**	24.000	-	-	3,1	3		
7	**März**	23.000	1,4	1	3,9	4		
8	**April**	20.000	5,7	6	6,3	6		
9	**Mai**	19.000	7,2	7	7,0	7		
10	**Juni**	15.000	12,9	13	10,2	10		
11	**Juli**	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	**Oktober**	20.000	5,7	6	6,3	6		
15	**November**	19.000	7,2	7	7,0	7		
16	**Dezember**	18.000	8,6	9	7,8	8		
17		**Prüfsumme**	**83,0**	**83**	**83,0**	**83**		

Tabellenblattformeln		
Bereich	Formel	
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**	=SUMME(C5:C16)

Komplexeres Beispiel

Hier werden alle Mitarbeiter einzeln berücksichtigt, auch hinsichtlich ihrer Anwesenheitsmonate.
In der letzten Tabelle kommt *RoundToSum* zum Einsatz, um summenerhaltend auf ganze Urlaubstage zu runden:

	A	B	C	D	E	F	G	H
1			Umsatzgrenze (kein Urlaub):		Höhere Werte erlauben auch Urlaub bei Maximalumsatz			
2			24.000					
3		Umsatz	Urlaub	Urlaub (ganze Tage)	Anwesenheit [x] und Urlaubsanspruch			
4	Total	230.000			Andrew	Benjamin	Charlie	David
5	Januar	20.000	5,7	6	x	x		x
6	Februar	24.000	-	-	x	x		x
7	März	23.000	1,4	1	x	x	x	x
8	April	20.000	5,7	6	x	x	x	x
9	Mai	19.000	7,2	7	x	x	x	
10	Juni	15.000	12,9	13	x	x	x	
11	Juli	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	Oktober	20.000	5,7	6	x	x	x	x
15	November	19.000	7,2	7	x		x	x
16	Dezember	18.000	8,6	9	x		x	x
17		Total	83,0	83	25,0	21,0	21,0	16,0
18								
19				Urlaub				
20				Total	Andrew	Benjamin	Charlie	David
21			Januar	6,1	1,8	1,9	-	2,5
22			Februar	-	-	-	-	-
23			März	1,3	0,3	0,3	0,3	0,4
24			April	7,8	1,8	1,9	1,6	2,5
25			Mai	6,2	2,1	2,2	1,9	-
26			Juni	11,5	3,9	4,1	3,5	-
27			Juli	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			Oktober	7,8	1,8	1,9	1,6	2,5
31			November	6,9	2,1	-	1,9	2,9
32			Dezember	8,9	2,7	-	2,5	3,7
33			Total	83,0	25,0	21,0	21,0	16,0
34								
35				Urlaub (ganze Tage)				
36				Total	Andrew	Benjamin	Charlie	David
37			Januar	7	2	2	-	3
38			Februar	-	-	-	-	-
39			März	-	-	-	-	-
40			April	8	2	2	2	2
41			Mai	6	2	2	2	-
42			Juni	11	4	4	3	-
43			Juli	13	4	5	4	-
44			August	9	3	3	3	-
45			September	5	1	1	1	2
46			Oktober	8	2	2	2	2
47			November	7	2	-	2	3
48			Dezember	9	3	-	2	4
49			Total	83	25	21	21	16

Tabellenblattformeln		
Bereich	Formel	
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)

Zuweisen von Arbeitseinheiten vermindert um geleistete

Wie können Sie Ihren Mitarbeitern Arbeitseinheiten fair zuweisen, wenn Sie bereits geleistete Arbeit berücksichtigen wollen?

Gelbe Zellen sind Eingabezellen, grüne zeigen Zwischenergebnisse, und blaue kennzeichnen endgültige Ergebnisse. Hinweis: Sie müssen ‚Units done' in absteigender Reihenfolge eingeben.

In diesem Beispiel wurden bereits 90,6 Einheiten geliefert, aber 86 weitere Einheiten müssen noch 28 Lehrern zugewiesen werden. Ein fairer Anteil wäre für jeden Lehrer *(90.6 + 86) / 28 = 6,3*. Aber 7 Lehrer haben bereits mehr als das geliefert.

Spalte C zeigt die Ergebnisse mit Nachkommastellen. In Spalte D wurde mit einfachen Tabellenblattfunktionen auf ganze Zahlen gerundet, ohne dass sich die ursprüngliche Summe ändert.

Wie Sie leicht sehen können, zeigt Spalte E bessere Ergebnisse. Sie wurde mit der benutzerdefinierten Funktion *RoundToSum* erstellt.

