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

Status: 12-October-2024

Abstract

Rounded values do not always sum up to their original total, as demonstrated in this article. How can you ensure that the sum of rounded percentages equals exactly 100%? Is it possible to guarantee that, for accounting purposes, the distribution of overhead costs precisely matches the original total? These challenges are well-known and have been studied extensively.

This article introduces a simple solution using Excel/VBA. The function presented here can round relative values (e.g., percentages) to ensure they sum to exactly 100%. It can also round absolute values (such as cost distributions) while preserving their original sum after rounding. A key parameter allows users to choose which type of error to minimize — absolute error or relative error — compared to the common half-up rounding method.

Table of Contents

Rounding Values Preserving Their Sum

If you need to round values without changing their sum, you might need to round one or more summands to the more distant rounded value.

Percentage Example

For example, the values 11, 45, and 555, which sum to 611, do not yield a percentage total of 100.00 but rather 99.99 if rounded to two decimal places. The **bold** values in non-sum cells have been adjusted using the *RoundToSum* function:

	Values	**Percentage rounded to 2 decimals**	**Minimize absolute Error**	**Minimize relative Error**
	11	1.80	1.80	1.80
	45	7.36	**7.37**	7.36
	555	90.83	90.83	**90.84**
Sum	611	99.99	100.00	100.00

The Excel / VBA function call *RoundToSum({11,45,555},2,FALSE,1)* would result in *{1.80,7.37,90.83}*, though. Here, the percentage value *7.364975* is rounded differently to achieve a percentage sum of *100.00* and to minimize the absolute error compared to half-up rounding. By using *RoundToSum({11,45,555},2,FALSE,2)* we would have received *{1.80,7.36,90.84}*, as this would minimize the relative error.

Example with Absolute Values

The sum of the second column differs by *+2,000* from the rounded sum. The **bold** values in non-sum cells have been adjusted using the *RoundToSum* function:

	Values	**Rounded to absolute 1,000**	**Minimize absolute Error**	**Minimize relative Error**
	4.523	5.000	5.000	5.000
	456	0	0	0
	-78.845	-79.000	-79.000	-79.000
	-14.491	-14.000	**-15.000**	-14.000
	65.789	66.000	66.000	66.000
	129.512	130.000	**129.000**	**129.000**
	15.562	16.000	16.000	16.000
	548.555	549.000	549.000	**548.000**
	1.590	2.000	2.000	2.000
	-897	-1.000	-1.000	-1.000
	6.968	7.000	7.000	7.000
	2.987	3.000	3.000	3.000
Sum	681.709	684.000	682.000	682.000

The User-Defined VBA Function *RoundToSum*

Name

RoundToSum – Rounding values preserving their rounded sum

Synopsis

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

Description

RoundToSum rounds values without altering their rounded sum. It uses the largest remainder method to minimize the error compared to the commonly used half-up rounding method. If the error is identical for one or more values, the first value(s) encountered will be adjusted.

Note: This solution is limited to one-dimensional tables without subtotals. There is no general solution for higher-dimensional tables or tables with subtotals.

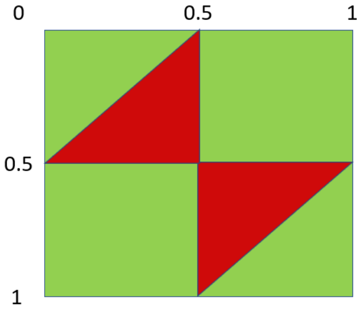
Parameters

vInput	Range or array containing the unrounded input values.
lDigits	Optional, default value is 2. The number of digits to round to. For example: 0 rounds to integers, 2 rounds to the nearest cent, -3 rounds to the nearest thousand.
bAbsSum	Optional, default value is TRUE. TRUE rounds the values directly. FALSE adjusts the percentages so they sum to exactly 100%.
lErrorType	Optional, default value is 1. The type of error to minimize: 1 for absolute error, 2 for relative error.
bDontAmend	Optional, default value is FALSE. TRUE prevents adjusting the values to match the rounded sum. FALSE makes adjustments as described above. This parameter is mainly for demonstration purposes to see the function’s impact.

RoundToSum Program Code

```
Option Explicit

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, Optional bDontAmend As Boolean = False) 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.
'V2.1 PB 12-Oct-2024 (C) (P) by Bernd Plumhoff
Dim 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
If Not bDontAmend Then
  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
End If
RoundToSum = vC
If TypeName(Application.Caller) = "Range" Then
  If Application.Caller.Rows.Count > Application.Caller.Columns.Count Then
    RoundToSum = .Transpose(vC) 'It's two-dimensional with 2nd dim const = 1
  End If
End If
Exit Function
Errhdl:
'Transpose variants to be able to address them with vA(i), not vA(i,1)
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 5) 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"
ArgDesc(5) = "[Optional = False] True does not amend the rounded summands to match the rounded sum; False performs the calculation as described"
Application.MacroOptions _
  Macro:=FuncName, _
  Description:=FuncDesc, _
  Category:=Category, _
  ArgumentDescriptions:=ArgDesc
End Sub
```

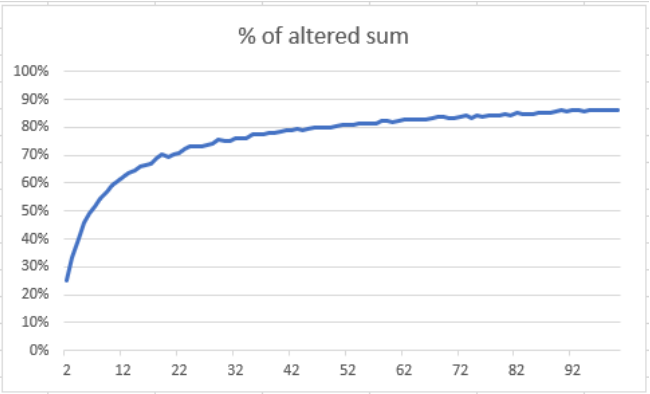
Rounding Values Alters Their Sum



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

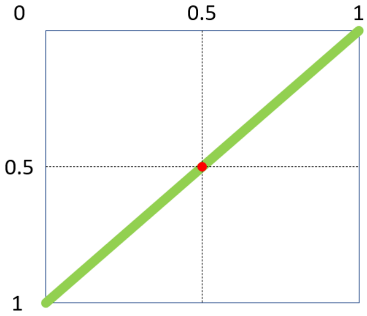
For two random floating point numbers this is obvious: The likelihood is around 25% - that is the percentage of **red** in this picture:

But it might be somewhat surprising that the likelihood approaches 90% if you round and add more and more numbers:



With seven floating point numbers the likelihood is already larger than 50% that the sum of rounded values is not equal to their rounded sum.

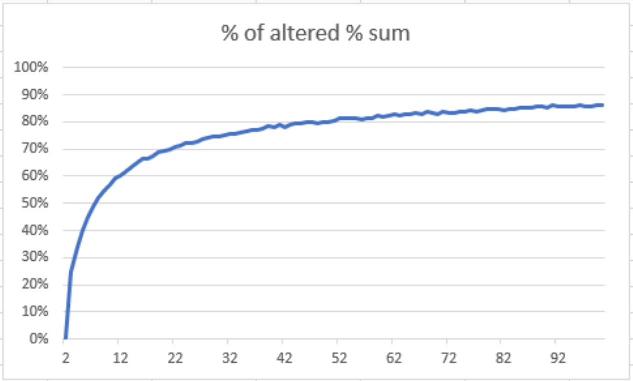
Rounded Percentages



Rounded percentages also often fail to add up to 100%.

With two random numbers the issue arises only if both numbers equal 0.5:

But with more random numbers it is similar to the problem stated initially, just with around one number more. Rounded percentages of three arbitrary numbers fail to add up to 1 with a chance of around 25%:



