Option Explicit

Option Base 1

'-----------------------------------------------------------------------------------------------------------------------------------

'Floating-point arithmetic may give inaccurate results in Excel

Private Const PUB\_MIN\_REAL\_VAL As Double = 1e-300

Private Const PUB\_DEFAULT\_TOLERANCE\_VAL As Double = 1e-14

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'When does 35 not equal 35?

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'References:

'http://blogs.office.com/b/microsoft-excel/archive/2008/04/10/understanding-floating-point-precision-aka-why-does-excel-give-me-seemingly-wrong-answers.aspx

'http://support.microsoft.com/kb/78113

'-----------------------------------------------------------------------------------------------------------------------------------

'When does 35 \_ 35 not equal zero? In the first case the answer is when one \_35? is on an Excel spreadsheet and the other 35 is in VBA.

'In this section we will generate a series of output points by calculating the length of one step (35/30) then adding this value to the

'previous length in 30 stages. This mysteriously will cause an error in the function for no obvious reason.

'Stepping through the code you may find that the error is occurring when the last output point is compared to the beam length and it is

'greater. The mysterious part is that both on the worksheet (formatted to 15 decimal places), and in the VBA Locals window, both

'values will display as being exactly 35. Also entering the formula = B36 \_ 35 returned exactly zero.

'To investigate what is going on lets write a short VBA function shown below:

Function DIFF\_FUNC(T1\_VAL As Double, T2\_VAL As Double)

DIFF\_FUNC = T1\_VAL - T2\_VAL

End Function

'The function simply subtracts B from A, but using this function with the incrementally generated \_35? as A and the \_exactÓ 35 as B

'returned a value of 7.11E-15, even though stepping through the routine both variables displayed as exactly 35 in the locals window.

'The answer to the second question is the same as the first (as would be expected), but also in some cases this behaviour can be seen

'on the spreadsheet with no VBA involved. As stated above, the formula = B36 - 35 returns exactly zero, but the formula = (B36 \_ 35)

'returns 7.11E-15, the same as the UDF! The subroutine "TEST\_DIFF\_FUNC" below show more detail of this behaviour.

Sub TEST\_DIFF\_FUNC() 'subroutine

Dim i As Long

Dim SROW As Long

Dim SCOLUMN As Long

Dim FORMULA\_STR As String

Dim SRC\_WSHEET As Worksheet

Set SRC\_WSHEET = Worksheets("Section2")

SROW = 1: SCOLUMN = 1

With SRC\_WSHEET

With .Cells

.Clear

.ColumnWidth = 10

End With

'In the spreadsheet in column B it generates 30 increments with the formula =$B$3+B6, where $B$3 is =$B$1/$B$2, i.e. 35/30.

'The value in column C are generated with =$B$1\*A7/$B$2, where $B$1 is the beam length, A7 is the increment number, and

'$B$2 is the number of increments (30). Column D contains the the formula =B7 - C7, Column E: =Diff(B7, C7), and column F: =(B7 - C7).

.Cells(SROW + 0, SCOLUMN + 0) = "Length"

.Cells(SROW + 0, SCOLUMN + 1) = 35

.Cells(SROW + 1, SCOLUMN + 0) = "Divisions"

.Cells(SROW + 1, SCOLUMN + 1) = 30

.Cells(SROW + 2, SCOLUMN + 0) = "Division Len"

.Cells(SROW + 2, SCOLUMN + 1).Formula = "=" & .Cells(SROW + 0, SCOLUMN + 1).Address & "/" & .Cells(SROW + 1, SCOLUMN + 1).Address

.Cells(SROW + 4, SCOLUMN + 0) = "Inc No"

.Cells(SROW + 4, SCOLUMN + 1) = "Step Increment"

.Cells(SROW + 4, SCOLUMN + 2) = "Factor"

.Cells(SROW + 4, SCOLUMN + 3) = "Col B - Col C"

.Cells(SROW + 4, SCOLUMN + 4) = "Diff (Col B, Col C)"