	A	B	C	D	E
1	**Noch offene Arbeitseinheiten**	86			
2	**Total**	90,6	86	86	86
3	**Lehrer**	**Erledigt**	**Hilfsspalte**	**Noch offen (Formel)**	**Noch offen (RoundToSum)**
4	**Fairer Anteil**	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

Tabellenblattformeln		
Bereich	Formel	
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));"Werte in Spalte B sind nicht absteigend!")
D5:D32	**D5**	=RUNDEN(SUMME(C$4:C5);0)-SUMME(D$4:D4)
E5	**E5**	=WENNFEHLER(RoundToSum(C5:C32;0);0)

RoundToSum im Vergleich

RoundToSum im Vergleich mit anderen “einfachen” Methoden

Es zirkulieren mehrere verschiedene naive Ansätze für das summenerhaltende Runden:

- (der schlechteste) Runde alle Werte mit Ausnahme des Letzten und ersetze dann den letzten Wert durch die Differenz der gerundeten Originalsumme minus der Summe der vorher gerundeten Werte (d.h. aggregiere alle Rundungsfehler im letzten Summanden):

	A	B	C	
1		**Originaldaten**	**Aggregiere Rundungsfehler**	**Formel in C**
2	**Total**	**2,594**	**2,59**	=SUMME(C4:C8)
3				
4		0,875	0,88	=RUNDEN(B4;2)
5		0,865	0,87	=RUNDEN(B5;2)
6		0,344	0,34	=RUNDEN(B6;2)
7		0,455	0,46	=RUNDEN(B7;2)
8		0,055	**0,04**	=RUNDEN(B$2;2)-SUMME(C$4:C7)

- (besser, aber immer noch schlecht) Wende das hintereinandergeschaltete (gleitende) Runden an:

	A	B	C	
1		**Originaldaten**	**Hintereinander-geschaltetes Runden**	**Formel in C**
2	**Total**	**2,593**	**2,59**	=SUMME(C4:C8)
3				
4		0,875	0,88	=RUNDEN(SUMME($B$3:$B4);2)-SUMME($C$3:$C3)
5		0,865	**0,86**	=RUNDEN(SUMME($B$3:$B5);2)-SUMME($C$3:$C4)
6		0,344	0,34	=RUNDEN(SUMME($B$3:$B6);2)-SUMME($C$3:$C5)
7		0,454	**0,46**	=RUNDEN(SUMME($B$3:$B7);2)-SUMME($C$3:$C6)
8		0,055	**0,05**	=RUNDEN(SUMME($B$3:$B8);2)-SUMME($C$3:$C7)

Vergleichen wir beide Ansätze mit *RoundToSum*.

Rechenbeispiel

Wir erzeugen 40 Zufallszahlen *ZUFALLSZAHL() * 1000* und vergleichen wie folgt:

	A	B	C	D	E	F	G	H	I	J
1			I	II	III	IV	V	VI	VII	VIII
2			Original	RoundToSum	Gleitendes Runden	Letzten Summanden anpassen	Einfaches Runden	Differenz II - V	Differenz III - V	Differenz IV - V
3		Summanen	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 Differenz zum Original			0,11	0,14	0,15	0,10			

Tabellenblattformeln		
Bereich	Formel	
D3	**D3**	=RoundToSum(C3:C42)
E3	**E3**	=RUNDEN(SUMME($C$3:$C3);2)-SUMME(E$2:E2)
F3	**F3**	=RUNDEN(C3:C41;2)
F42	**F42**	=RUNDEN(C43;2)-SUMME(F3:F41)
G3	**G3**	=RUNDEN(C3:C42;2)
H3:J3	**H3**	=WENN(ABS(D3:D42-$G3:$G42)<0,000001;"";D3:D42-$G3:$G42)
C43:J43	**C43**	=SUMME(C3:C42)
D44:G44	**D44**	=SUMMENPRODUKT(ABS(D3:D42-$C$3:$C$42))

Wie man sieht, erhalten wir einen aggregierten Rundungsfehler in Höhe von *0,06*, wenn wir alle Werte einfach einzeln runden. Spalte J (VIII) zeigt die Differenz des aggregierten Rundungsfehlers von *0,06* im letzten Summanden. Spalte F (IV) zeigt die korrespondierenden gerundeten Werte. Im schlimmsten Fall würden Sie hier einen aggregierten Rundungsfehler von *n * 0,005* erhalten, wobei n die Anzahl der Zahlen ist. Beispiel: Nehmen Sie an Stelle der 40 Zufallszahlen 40-mal die Zahl *0,005*.