Monte Carlo Program Code

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

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

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

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

With Application.WorksheetFunction
Randomize
For i = 2 To n
  m = 0
  ReDim e(1 To i) As Double
  For j = 1 To runs
    s1 = 0#
    For k = 1 To i
      If bOnlyPositive Then
        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
```

Usage Examples of *RoundToSum*

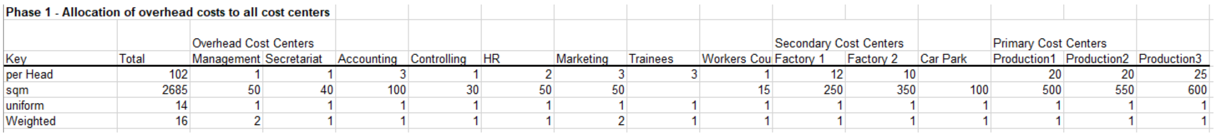
Allocation of Overheads

When allocating overhead costs to products you often encounter the fact that the resulting sum of allocated overheads does not equal the original cost sum. Due to rounding differences you frequently face a little cent difference. In this case the user defined function *RoundToSum* can help.

A Real-Life Example

We present an allocation of overheads where all individual cent values accurately add up to their intermediate or final sums.

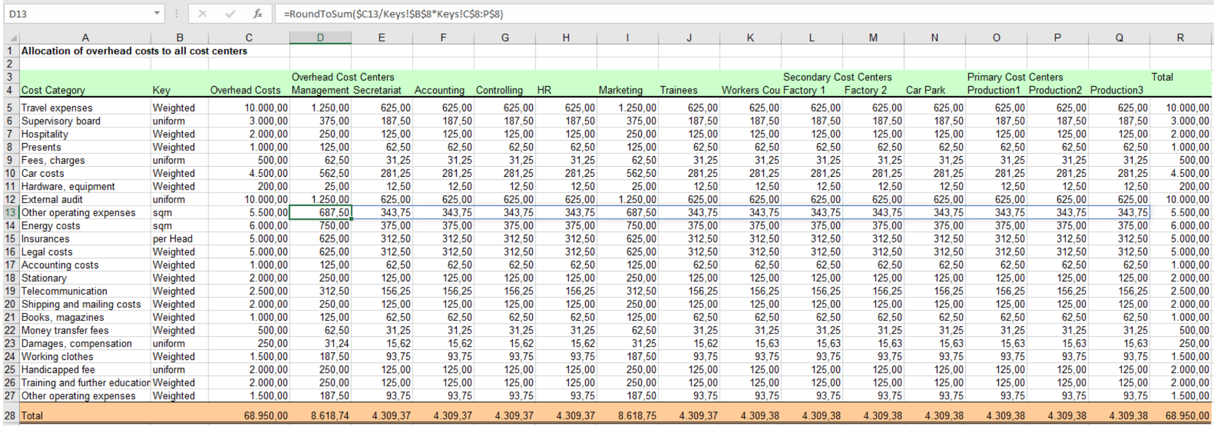
First you define how the overheads have to be allocated to support cost centres:



Phase 1 - Allocation of overhead costs to all cost centers

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

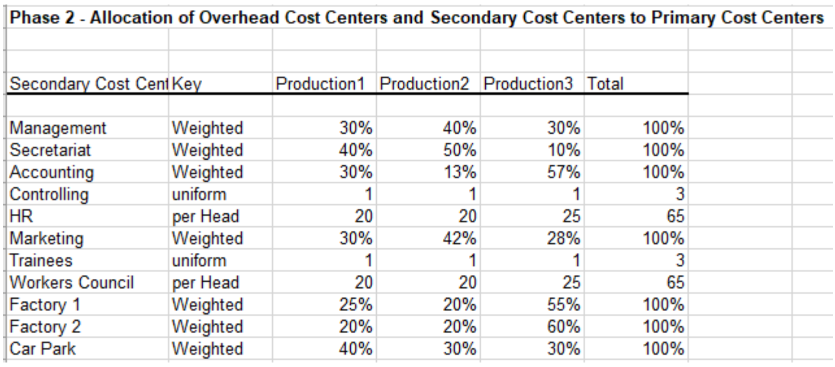
The first allocation of overheads uses a rounding correction so that all summands accurately sum up on support cost centre level:



D13 =RoundToSum($C13/Keys!$B$8*Keys!C$8:P$8)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	**Allocation of overhead costs to all cost centers**																	
2																		
3				Overhead Cost Centers								Secondary Cost Centers			Primary Cost Centers			Total
4	Cost Category	Key	Overhead Costs	Management	Secretariat	Accounting	Controlling	HR	Marketing	Trainees	Workers Cou	Factory 1	Factory 2	Car Park	Production1	Production2	Production3	
5	Travel expenses	Weighted	10.000,00	1.250,00	625,00	625,00	625,00	625,00	1.250,00	625,00	625,00	625,00	625,00	625,00	625,00	625,00	625,00	10.000,00
6	Supervisory board	uniform	3.000,00	375,00	187,50	187,50	187,50	187,50	375,00	187,50	187,50	187,50	187,50	187,50	187,50	187,50	187,50	3.000,00
7	Hospitality	Weighted	2.000,00	250,00	125,00	125,00	125,00	125,00	250,00	125,00	125,00	125,00	125,00	125,00	125,00	125,00	125,00	2.000,00
8	Presents	Weighted	1.000,00	125,00	62,50	62,50	62,50	62,50	125,00	62,50	62,50	62,50	62,50	62,50	62,50	62,50	62,50	1.000,00
9	Fees, charges	uniform	500,00	62,50	31,25	31,25	31,25	31,25	62,50	31,25	31,25	31,25	31,25	31,25	31,25	31,25	31,25	500,00
10	Car costs	Weighted	4.500,00	562,50	281,25	281,25	281,25	281,25	562,50	281,25	281,25	281,25	281,25	281,25	281,25	281,25	281,25	4.500,00
11	Hardware, equipment	Weighted	200,00	25,00	12,50	12,50	12,50	12,50	25,00	12,50	12,50	12,50	12,50	12,50	12,50	12,50	12,50	200,00
12	External audit	uniform	10.000,00	1.250,00	625,00	625,00	625,00	625,00	1.250,00	625,00	625,00	625,00	625,00	625,00	625,00	625,00	625,00	10.000,00
13	Other operating expenses	sqm	5.500,00	687,50	343,75	343,75	343,75	343,75	687,50	343,75	343,75	343,75	343,75	343,75	343,75	343,75	343,75	5.500,00
14	Energy costs	sqm	6.000,00	750,00	375,00	375,00	375,00	375,00	750,00	375,00	375,00	375,00	375,00	375,00	375,00	375,00	375,00	6.000,00
15	Insurances	per Head	5.000,00	625,00	312,50	312,50	312,50	312,50	625,00	312,50	312,50	312,50	312,50	312,50	312,50	312,50	312,50	5.000,00
16	Legal costs	Weighted	5.000,00	625,00	312,50	312,50	312,50	312,50	625,00	312,50	312,50	312,50	312,50	312,50	312,50	312,50	312,50	5.000,00
17	Accounting costs	Weighted	1.000,00	125,00	62,50	62,50	62,50	62,50	125,00	62,50	62,50	62,50	62,50	62,50	62,50	62,50	62,50	1.000,00
18	Stationary	Weighted	2.000,00	250,00	125,00	125,00	125,00	125,00	250,00	125,00	125,00	125,00	125,00	125,00	125,00	125,00	125,00	2.000,00
19	Telecommunication	Weighted	2.500,00	312,50	156,25	156,25	156,25	156,25	312,50	156,25	156,25	156,25	156,25	156,25	156,25	156,25	156,25	2.500,00
20	Shipping and mailing costs	Weighted	2.000,00	250,00	125,00	125,00	125,00	125,00	250,00	125,00	125,00	125,00	125,00	125,00	125,00	125,00	125,00	2.000,00
21	Books, magazines	Weighted	1.000,00	125,00	62,50	62,50	62,50	62,50	125,00	62,50	62,50	62,50	62,50	62,50	62,50	62,50	62,50	1.000,00
22	Money transfer fees	Weighted	500,00	62,50	31,25	31,25	31,25	31,25	62,50	31,25	31,25	31,25	31,25	31,25	31,25	31,25	31,25	500,00
23	Damages, compensation	uniform	250,00	31,24	15,62	15,62	15,62	15,62	31,25	15,62	15,63	15,63	15,63	15,63	15,63	15,63	15,63	250,00
24	Working clothes	Weighted	1.500,00	187,50	93,75	93,75	93,75	93,75	187,50	93,75	93,75	93,75	93,75	93,75	93,75	93,75	93,75	1.500,00
25	Handicapped fee	uniform	2.000,00	250,00	125,00	125,00	125,00	125,00	250,00	125,00	125,00	125,00	125,00	125,00	125,00	125,00	125,00	2.000,00
26	Training and further educatior	Weighted	2.000,00	250,00	125,00	125,00	125,00	125,00	250,00	125,00	125,00	125,00	125,00	125,00	125,00	125,00	125,00	2.000,00
27	Other operating expenses	Weighted	1.500,00	187,50	93,75	93,75	93,75	93,75	187,50	93,75	93,75	93,75	93,75	93,75	93,75	93,75	93,75	1.500,00
28	Total		68.950,00	8.618,74	4.309,37	4.309,37	4.309,37	4.309,37	8.618,75	4.309,37	4.309,38	4.309,38	4.309,38	4.309,38	4.309,38	4.309,38	4.309,38	68.950,00

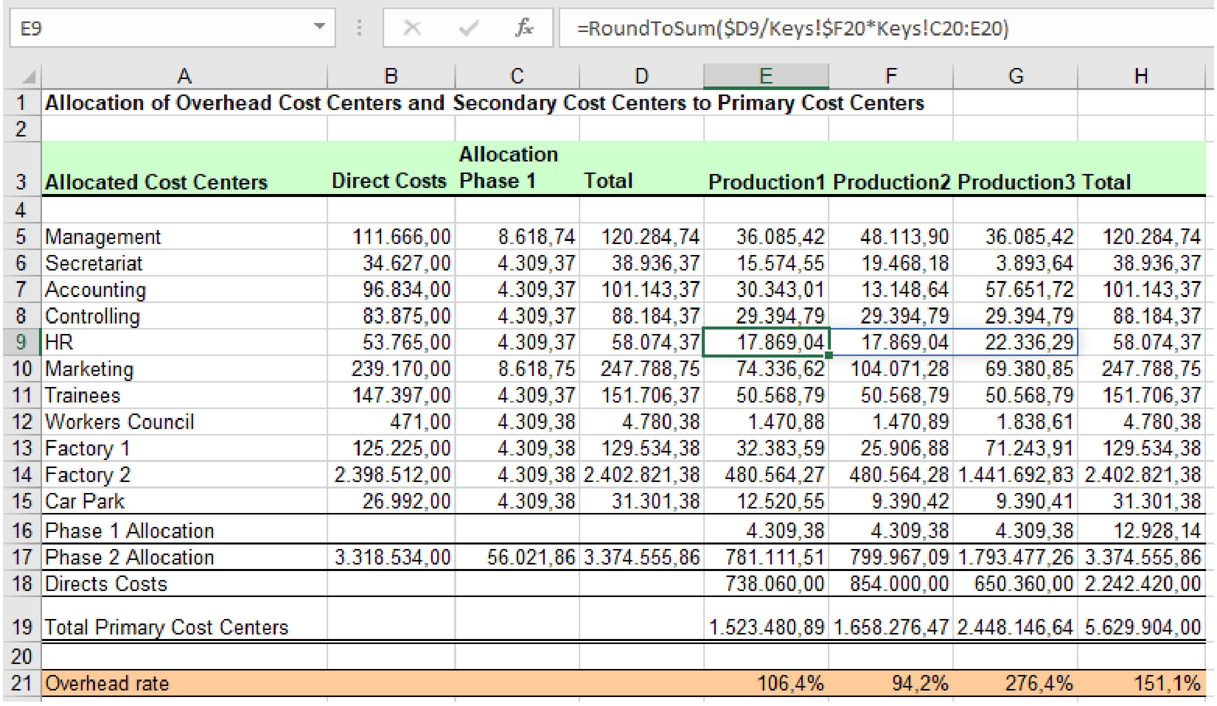
The second allocation of overheads also uses a rounding correction so that all support cost centres get accurately distributed to products:



Phase 2 - Allocation of Overhead Cost Centers and Secondary Cost Centers to Primary Cost Centers

Secondary Cost Cent	Key	Production1	Production2	Production3	Total
Management	Weighted	30%	40%	30%	100%
Secretariat	Weighted	40%	50%	10%	100%