.Cells(SROW + 4, SCOLUMN + 5) = "(Col B - Col C)"

For i = 1 To 30

.Cells(SROW + 5 + i, SCOLUMN + 0).Formula = "=1+" & \_

.Cells(SROW + 5 + i - 1, SCOLUMN + 0).Address(False, False)

.Cells(SROW + 5 + i, SCOLUMN + 1).Formula = "=" & \_

.Cells(SROW + 2, SCOLUMN + 1).Address & "+" & \_

.Cells(SROW + 5 + i - 1, SCOLUMN + 1).Address(False, False)

.Cells(SROW + 5 + i, SCOLUMN + 2).Formula = "=" & \_

.Cells(SROW + 0, SCOLUMN + 1).Address & "\*" & \_

.Cells(SROW + 5 + i, SCOLUMN + 0).Address(False, False) & "/" & \_

.Cells(SROW + 1, SCOLUMN + 1).Address

.Cells(SROW + 5 + i, SCOLUMN + 3).Formula = "=" & \_

.Cells(SROW + 5 + i, SCOLUMN + 1).Address(False, False) & "-" & \_

.Cells(SROW + 5 + i, SCOLUMN + 2).Address(False, False)

If DIFF\_FUNC(.Cells(SROW + 5 + i, SCOLUMN + 1).Value, .Cells(SROW + 5 + i, SCOLUMN + 2)) <> 0 Then

Debug.Print i, Format(.Cells(SROW + 5 + i, SCOLUMN + 1).Value, "0.000000000000000"), \_

Format(.Cells(SROW + 5 + i, SCOLUMN + 2).Value, "0.000000000000000"), \_

DIFF\_FUNC(.Cells(SROW + 5 + i, SCOLUMN + 1).Value, .Cells(SROW + 5 + i, SCOLUMN + 2))

End If

.Cells(SROW + 5 + i, SCOLUMN + 4).Formula = "=DIFF\_FUNC(" & \_

.Cells(SROW + 5 + i, SCOLUMN + 1).Address(False, False) & "," & \_

.Cells(SROW + 5 + i, SCOLUMN + 2).Address(False, False) & ")"

.Cells(SROW + 5 + i, SCOLUMN + 5).Formula = "=(" & \_

.Cells(SROW + 5 + i, SCOLUMN + 1).Address(False, False) & "-" & \_

.Cells(SROW + 5 + i, SCOLUMN + 2).Address(False, False) & ")"

Next i

End With

End Sub

'It can be seen that Column D has returned a difference of zero in all cases, even though the VBA function in Column E and the

'formula in Column F show a difference of up to 1.42E-14.

'These differences are of course caused in part by the fact that all values are stored as binary floating point values, that

'cannot represent all decimal or fractional values exactly. This is described in several Microsoft documents (e.g.

'Understanding Floating Point Precision), which all claim that Excel follows the IEEE Standard for Binary Floating-Point

'Arithmetic. Clearly this is not the whole story though, since the example given generates errors up to an order of magnitude

'greater than the maximum difference between any exact value and the nearest floating point value, and also generates different

'results depending on how worksheet formulas are entered.

'It would be interested if anyone has any more background on exactly how these things are handled in Excel, but for practical

'purposes I think the lessons are:

'i) When comparing non-integer values be aware that values that display as exactly equal may be stored as different values.

'ii) When comparing doubles in VBA either round the values to a suitable precision, or check that the difference is less than

'some small value, rather than exactly zero.

'iii) Calculations that involve the difference between two nearly equal values may give incorrect results. Consider carrying

'out this type of calculation entirely in VBA (or if necessary in a different language offering higher precision calculations).