Gute Praxisbeispiele, warum man die Rundungsfehler nicht im letzten Summanden sammeln sollte, bieten normalverteilte Stichproben ganzer Zahlen. Siehe Kapital *Stichprobe normalverteilen*.

Der Ansatz des hintereinandergeschalteten (gleitenden) Rundens in Spalte I (VII) zeigt 12 Rundungen zur falschen Seite. Spalte E (III) zeigt die korrespondierenden gerundeten Werte. Beim hintereinandergeschalteten Runden können Sie im schlimmsten Fall die Hälfte der Zahlen zur

falschen Seite runden, obwohl alle Zahlen korrekt gerundet werden könnten. Beispiel: Nehmen Sie an Stelle der 40 Zufallszahlen 20-mal die Zahl *-0,0049999* und dann 20-mal die Zahl *0,0049999*.

Im Gegensatz dazu rundet das optimale *RoundToSum* lediglich 6 Werte zur falschen Seite und wendet die geringste Anzahl von Änderungen an, um die korrekte gerundete Gesamtsumme mit dem kleinsten absoluten Fehler zu erhalten. Im schlimmsten Fall würden Sie nun n/2 Male zur falschen Seite runden müssen, wobei n die Anzahl der Zahlen ist. Beispiel: Nehmen Sie an Stelle der 40 Zufallszahlen erneut 40-mal die Zahl *0,005*. Es ist jedoch die beste Lösung mit dem kleinsten absoluten Rundungsfehler für jede Zahl und danach mit der geringsten Anzahl von Rundungen zur falschen Seite.

Zusammenfassung

Verwenden Sie *RoundToSum*. Es wendet die geringste Anzahl von Änderungen an, um die korrekte gerundete Gesamtsumme mit dem kleinsten absoluten (oder relativen) Fehler zu erhalten.

Ein hintereinandergeschaltetes Runden benötigt kein VBA und auch keine Matrixformel, aber es benötigt mindestens so viele Rundungsabweichungen wie *RoundToSum* und es kann leicht wesentlich mehr unnatürliche Rundungen aufweisen, die man nur schwerlich erklären kann.

Am schlimmsten ist jedoch der Ansatz, alle Rundungsdifferenzen in einer (hier: der letzten) Zelle zu aggregieren. Man stelle sich *1.000* Menschen mit je *49* Cent in der Tasche (also insgesamt *490* EUR) vor. Wenn Sie diese Beträge fair auf ganze Euro gerundet verteilen wollen, würde bei diesem Ansatz der Letzte die gesamten *490* Euro erhalten. RoundToSum gäbe den ersten *490* Personen je einen Euro und allen anderen Null Euro.

RoundToSum im Vergleich zum D'Hondt Verfahren

RoundToSum implementiert das Hare-Niemeyer Verfahren. Dies ist in mancher Hinsicht hinsichtlich der fairen Mandatsverteilung dem D'Hondt Verfahren überlegen. So ist z. B. beim Hare-Niemeyer Verfahren die relative Prozentdifferenz zur idealen Verteilung absolut geringer:

	A	B	C	D	E	F
1		69	Sitze		Rel. % Differenz zur Idealverteilung	
2	Partei	Stimmen	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		

Tabellenblattformeln		
Bereich	Formel	
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	=SUMME(B3:B7)

Programmcode sbDHondt

```
Function sbdHondt(lSeats As Long, vVotes As Variant) As Variant
'Implements the d'Hondt method for allocating seats in
'party-list proportional representation political election
'systems.
'(C) (P) by Bernd Plumhoff 01-Dec-2009 PB V0.10
Dim i As Long, k As Long, n As Long
Dim vA As Variant, vB As Variant, vR As Variant
Dim dMax As Double

With Application.WorksheetFunction
vA = .Transpose(.Transpose(vVotes))
vB = vA
n = UBound(vA, 1)
ReDim vR(1 To n, 1 To 1) As Variant
ReDim lDenom(1 To n) As Long

Do While i < lSeats
  'identify max
  dMax = .Max(vB)
  k = .Match(dMax, vB, 0)
  lDenom(k) = lDenom(k) + 1
  vB(k, 1) = vA(k, 1) / (lDenom(k) + 1#)
  vR(k, 1) = vR(k, 1) + 1
  i = i + 1
Loop
sbdHondt = vR
End With
End Function
```

Literatur

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.

N. Herrmann, Mathematik ist überall, Oldenbourg Verlag München Wien, ISBN 3-486-57583-X (Kapitel 12, Das Wahl-Problem).

Index