Accounting	Weighted	30%	13%	57%	100%
Controlling	uniform	1	1	1	3
HR	per Head	20	20	25	65
Marketing	Weighted	30%	42%	28%	100%
Trainees	uniform	1	1	1	3
Workers Council	per Head	20	20	25	65
Factory 1	Weighted	25%	20%	55%	100%
Factory 2	Weighted	20%	20%	60%	100%
Car Park	Weighted	40%	30%	30%	100%

The final result:



=RoundToSum($D9/Keys!$F20*Keys!C20:E20)

	A	B	C	D	E	F	G	H
1	**Allocation of Overhead Cost Centers and Secondary Cost Centers to Primary Cost Centers**							
2								
3	**Allocated Cost Centers**	**Direct Costs**	**Allocation Phase 1**	**Total**	**Production1**	**Production2**	**Production3**	**Total**
4								
5	Management	111.666,00	8.618,74	120.284,74	36.085,42	48.113,90	36.085,42	120.284,74
6	Secretariat	34.627,00	4.309,37	38.936,37	15.574,55	19.468,18	3.893,64	38.936,37
7	Accounting	96.834,00	4.309,37	101.143,37	30.343,01	13.148,64	57.651,72	101.143,37
8	Controlling	83.875,00	4.309,37	88.184,37	29.394,79	29.394,79	29.394,79	88.184,37
9	HR	53.765,00	4.309,37	58.074,37	17.869,04	17.869,04	22.336,29	58.074,37
10	Marketing	239.170,00	8.618,75	247.788,75	74.336,62	104.071,28	69.380,85	247.788,75
11	Trainees	147.397,00	4.309,37	151.706,37	50.568,79	50.568,79	50.568,79	151.706,37
12	Workers Council	471,00	4.309,38	4.780,38	1.470,88	1.470,89	1.838,61	4.780,38
13	Factory 1	125.225,00	4.309,38	129.534,38	32.383,59	25.906,88	71.243,91	129.534,38
14	Factory 2	2.398.512,00	4.309,38	2.402.821,38	480.564,27	480.564,28	1.441.692,83	2.402.821,38
15	Car Park	26.992,00	4.309,38	31.301,38	12.520,55	9.390,42	9.390,41	31.301,38
16	Phase 1 Allocation				4.309,38	4.309,38	4.309,38	12.928,14
17	Phase 2 Allocation	3.318.534,00	56.021,86	3.374.555,86	781.111,51	799.967,09	1.793.477,26	3.374.555,86
18	Directs Costs				738.060,00	854.000,00	650.360,00	2.242.420,00
19	Total Primary Cost Centers				1.523.480,89	1.658.276,47	2.448.146,64	5.629.904,00
20								
21	Overhead rate				106,4%	94,2%	276,4%	151,1%

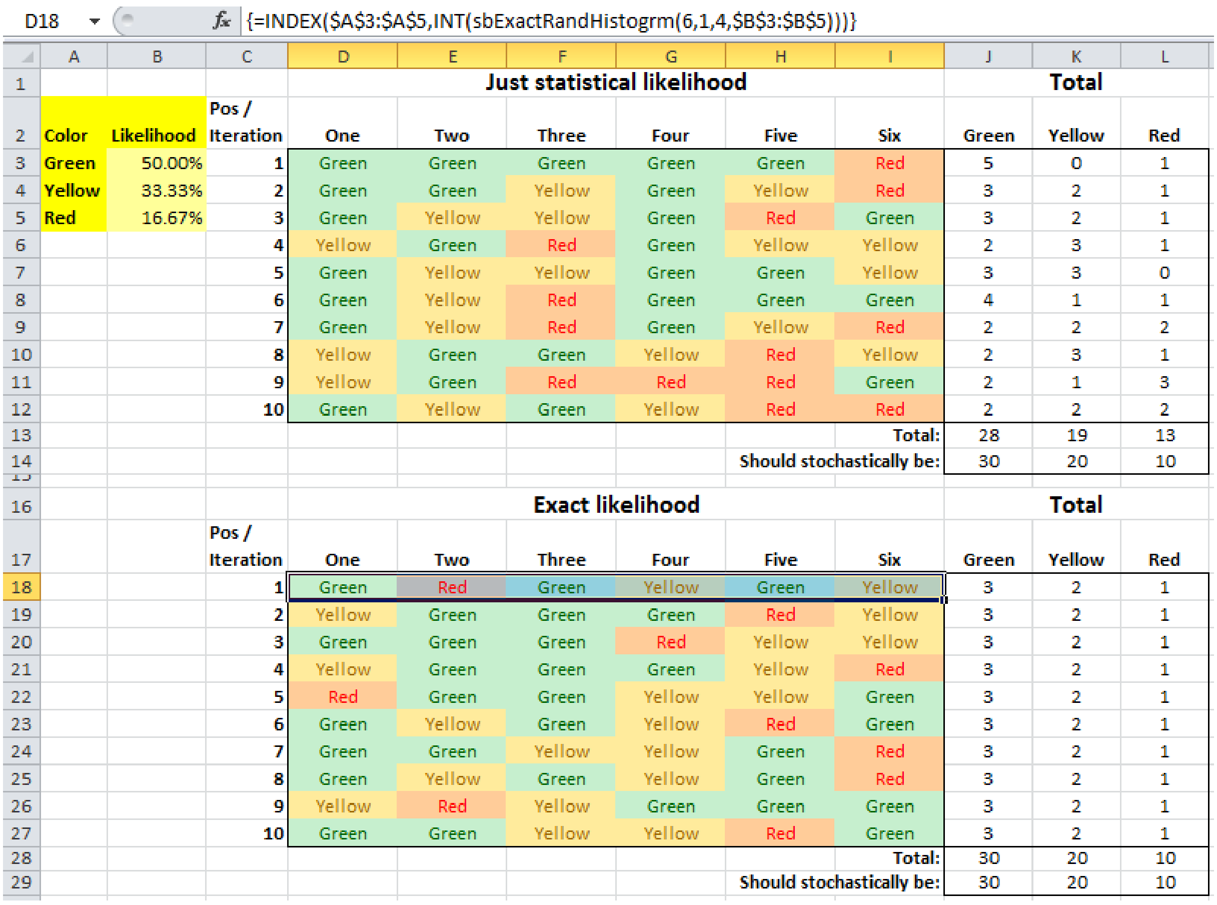
This correct allocation of overheads you will be able to enter into a general ledger without any cent / penny difference.

Example of an Exact Relation of Random Numbers

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

But what if you want to create 7 rolls of this die and all numbers between 1 and 5 should appear exactly once and 6 exactly twice?

Here is a general solution:



D18 fx {=INDEX($A$3:$A$5,INT(sbExactRandHistogrm(6,1,4,$B$3:$B$5)))}

	A	B	C	D	E	F	G	H	I	J	K	L
1				Just statistical likelihood						Total		
2	Color	Likelihood	Pos / Iteration	One	Two	Three	Four	Five	Six	Green	Yellow	Red
3	Green	50.00%	1	Green	Green	Green	Green	Green	Red	5	0	1
4	Yellow	33.33%	2	Green	Green	Yellow	Green	Yellow	Red	3	2	1
5	Red	16.67%	3	Green	Yellow	Yellow	Green	Red	Green	3	2	1
6			4	Yellow	Green	Red	Green	Yellow	Yellow	2	3	1
7			5	Green	Yellow	Yellow	Green	Green	Yellow	3	3	0
8			6	Green	Yellow	Red	Green	Green	Green	4	1	1
9			7	Green	Yellow	Red	Green	Yellow	Red	2	2	2
10			8	Yellow	Green	Green	Yellow	Red	Yellow	2	3	1
11			9	Yellow	Green	Red	Red	Red	Green	2	1	3
12			10	Green	Yellow	Green	Yellow	Red	Red	2	2	2
13									Total:	28	19	13
14									Should stochastically be:	30	20	10
16				Exact likelihood						Total		
17			Pos / Iteration	One	Two	Three	Four	Five	Six	Green	Yellow	Red
18			1	Green	Red	Green	Yellow	Green	Yellow	3	2	1
19			2	Yellow	Green	Green	Green	Red	Yellow	3	2	1
20			3	Green	Green	Green	Red	Yellow	Yellow	3	2	1
21			4	Yellow	Green	Green	Green	Yellow	Red	3	2	1
22			5	Red	Green	Green	Yellow	Yellow	Green	3	2	1
23			6	Green	Yellow	Green	Yellow	Red	Green	3	2	1
24			7	Green	Green	Yellow	Yellow	Green	Red	3	2	1
25			8	Green	Yellow	Green	Yellow	Green	Red	3	2	1
26			9	Yellow	Red	Yellow	Green	Green	Green	3	2	1
27			10	Green	Green	Yellow	Yellow	Red	Green	3	2	1
28									Total:	30	20	10
29									Should stochastically be:	30	20	10

The User-Defined VBA Function sbExactRandHistogrm

Name

sbExactRandHistogrm – Create an exact double histogram distribution.

Synopsis

sbExactRandHistogrm(ldraw, dmin, dmax, vWeight)

Description

sbExactRandHistogrm creates an exact histogram distribution for ldraw draws of floating point numbers with double precision within range dmin:dmax. This range is divided into *vWeight.count* classes. Each class has weight *vWeight(i)*, reflecting the probability of occurrence of a value within the class. If weights can’t be achieved exactly for *ldraw* draws the largest remainder method will be applied to minimize the absolute error. This function calls *RoundToSum*.

Parameters

ldraw Number of draws

dmin Minimum = lower boundary of range of numbers to draw

dmax Maximum = upper boundary of range of numbers to draw

vWeight Array of weights. Array size determines the number of different classes the range *dmin : dmax* is divided into. Values in this array specify likelihood of this class' numbers to appear (be drawn).

sbRandHistogrm Program Code

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

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

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

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

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

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

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

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

For j = 1 To ldraw

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

    dR = dSumWeight * Rnd

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

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

Next j

sbExactRandHistogrm = vR

Exit Function

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

End Function
```