Sub TEST\_EFFECT\_OF\_BRACKETS()

Dim B2\_VAL As Double

Dim B3\_VAL As Double

Dim B4\_VAL As Double

Dim B5\_VAL As Double

Dim B6\_VAL As Double

Dim B7\_VAL As Double

Dim B8\_VAL As Double

Dim B9\_VAL As Double

Dim SIG\_STR As String

SIG\_STR = "0.000000000000000000"

B2\_VAL = 1.6e-15

B3\_VAL = 1

B4\_VAL = B2\_VAL + B3\_VAL

'displays as exactly 1, because the 1.6E-15 would be the 16th and 17th significant figure, but only 15 significant figures are displayed

Debug.Print Format(B4\_VAL, SIG\_STR) '1.000000000000000000

B5\_VAL = B4\_VAL - 1

'displays as 0.000000000000001554, using the decimal equivalent of the underlying value stored in B4, rather than the displayed value

Debug.Print Format(B5\_VAL, SIG\_STR) '0.000000000000001554

B6\_VAL = (B2\_VAL + B3\_VAL)

'displays as exactly 1, the same as B4, as would be expected.

Debug.Print Format(B6\_VAL, SIG\_STR) '1.000000000000000

B7\_VAL = B6\_VAL - 1

'displays as 0.000000000000001554, using the decimal equivalent of the underlying value stored in B6, rather than the displayed value

Debug.Print Format(B7\_VAL, SIG\_STR) '0.000000000000001554

B8\_VAL = (B6\_VAL - B4\_VAL)

'displays exactly 0, using the displayed values in both cells

Debug.Print Format(B8\_VAL, SIG\_STR) '0.000000000000000000

B9\_VAL = B7\_VAL - B5\_VAL

'displays exactly 0, using the displayed values in both cells

Debug.Print Format(B9\_VAL, SIG\_STR) '0.000000000000000000

'If it does not display zero and rather the same value as B7), then ((1+x))-(1+x) <> ((1+x)-1)-(1+x-1)!!!!

End Sub

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Example: Comparing floating point numbers

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Some of the issues raised in the previous section arise when comparing floating point numbers.

'Private Const PUB\_MIN\_REAL\_VAL As Double = 1E-300

'Private Const PUB\_DEFAULT\_TOLERANCE\_VAL As Double = 0.00000000000001

Sub Test\_Comparing\_Floating\_Point\_Numbers()

Dim i As Long

Dim j As Long

Dim k As Long

Dim SROW As Long

Dim SCOLUMN As Long

Dim NCOLUMNS As Long

Dim NROWS As Long

Dim T1\_STR As String

Dim T2\_STR As String

Dim TOL\_STR As String

Dim FORMULAS\_STR As String

Dim ARR1\_RNG As Range

Dim ARR2\_RNG As Range

Dim SRC\_WSHEET As Worksheet

Set SRC\_WSHEET = Worksheets("Section2")

SROW = 1: SCOLUMN = 1

With SRC\_WSHEET

With .Cells

.Clear

.ColumnWidth = 10

End With

With .Cells(SROW + 0, SCOLUMN)

.Value = "Floating point comparison functions"

.Font.Bold = True

.Font.Italic = True

End With

.Cells(SROW + 1, SCOLUMN) = "EQUALT\_FUNC(T1, T2, Tol) -> Test if T1\_VAL is equal to T2\_VAL within tolerance --> True if Abs(T1 / T2 - 1) < Tol"

.Cells(SROW + 2, SCOLUMN) = "LTEQUALT\_FUNC(T1, T2, Tol) -> Test if T1\_VAL is less than or equal to T2\_VAL within tolerance --> True if T1-T2 < 0 or Abs(T2 / T1 - 1) < Tol"

.Cells(SROW + 3, SCOLUMN) = "GTEQUALT\_FUNC(T1, T2, Tol) -> Test if T1\_VAL is greater than or equal to T2\_VAL within tolerance --> True if T1-T2 > 0 or Abs(T1 / T2 - 1) < Tol"

.Cells(SROW + 4, SCOLUMN) = "LTT\_FUNC(T1, T2) -> Test if T1\_VAL is at least tolerance less than T2\_VAL"