Fair Staff Selection Based on Team Size

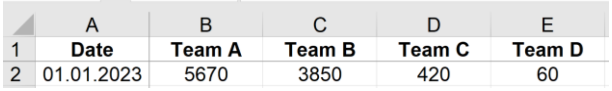
Let us assume your company needs to get some special tasks done. All staff members can do the work. You want the teams to second their staff based on the size of each team. This selection can be done by the user-defined function *RoundToSum*.

Since we cannot guarantee that each team can provide staff exactly in relation to its staff number for each special task, we need to call *RoundToSum* including a lookback onto previous staff selections.

RoundToSum uses the largest remainder method (also called Hare-Niemeyer) which can suffer from the Alabama paradoxon. If the total number of staff to be selected increases it can happen that a team needs to provide less staff than before. Because we cannot account for this in hindsight, this paradoxon needs to be dealt with as soon as it occurs.

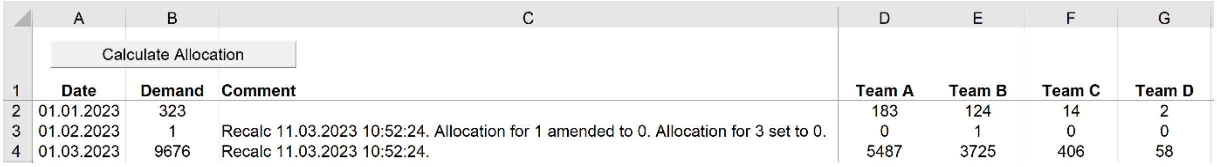
Example

On 1-Jan-2023 these teams exist:



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

Over the following three months these staff numbers are required for special tasks and are selected:



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

On 1-Feb-2023 the largest remainder method would have selected a total number of 184, 125, 13, and 2 employees of teams A, B, C, and D ausgewählt. But on 1-Jan-2023 team C had already provided 14 members of staff which cannot be taken back. This means that team A or team B needs to provide one employee less. The implemented algorithm looks left to right to account for this, so in this case team A is impacted.

On 1-Mar-2023 all remaining staff counts of all teams are requested. The algorithm selects for each team exactly its staff count in total because the lookback includes all request data records.

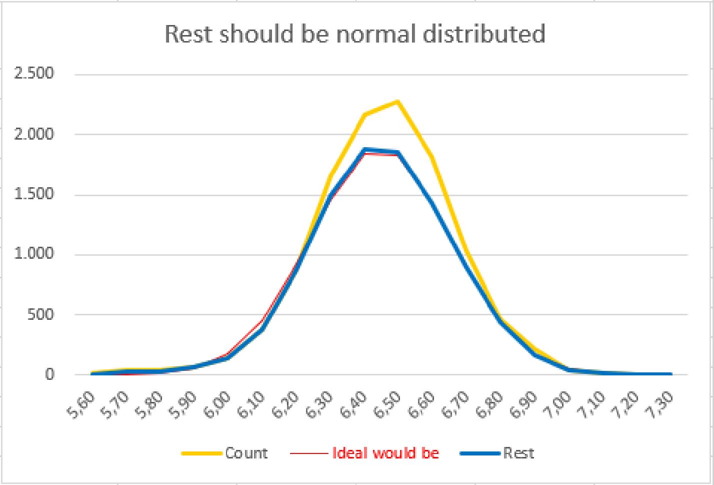
sbFairStaffSelection Program Code

```
Enum TeamColums
    tc_Date = 1
    tc_TeamStart
End Enum

Enum AllocationColumns
    ac_Date = 1
    ac_Demand
    ac_Comment
    ac_TeamStart
End Enum

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

Dim bLookBack                As Boolean
Dim bReCalc                  As Boolean

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

Dim sComment                 As String

Dim vAlloc                   As Variant
Dim vTeams                   As Variant

Dim state                    As SystemState

Set state = New SystemState

With Application.WorksheetFunction

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

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

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

    If lDemand <> lRowSum Then bReCalc = True

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

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

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

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

        If bLookBack Then
            For m = 1 To j
                lCellResult = vAlloc(m) - lTeamSum(m)
                If lCellResult < 0 Then
                    'The Alabama Paradoxon: we have to reduce other parties'
                    'allocations because we cannot have negative allocations
                    lAmend = lAmend - lCellResult
                End If
                vAlloc(m) = lCellResult
            Next m
            If lAmend > 0 Then
                For m = 1 To j
                    lCellResult = vAlloc(m)
                    If lCellResult < 0 Then
                        vAlloc(m) = 0
                        sComment = sComment & "Allocation for " & m & " set to 0. "
                    ElseIf lCellResult > 0 And lAmend > 0 Then
                        If lCellResult > lAmend Then
                            vAlloc(m) = lCellResult - lAmend
                            lAmend = 0
                        Else
                            vAlloc(m) = 0
                            lAmend = lAmend - lCellResult
                        End If
                        sComment = sComment & "Allocation for " & m & " amended to " & _
                                   vAlloc(m) & ". "
                    End If
                Next m
            End If
        End If
        wsA.Cells(i, ac_Comment) = sComment
        For m = 1 To j
            wsA.Cells(i, ac_TeamStart + m - 1) = vAlloc(m)
        Next m

    End If

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

Loop

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

End With

End Sub
```

Distribute a Sample Normally

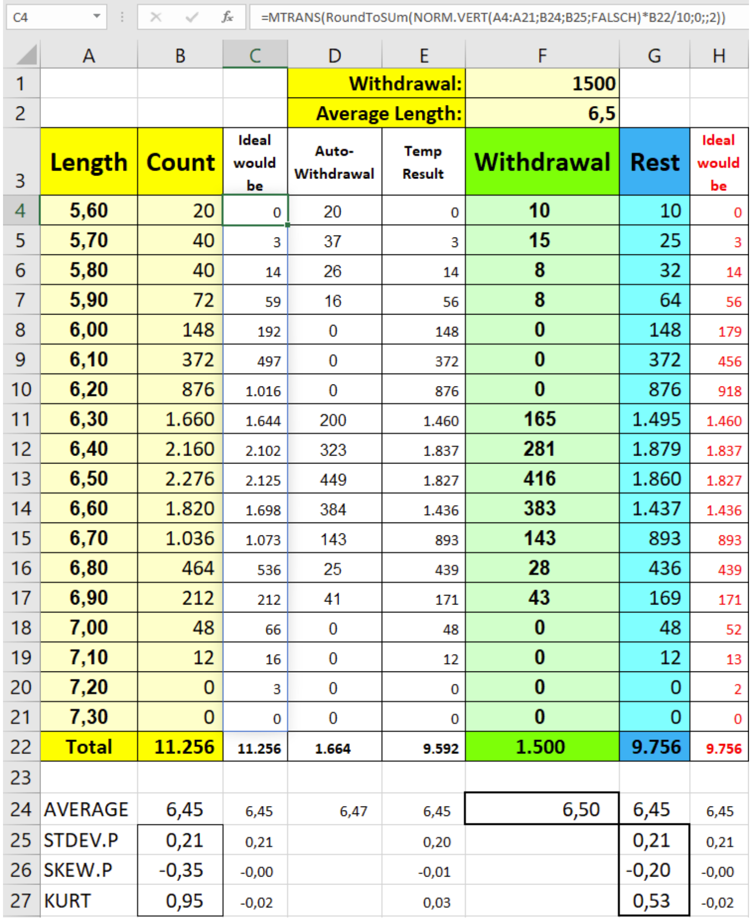
You have 11.256 christmas trees in stock. A customer wants to purchse 1.500 of them, with one condition: the average height of the trees must be 6.50 meters.

Your goal is to keep the remaining trees as close to a normal distribution as possible.



How can you achieve this?

A Sample Calculation



C4 =MTRANS(RoundToSUm(NORM.VERT(A4:A21;B24;B25;FALSCH)*B22/10;0;;2))

	A	B	C	D	E	F	G	H
1				Withdrawal:		1500		
2				Average Length:		6,5		
3	Length	Count	Ideal would be	Auto-Withdrawal	Temp Result	Withdrawal	Rest	Ideal would be
4	5,60	20	0	20	0	10	10	0
5	5,70	40	3	37	3	15	25	3
6	5,80	40	14	26	14	8	32	14
7	5,90	72	59	16	56	8	64	56
8	6,00	148	192	0	148	0	148	179
9	6,10	372	497	0	372	0	372	456
10	6,20	876	1.016	0	876	0	876	918
11	6,30	1.660	1.644	200	1.460	165	1.495	1.460
12	6,40	2.160	2.102	323	1.837	281	1.879	1.837
13	6,50	2.276	2.125	449	1.827	416	1.860	1.827
14	6,60	1.820	1.698	384	1.436	383	1.437	1.436
15	6,70	1.036	1.073	143	893	143	893	893
16	6,80	464	536	25	439	28	436	439
17	6,90	212	212	41	171	43	169	171
18	7,00	48	66	0	48	0	48	52
19	7,10	12	16	0	12	0	12	13
20	7,20	0	3	0	0	0	0	2
21	7,30	0	0	0	0	0	0	0
22	Total	11.256	11.256	1.664	9.592	1.500	9.756	9.756
23								
24	AVERAGE	6,45	6,45	6,47	6,45	6,50	6,45	6,45
25	STDEV.P	0,21	0,21		0,20		0,21	0,21
26	SKEW.P	-0,35	-0,00		-0,01		-0,20	-0,00
27	KURT	0,95	-0,02		0,03		0,53	-0,02

Let's assume the count and distribution of trees are as shown in the diagram above.

A useful first step is to check whether your original sample is already fairly normally distributed. We can calculate skewness using the function *sbSWV*. The skewness is approximately *=sbSWV("SKEW.P";$A$4:$A$21;B$4:B$21) = -0.35*. Similarly, we calculate kurtosis with *=sbSWV("KURT";$A$4:$A$21;B$4:B$21)*, resulting in approximately *0.95*. As shown in the diagram (yellow-orange graph), the original sample is already fairly normally distributed.

However, ideally, the distribution would match the one shown in column C (formula: *=TRANSPOSE(RoundToSum(NORM.DIST(A4:A21,B24,B25,FALSE)*B22/10,0)))*. In this case, skewness and kurtosis would be close to zero, though rounding may lead to slight deviations.

Column H displays the ideal normal distribution after the withdrawal of trees.

The automated withdrawals in column D aim to achieve this ideal distribution. However, this is only possible if there are enough trees of the necessary lengths. If not, you must enter a withdrawal of 0, as trees cannot be added to the sample. For instance, in the diagram, you can see that at a length of 6.10 meters, the ideal distribution exceeds the actual remaining distribution.

The original formulas in column F should read =D4 to =D21.

These formulas are manually overwritten to:

- Achieve a total withdrawal of 1,500 trees.
- Ensure an average tree height of 6.5 meters.
- Maintain a standard deviation in the remaining sample close to the original.
- Reduce the absolute skewness compared to the original.
- Reduce the absolute kurtosis compared to the original.

In the provided sample file, significant deviations are highlighted using conditional formatting.

A Note of Caution:

It’s not always possible to achieve a fairly normal distribution in the remaining sample. It might even be impossible to withdraw trees that meet a desired average - for example, asking for 21 trees with an average height of 5.60 meters could be unachievable.

Helper Functions

Excel offers many basic statistical functions, but they don’t handle weighted values. The user-defined function *sbSWV* (statistics for weighted values) used here provides an easy and quick assessment of how well the samples are normally distributed.