.Cells(SROW + 5, SCOLUMN) = "GTT\_FUNC(T1, T2) -> Test if T1\_VAL is at least tolerance greater than T2\_VAL"

FORMULAS\_STR = "T1|T2|EQUALT|LTEQUALT|GTEQUALT|LTT|GTT|Diff|Tol|"

NCOLUMNS = 9

i = 1

For k = 1 To NCOLUMNS

j = InStr(i, FORMULAS\_STR, "|")

.Cells(SROW + 7, SCOLUMN + k - 1).Value = Mid(FORMULAS\_STR, i, j - i)

i = j + 1

Next k

For i = 1 To 3

.Cells(SROW + 7 + i, SCOLUMN + 0).Value = i + 3

.Cells(SROW + 7 + i, SCOLUMN + 1).Value = 5

Next i

For i = 4 To 5

.Cells(SROW + 7 + i, SCOLUMN + 0).Value = 5

.Cells(SROW + 7 + i, SCOLUMN + 1).Formula = "=5+" & .Cells(SROW + 7 + i, SCOLUMN + 1).Offset(0, 6).Address(False, False)

.Cells(SROW + 7 + i, SCOLUMN + 1).Offset(0, 6).Value = 1e-12

If i = 5 Then

.Cells(SROW + 7 + i, SCOLUMN + 1).Offset(0, 7).Value = 1e-12

End If

Next i

i = 6

.Cells(SROW + 7 + i, SCOLUMN + 0).Value = 1e-306

.Cells(SROW + 7 + i, SCOLUMN + 1).Value = 1e-307

For i = 1 To 6

T1\_STR = .Cells(SROW + 7 + i, SCOLUMN + 2).Offset(0, -2).Address(False, False)

T2\_STR = .Cells(SROW + 7 + i, SCOLUMN + 2).Offset(0, -1).Address(False, False)

TOL\_STR = .Cells(SROW + 7 + i, SCOLUMN + 2).Offset(0, 6).Address(False, False)

If i <= 4 Then

.Cells(SROW + 7 + i, SCOLUMN + 2).Formula = "=EQUALT\_FUNC(" & T1\_STR & "," & T2\_STR & ")"

.Cells(SROW + 7 + i, SCOLUMN + 5).Formula = "=LTT\_FUNC(" & T1\_STR & "," & T2\_STR & ")"

.Cells(SROW + 7 + i, SCOLUMN + 6).Formula = "=GTT\_FUNC(" & T1\_STR & "," & T2\_STR & ")"

Else

.Cells(SROW + 7 + i, SCOLUMN + 2).Formula = "=EQUALT\_FUNC(" & T1\_STR & "," & T2\_STR & "," & TOL\_STR & ")"

.Cells(SROW + 7 + i, SCOLUMN + 5).Formula = "=LTT\_FUNC(" & T1\_STR & "," & T2\_STR & "," & TOL\_STR & ")"

.Cells(SROW + 7 + i, SCOLUMN + 6).Formula = "=GTT\_FUNC(" & T1\_STR & "," & T2\_STR & "," & TOL\_STR & ")"

End If

.Cells(SROW + 7 + i, SCOLUMN + 3).Formula = "=LTEQUALT\_FUNC(" & T1\_STR & "," & T2\_STR & ")"

.Cells(SROW + 7 + i, SCOLUMN + 4).Formula = "=GTEQUALT\_FUNC(" & T1\_STR & "," & T2\_STR & ")"

Next i

End With

End Sub

'Note that the on sheet formulas return values a 5 from the first array, and just over 5 from the second, whereas

'for the UDF results the tolerance has been set high enough for these two values to be treated as equal.

Function DIFF\_RATIO\_FUNC(T1\_VAL As Variant, \_

T2\_VAL As Variant) As Double

Dim N\_VAL As Double

Dim D\_VAL As Double

If Abs(T1\_VAL) > Abs(T2\_VAL) Then

D\_VAL = T1\_VAL

N\_VAL = T2\_VAL

Else

D\_VAL = T2\_VAL

N\_VAL = T1\_VAL

End If

DIFF\_RATIO\_FUNC = Abs((N\_VAL / D\_VAL) - 1)

'The basis of this function is to find the value abs(N/D -1), where D is the lesser of the two values, and N is the greater.

'Is this value is less than the specified tolerance then the two values are treted as being equal. If the tolerance is not

'specified it defaults to 0.00000000000001 = 1E-14 = 10^(-14).

'The use of a relative tolerance raises a problem if both numbers are very close to the minimum values allowed by the floating

'point number system. For this reason the numbers are treated as being equal if the absolute value of the difference is less

'than a constant, PUB\_MIN\_REAL\_VAL, currently set to 1E-300.

End Function

' Test if T1\_VAL is equal to T2\_VAL within tolerance

Function EQUALT\_FUNC(T1\_VAL As Variant, \_

T2\_VAL As Variant, \_

Optional tolerance As Double = PUB\_DEFAULT\_TOLERANCE\_VAL) As Boolean

If Abs(T1\_VAL - T2\_VAL) <= PUB\_MIN\_REAL\_VAL Then

EQUALT\_FUNC = True

Exit Function

End If

If DIFF\_RATIO\_FUNC(T1\_VAL, T2\_VAL) < tolerance Then

EQUALT\_FUNC = True

Else

EQUALT\_FUNC = False

End If

End Function

' Test if T1\_VAL is less than or equal to T2\_VAL within tolerance

Function LTEQUALT\_FUNC(T1\_VAL As Variant, \_

T2\_VAL As Variant, \_

Optional tolerance As Double = PUB\_DEFAULT\_TOLERANCE\_VAL) As Boolean

If (T1\_VAL - T2\_VAL) <= PUB\_MIN\_REAL\_VAL Then

LTEQUALT\_FUNC = True

Exit Function

End If

If DIFF\_RATIO\_FUNC(T1\_VAL, T2\_VAL) < tolerance Then

LTEQUALT\_FUNC = True

Else

LTEQUALT\_FUNC = False

End If

End Function

' Test if T1\_VAL is greater than or equal to T2\_VAL within tolerance

Function GTEQUALT\_FUNC(T1\_VAL As Variant, \_

T2\_VAL As Variant, \_

Optional tolerance As Double = PUB\_DEFAULT\_TOLERANCE\_VAL) As Boolean

If (T1\_VAL - T2\_VAL) >= -PUB\_MIN\_REAL\_VAL Then

GTEQUALT\_FUNC = True

Exit Function

End If

If DIFF\_RATIO\_FUNC(T1\_VAL, T2\_VAL) < tolerance Then

GTEQUALT\_FUNC = True

Else

GTEQUALT\_FUNC = False

End If

End Function

' Test if T1\_VAL is at least tolerance less than T2\_VAL

Function LTT\_FUNC(T1\_VAL As Variant, \_

T2\_VAL As Variant, \_

Optional tolerance As Double = PUB\_DEFAULT\_TOLERANCE\_VAL) As Boolean

If (T1\_VAL - T2\_VAL) < PUB\_MIN\_REAL\_VAL Then

If DIFF\_RATIO\_FUNC(T1\_VAL, T2\_VAL) < tolerance Then

LTT\_FUNC = False

Else

LTT\_FUNC = True

End If

Else

LTT\_FUNC = False

End If

End Function

' Test if T1\_VAL is at least tolerance greater than T2\_VAL

Function GTT\_FUNC(T1\_VAL As Variant, \_

T2\_VAL As Variant, \_

Optional tolerance As Double = PUB\_DEFAULT\_TOLERANCE\_VAL) As Boolean

If (T1\_VAL - T2\_VAL) > -PUB\_MIN\_REAL\_VAL Then

If DIFF\_RATIO\_FUNC(T1\_VAL, T2\_VAL) < tolerance Then

GTT\_FUNC = False

Else

GTT\_FUNC = True

End If

Else

GTT\_FUNC = False

End If

End Function