To ensure that the sum of the ideal integer distributions matches the sum of the original samples, the user-defined function *RoundToSum* was employed. Note that the parameter 2 is used for error type to minimize the relative error, preventing artificial rounding errors, particularly in the tails of the distributions.

sbSWV Program Code

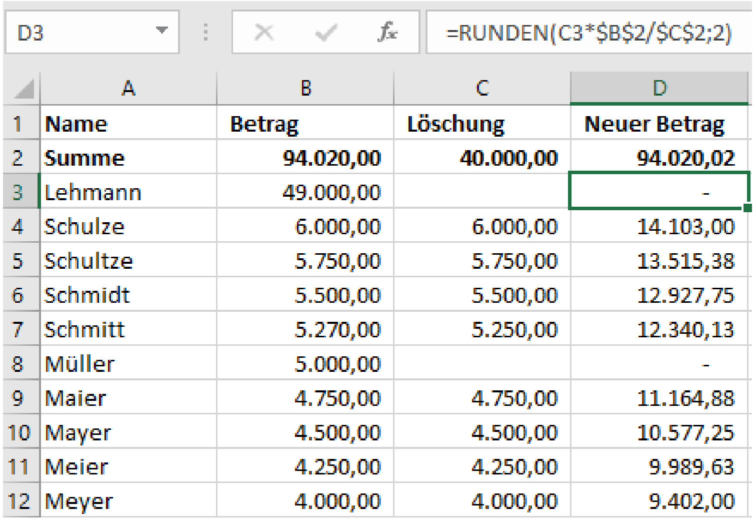
```
#Const SORTED = False

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

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

Distribution of Budgets Among Remaining Staff

When staff members leave, their budgets can be redistributed among the remaining employees based on initial budget. But how can this redistribution be done accurately and fairly?

A Simple Approach

A simple formula which you can copy down from D3 to D12 is *=ROUND(C3*$B$2/$C$2,2)*:



D3

=RUNDEN(C3*$B$2/$C$2;2)

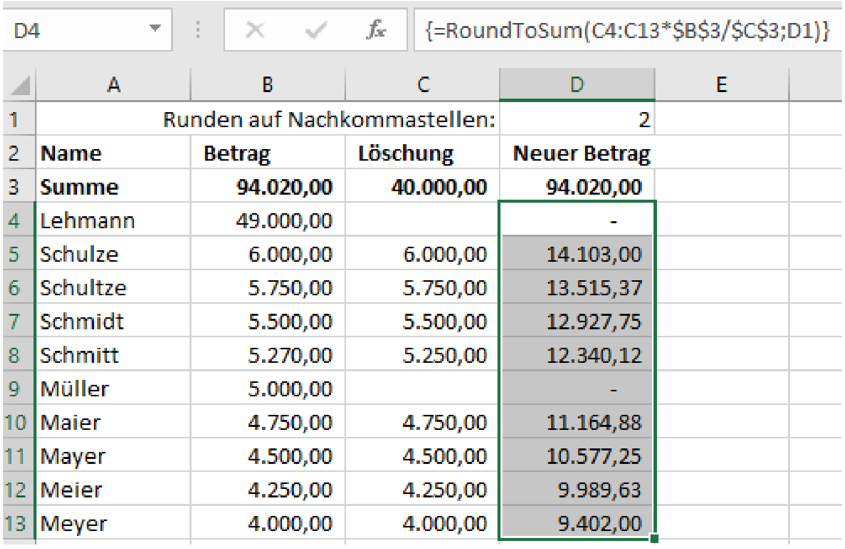
	A	B	C	D
1	**Name**	**Betrag**	**Löschung**	**Neuer Betrag**
2	**Summe**	**94.020,00**	**40.000,00**	**94.020,02**
3	Lehmann	49.000,00		-
4	Schulze	6.000,00	6.000,00	14.103,00
5	Schultze	5.750,00	5.750,00	13.515,38
6	Schmidt	5.500,00	5.500,00	12.927,75
7	Schmitt	5.270,00	5.250,00	12.340,13
8	Müller	5.000,00		-
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

You can delete the budgets of leavers easily in column C. The order of deletions does not matter.

The obvious disadvantage of this approach is a potential rounding difference, because the sum of rounded values is not necessary equal to the rounded sum of not-rounded summands. The example above shows a difference of 0.02.

A correct Calculation

With the user defined function RoundToSum you can use the matrix formula *{=RoundToSum(C4:C13*$B$3/$C$3,D1)}*:



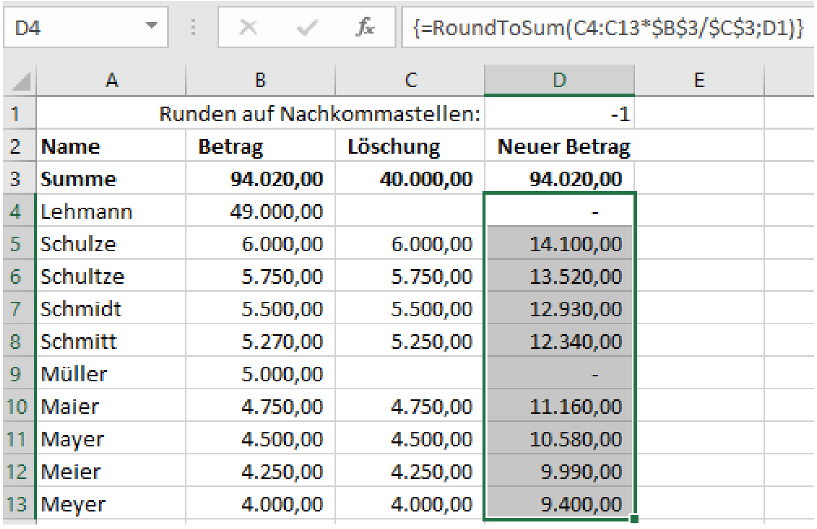
D4

{=RoundToSum(C4:C13*$B$3/$C$3;D1)}

	A	B	C	D	E
1		Runden auf Nachkommastellen:		2	
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	14.103,00	
6	Schultze	5.750,00	5.750,00	13.515,37	
7	Schmidt	5.500,00	5.500,00	12.927,75	
8	Schmitt	5.270,00	5.250,00	12.340,12	
9	Müller	5.000,00		-	
10	Maier	4.750,00	4.750,00	11.164,88	
11	Mayer	4.500,00	4.500,00	10.577,25	
12	Meier	4.250,00	4.250,00	9.989,63	
13	Meyer	4.000,00	4.000,00	9.402,00	

RoundToSum sometime needs to round to the ‘wrong’ side but then it ensures a minimal error.

In addition to that you can round to other decimals with *RoundToSum*, for example to 10s:



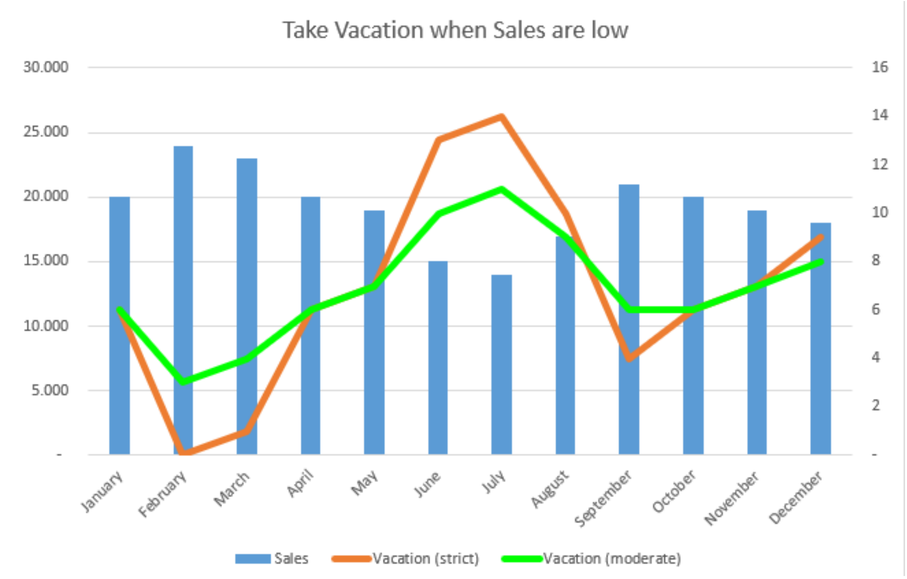
```
{=RoundToSum(C4:C13*$B$3/$C$3;D1)}
```

	A	B	C	D	E
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	

Please notice here, that *RoundToSum* cannot create the original sum if it is not rounded to the number of desired decimals.

Take Vacation When Less is Going on

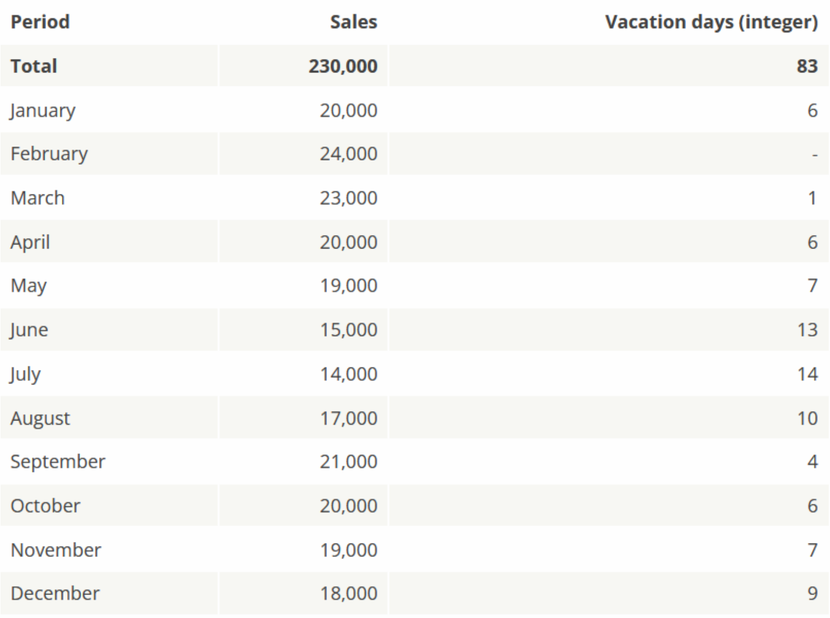
If your business fluctuates strongly seasonally, you can plan the vacation of your staff accordingly and consider hiring seasonal staff:



Note: Of course you cannot force anybody when to take a vacation and how many days are to be taken. These calculations are just meant to be suggestions of reasonable indicators.

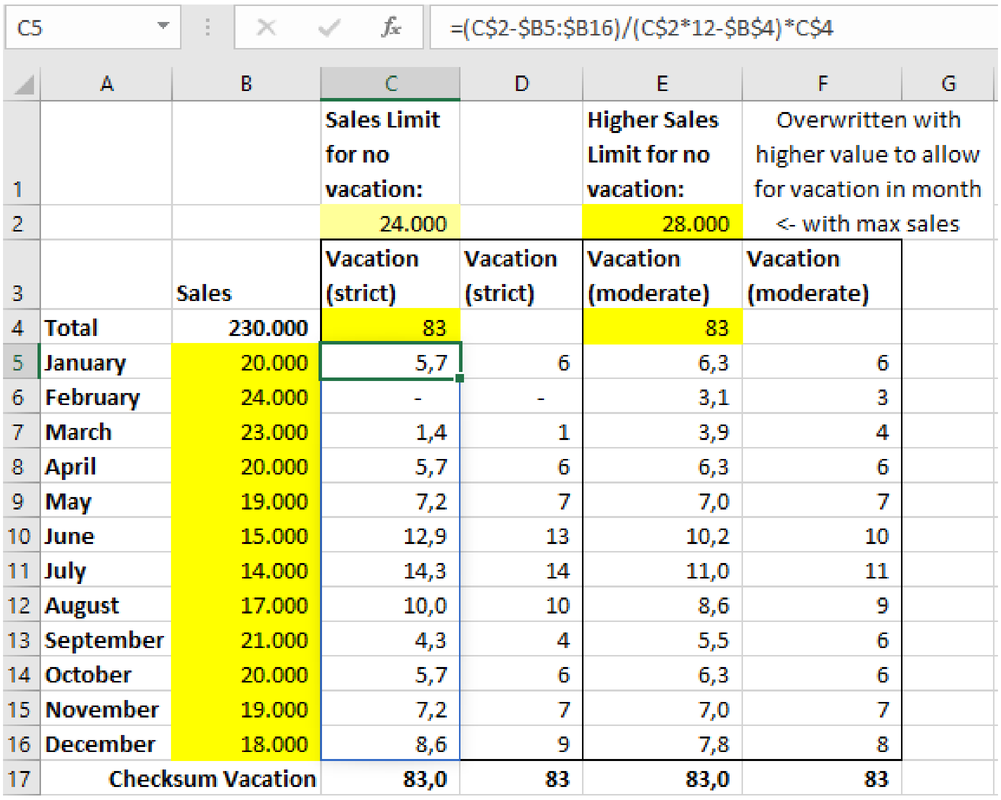
Simple Example

If you like to take the maximum sales values (here: 24,000) as a basis, applying zero vacations to it, and scale the vacation days linearly to the other sales values:



Period	Sales	Vacation days (integer)
Total	**230,000**	**83**
January	20,000	6
February	24,000	-
March	23,000	1
April	20,000	6
May	19,000	7
June	15,000	13
July	14,000	14
August	17,000	10
September	21,000	4
October	20,000	6
November	19,000	7
December	18,000	9

The formula in cell C5 which spills down to B16 is *=($C$2-B5:B16)/($C$2*12-$B$4)*$C$4*:



C5 | =(C$2-$B5:$B16)/(C$2*12-$B$4)*C$4

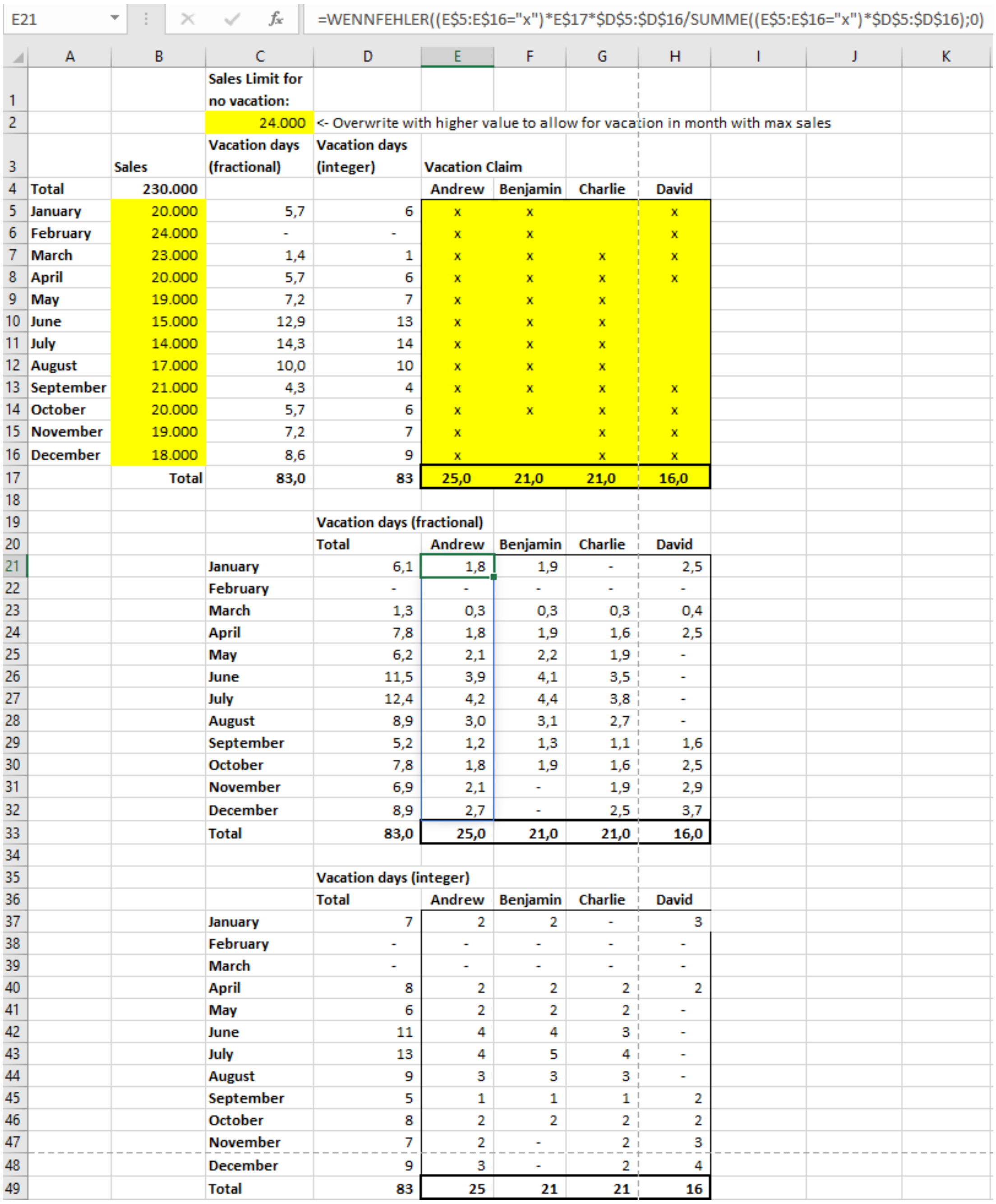
	A	B	C	D	E	F	G
1			Sales Limit for no vacation:		Higher Sales Limit for no vacation:	Overwritten with higher value to allow for vacation in month	
2			24.000		28.000	<- with max sales	
3		Sales	Vacation (strict)	Vacation (strict)	Vacation (moderate)	Vacation (moderate)	
4	Total	230.000	83		83		
5	January	20.000	5,7	6	6,3	6	
6	February	24.000	-	-	3,1	3	
7	March	23.000	1,4	1	3,9	4	
8	April	20.000	5,7	6	6,3	6	
9	May	19.000	7,2	7	7,0	7	
10	June	15.000	12,9	13	10,2	10	
11	July	14.000	14,3	14	11,0	11	
12	August	17.000	10,0	10	8,6	9	
13	September	21.000	4,3	4	5,5	6	
14	October	20.000	5,7	6	6,3	6	
15	November	19.000	7,2	7	7,0	7	
16	December	18.000	8,6	9	7,8	8	
17	Checksum Vacation		83,0	83	83,0	83	

More Complex Example

If you got employees who are not present at specified months:

Formula in cell E21 reads:
*=IFERROR((E$5:E$16="x")*E$17*$D$5:$D$16/SUM((E$5:E$16="x")*$D$5:$D$16),0)*

Formula in Cell E37:
=TRANSPOSE(IFERROR(RoundToSum(E21:E32,0),0))



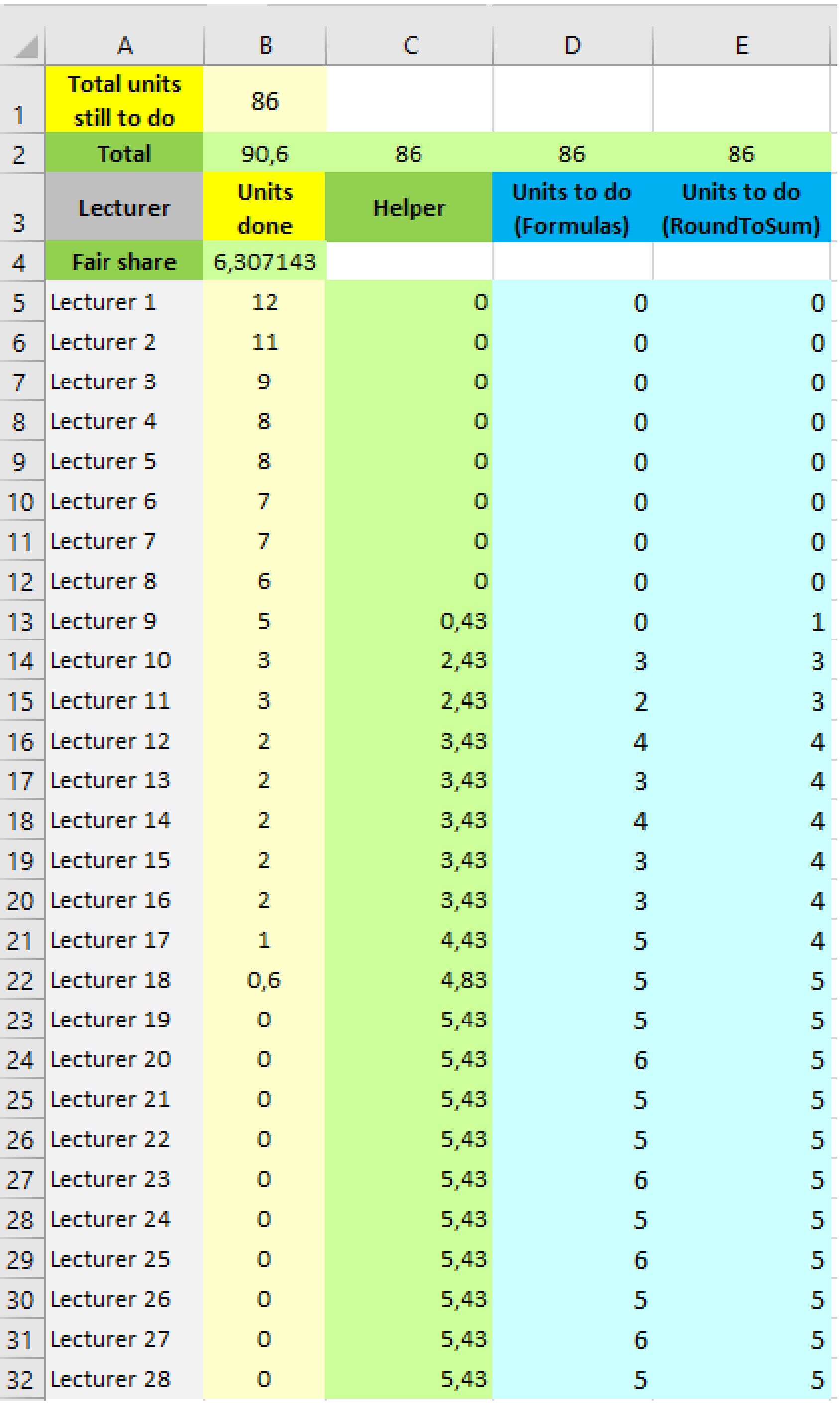
E21

=WENNFEHLER((E$5:E$16="x")*E$17*$D$5:$D$16/SUMME((E$5:E$16="x")*$D$5:$D$16);0)

	A	B	C	D	E	F	G	H	I	J	K
1			Sales Limit for no vacation:								
2			24.000	<- Overwrite with higher value to allow for vacation in month with max sales							
3		Sales	Vacation days (fractional)	Vacation days (integer)	Vacation Claim						
4	Total	230.000			Andrew	Benjamin	Charlie	David			
5	January	20.000	5,7	6	x	x		x			
6	February	24.000	-	-	x	x		x			
7	March	23.000	1,4	1	x	x	x	x			
8	April	20.000	5,7	6	x	x	x	x			
9	May	19.000	7,2	7	x	x	x				
10	June	15.000	12,9	13	x	x	x				
11	July	14.000	14,3	14	x	x	x				
12	August	17.000	10,0	10	x	x	x				
13	September	21.000	4,3	4	x	x	x	x			
14	October	20.000	5,7	6	x	x	x	x			
15	November	19.000	7,2	7	x		x	x			
16	December	18.000	8,6	9	x		x	x			
17		Total	83,0	83	25,0	21,0	21,0	16,0			
18											
19				Vacation days (fractional)							
20				Total	Andrew	Benjamin	Charlie	David			
21			January	6,1	1,8	1,9	-	2,5			
22			February	-	-	-	-	-			
23			March	1,3	0,3	0,3	0,3	0,4			
24			April	7,8	1,8	1,9	1,6	2,5			
25			May	6,2	2,1	2,2	1,9	-			
26			June	11,5	3,9	4,1	3,5	-			
27			July	12,4	4,2	4,4	3,8	-			
28			August	8,9	3,0	3,1	2,7	-			
29			September	5,2	1,2	1,3	1,1	1,6			
30			October	7,8	1,8	1,9	1,6	2,5			
31			November	6,9	2,1	-	1,9	2,9			
32			December	8,9	2,7	-	2,5	3,7			
33			Total	83,0	25,0	21,0	21,0	16,0			
34											
35				Vacation days (integer)							
36				Total	Andrew	Benjamin	Charlie	David			
37			January	7	2	2	-	3			
38			February	-	-	-	-	-			
39			March	-	-	-	-	-			
40			April	8	2	2	2	2			
41			May	6	2	2	2	-			
42			June	11	4	4	3	-			
43			July	13	4	5	4	-			
44			August	9	3	3	3	-			
45			September	5	1	1	1	2			
46			October	8	2	2	2	2			
47			November	7	2	-	2	3			
48			December	9	3	-	2	4			
49			Total	83	25	21	21	16			

Assign Work Units Adjusted by Delivered Output

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



	A	B	C	D	E
1	Total units still to do	86			
2	Total	90,6	86	86	86
3	Lecturer	Units done	Helper	Units to do (Formulas)	Units to do (RoundToSum)
4	Fair share	6,307143			
5	Lecturer 1	12	0	0	0
6	Lecturer 2	11	0	0	0
7	Lecturer 3	9	0	0	0
8	Lecturer 4	8	0	0	0
9	Lecturer 5	8	0	0	0
10	Lecturer 6	7	0	0	0
11	Lecturer 7	7	0	0	0
12	Lecturer 8	6	0	0	0
13	Lecturer 9	5	0,43	0	1
14	Lecturer 10	3	2,43	3	3
15	Lecturer 11	3	2,43	2	3
16	Lecturer 12	2	3,43	4	4
17	Lecturer 13	2	3,43	3	4
18	Lecturer 14	2	3,43	4	4
19	Lecturer 15	2	3,43	3	4
20	Lecturer 16	2	3,43	3	4
21	Lecturer 17	1	4,43	5	4
22	Lecturer 18	0,6	4,83	5	5
23	Lecturer 19	0	5,43	5	5
24	Lecturer 20	0	5,43	6	5
25	Lecturer 21	0	5,43	5	5
26	Lecturer 22	0	5,43	5	5
27	Lecturer 23	0	5,43	6	5
28	Lecturer 24	0	5,43	5	5
29	Lecturer 25	0	5,43	6	5
30	Lecturer 26	0	5,43	5	5
31	Lecturer 27	0	5,43	6	5
32	Lecturer 28	0	5,43	5	5

Yellow cells show input values, green ones are intermediate or helper cells, and blue cells mark final output values. Note: You need to enter 'Units done' in descending order.

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

The key worksheet formula is in cell C5:
=MAX(0,B$4-B5-SUMPRODUCT(--(C$4:C4=0),B$4:B4-B$4)/(ROWS(B$5:B$32)-SUMPRODUCT(--(C$4:C4=0))+1))

Please notice that the fair share has been put into cell B4 intentionally and C4 has been kept empty, so that this formula could be just entered into C5 and copied down.

Column C shows the fractional results. In column D a simple worksheet function approach has been applied to round values of column C to integers, preserving their original sum.

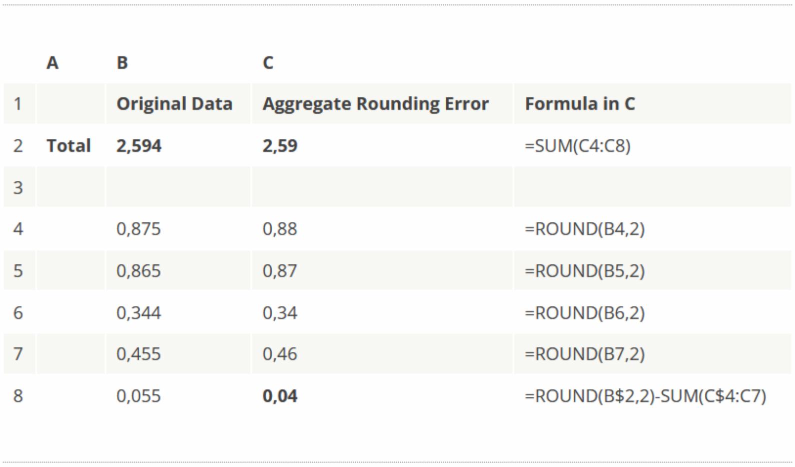
As you can easily see, column E shows smoother results using the user defined function *RoundToSum*.

RoundToSum Versus Other Methods

RoundToSum Versus Other “Simple” Methods

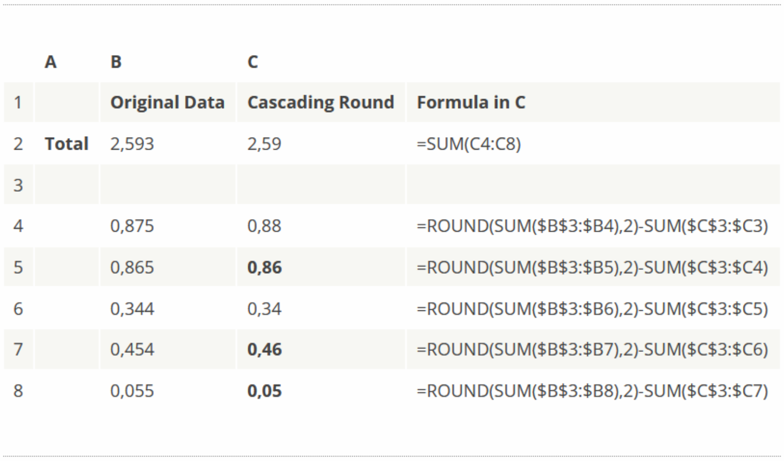
There are several different naïve approaches circulating around which try to round values preserving their rounded sum:

- (worst) Round all values but the last one and replace the last one by the rounded original sum minus the sum of the previously rounded values (i.e. aggregate all rounding errors in the last summand):



	A	B	C	
1		**Original Data**	**Aggregate Rounding Error**	**Formula in C**
2	**Total**	**2,594**	**2,59**	=SUM(C4:C8)
3				
4		0,875	0,88	=ROUND(B4,2)
5		0,865	0,87	=ROUND(B5,2)
6		0,344	0,34	=ROUND(B6,2)
7		0,455	0,46	=ROUND(B7,2)
8		0,055	**0,04**	=ROUND(B$2,2)-SUM(C$4:C7)

- (better, but still bad) Apply a cascading (sliding) round:

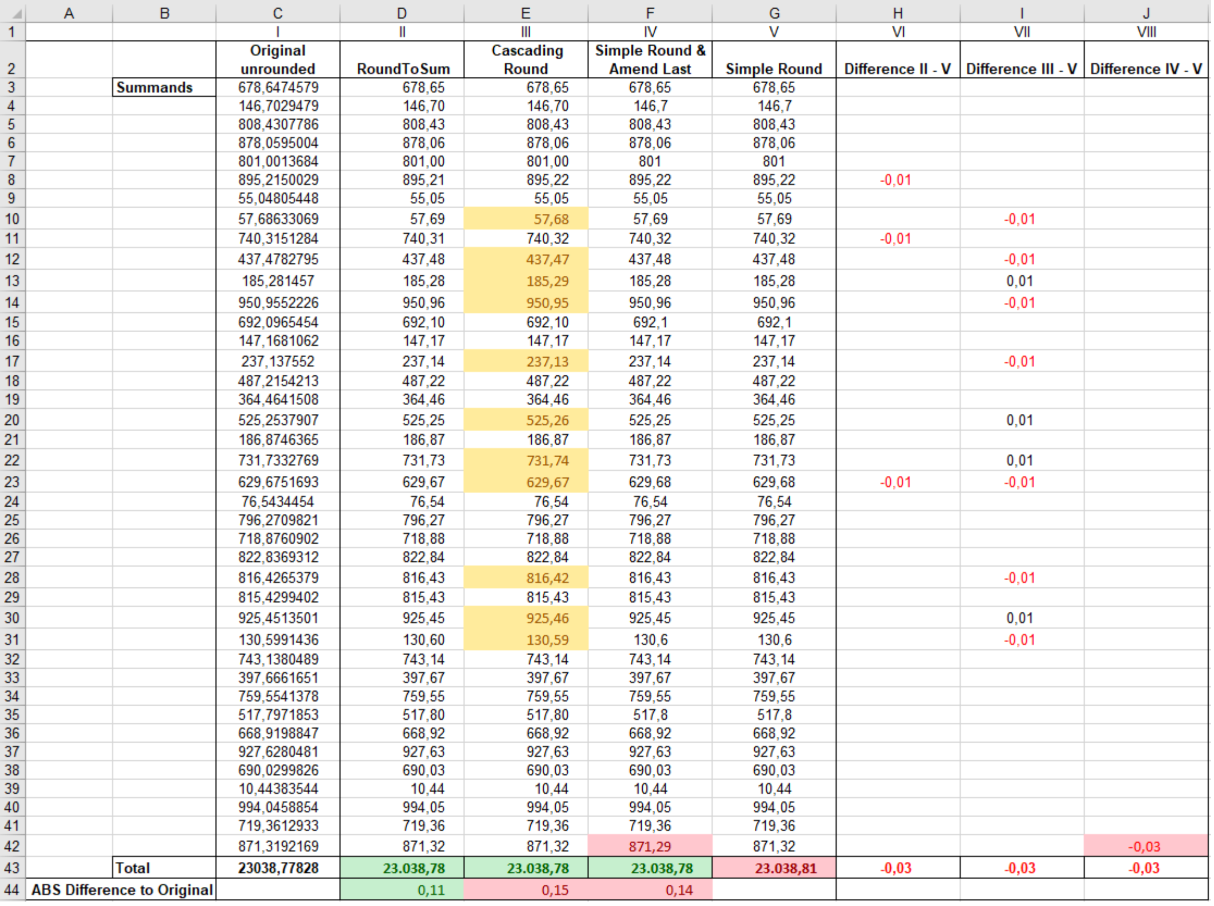


	A	B	C	
1		**Original Data**	**Cascading Round**	**Formula in C**
2	**Total**	2,593	2,59	=SUM(C4:C8)
3				
4		0,875	0,88	=ROUND(SUM($B$3:$B4),2)-SUM($C$3:$C3)
5		0,865	**0,86**	=ROUND(SUM($B$3:$B5),2)-SUM($C$3:$C4)
6		0,344	0,34	=ROUND(SUM($B$3:$B6),2)-SUM($C$3:$C5)
7		0,454	**0,46**	=ROUND(SUM($B$3:$B7),2)-SUM($C$3:$C6)
8		0,055	**0,05**	=ROUND(SUM($B$3:$B8),2)-SUM($C$3:$C7)

Let us compare these approaches to *RoundToSum*.

Calculation Example

We create 40 random numbers *RAND() * 1000* and compare as follows:



	A	B	C	D	E	F	G	H	I	J
1			I	II	III	IV	V	VI	VII	VIII
2			Original unrounded	RoundToSum	Cascading Round	Simple Round & Amend Last	Simple Round	Difference II - V	Difference III - V	Difference IV - V
3		Summands	678,6474579	678,65	678,65	678,65	678,65			
4			146,7029479	146,70	146,70	146,7	146,7			
5			808,4307786	808,43	808,43	808,43	808,43			
6			878,0595004	878,06	878,06	878,06	878,06			
7			801,0013684	801,00	801,00	801	801			
8			895,2150029	895,21	895,22	895,22	895,22	-0,01		
9			55,04805448	55,05	55,05	55,05	55,05			
10			57,68633069	57,69	57,68	57,69	57,69		-0,01	
11			740,3151284	740,31	740,32	740,32	740,32	-0,01		
12			437,4782795	437,48	437,47	437,48	437,48		-0,01	
13			185,281457	185,28	185,29	185,28	185,28		0,01	
14			950,9552226	950,96	950,95	950,96	950,96		-0,01	
15			692,0965454	692,10	692,10	692,1	692,1			
16			147,1681062	147,17	147,17	147,17	147,17			
17			237,137552	237,14	237,13	237,14	237,14		-0,01	
18			487,2154213	487,22	487,22	487,22	487,22			
19			364,4641508	364,46	364,46	364,46	364,46			
20			525,2537907	525,25	525,26	525,25	525,25		0,01	
21			186,8746365	186,87	186,87	186,87	186,87			
22			731,7332769	731,73	731,74	731,73	731,73		0,01	
23			629,6751693	629,67	629,67	629,68	629,68	-0,01	-0,01	
24			76,5434454	76,54	76,54	76,54	76,54			
25			796,2709821	796,27	796,27	796,27	796,27			
26			718,8760902	718,88	718,88	718,88	718,88			
27			822,8369312	822,84	822,84	822,84	822,84			
28			816,4265379	816,43	816,42	816,43	816,43		-0,01	
29			815,4299402	815,43	815,43	815,43	815,43			
30			925,4513501	925,45	925,46	925,45	925,45		0,01	
31			130,5991436	130,60	130,59	130,6	130,6		-0,01	
32			743,1380489	743,14	743,14	743,14	743,14			
33			397,6661651	397,67	397,67	397,67	397,67			
34			759,5541378	759,55	759,55	759,55	759,55			
35			517,7971853	517,80	517,80	517,8	517,8			
36			668,9198847	668,92	668,92	668,92	668,92			
37			927,6280481	927,63	927,63	927,63	927,63			
38			690,0299826	690,03	690,03	690,03	690,03			
39			10,44383544	10,44	10,44	10,44	10,44			
40			994,0458854	994,05	994,05	994,05	994,05			
41			719,3612933	719,36	719,36	719,36	719,36			
42			871,3192169	871,32	871,32	871,29	871,32			-0,03
43		Total	23038,77828	23.038,78	23.038,78	23.038,78	23.038,81	-0,03	-0,03	-0,03
44	ABS Difference to Original			0,11	0,15	0,14				

Please note that the formula in C3 for the cascading round includes the title row so that it can be copied down.

As you can see, if we simply round each single number, the resulting sum would differ from the original rounded sum by -0.03. Column J (VIII) shows the difference of the aggregated rounding error -0.03 in the last summand. Column F (IV) shows the corresponding rounded numbers. Worst case would be here to come up with an aggregated rounding error of n * 0,005 with n being the count of your numbers. Example: Take 40 times the number 0.005 instead of the 40 random numbers.

Good practical examples, why you should not aggregate rounding errors in the last summand, are normally distributed samples of integers.

The cascading (sliding) round in column I (VII) shows 11 roundings to the wrong side. Column E (III) shows the corresponding rounded numbers. Worst case would be for the cascading round to round half of your numbers to the wrong side when all numbers could have been rounded correctly. Example: Take 20 times the number -0.0049999 and then 20 times the number 0.0049999 instead of the 40 random numbers.

On the other hand, the optimal RoundToSum just rounds 3 values to the wrong side which result in the least number of changes which achieve the correct rounded sum. The worst case would now involve n/2 roundings to the wrong side with n being the count of your numbers. Example: Take 40 times the number 0.005 again instead of the 40 random numbers. This is the best solution with the smallest absolute rounding error for each number and then with the smallest number of roundings to the wrong side.

Conclusion

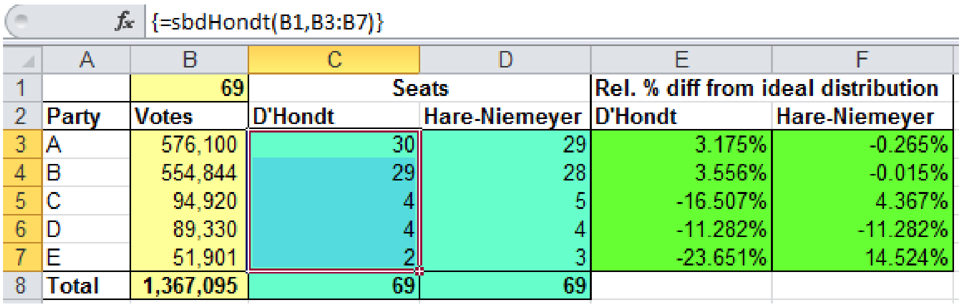
Use RoundToSum. It will apply the least number of changes and it will result in the correct sum with the smallest absolute (or relative) error. It needs to be applied with an array formula because n > 1 output values depend on n > 1 input values.

A cascading round as shown above, i.e. using in cell E6 the formula *=ROUND(SUM($C$5:$C5),2)-SUM($E$4:$E4)* does not need any VBA nor does it apply any array formula, but it can leave you with a fairly high number of unnatural roundings which you can hardly explain to any senior manager.

But worst of all is the stupid approach of aggregating all rounding differences in the last summand. Just imagine 1,000 people, each having 49 Cents, adding up to $490, which you should distribute fairly, but rounded to a whole Dollar. In this case you would end up with $490 at the last person, while *RoundToSum* would give the first 490 persons one Dollar each and all the others zero.

RoundToSum Compared to sbDHondt

RoundToSum implements the Hare-Niemeyer approach. In the context of distributing parliamentary seats, this method is superior to the D'Hondt approach. One key advantage is that the relative percentage difference from an ideal proportional distribution is generally lower, as illustrated by the following example:



{=sbdHondt(B1,B3:B7)}

	A	B	C	D	E	F
1		69	Seats		Rel. % diff from ideal distribution	
2	Party	Votes	D'Hondt	Hare-Niemeyer	D'Hondt	Hare-Niemeyer
3	A	576,100	30	29	3.175%	-0.265%
4	B	554,844	29	28	3.556%	-0.015%
5	C	94,920	4	5	-16.507%	4.367%
6	D	89,330	4	4	-11.282%	-11.282%
7	E	51,901	2	3	-23.651%	14.524%
8	Total	1,367,095	69	69		

sbDHondt Program Code

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

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

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

Literature

Diaconis, P., & Freedman, D. (13. Juli 2007), On Rounding Percentages.

Sande, G. (2005, August 7), Guaranteed Controlled Rounding for Many Totals in Multi-way and Hierarchical Tables.