A good TREND function in Excel (VBA)

Summary

1 Purpose of the analysis 1

2 Example using the function TendanceSpline 1

2.1 By default Spline 1
2.2 Example using the function TendanceSpline with other interpolation/extrapolation functions 2

3 Conclusion 2

4 Listing of the VBA function 2

4.1 Public Static Function TendanceSpline(YY_known As Range, XXknown As Range, Xnew As Variant, Optional Y_Spl0Lin1Exp2Log3Hyp4 = 0, Optional XLin1Exp2Log3Hyp4 = 1) 2
4.2 Static Function UsrFctX(X, Optional NormalInverse As Variant, Optional Typ As Variant) *'From UtileNewPerso05* 5
4.3 Static Function UsrFctY(Y, Optional NormalInverse As Variant, Optional Typ As Variant) *'From UtileNewPerso05* 6
4.4 Function Hunt(X() As Double, nbRows As Long, var As Double, lo As Long) ' lo=Hunt(X,nbRows,var,lo) ... VBA implementation of that found on pp 112 Numerical Recipes in Fortran 77, 2001 ISBN 0 521 43064 X 6
4.5 Sub SortKey(Tabl() As Double, subKey() As Double, LB As Long, UB As Long, Optional MaxRecur = 10000) 7

5 Reference Spline 10

1 Purpose of the analysis

The original TREND function (also called Tendance) is very useful when it is used once for a set of data having a linear evolution. It founds the Y value corresponding to the given X value.

Generally, when the evolution is not linear, the TREND shall use a limited set of data near or bracketing the given X value. If this has to be done several times, one can manually change the set of data according to each new given X values. When this must be done for many cells, this approach is very time consuming.

Hence a function has been built, very similar to TREND, with the same arguments: this is TendanceSpline function. It accepts a full nonlinear set of data, and found the best data near or bracketing each given X values.

Function **TendanceSpline**(YY_known, XXknown, Xnew, Y_Spl0Lin1Exp2Log3Hyp4 = 0, XLin1Exp2Log3Hyp4 = 1)

2 Example using the function TendanceSpline

2.1 By default Spline

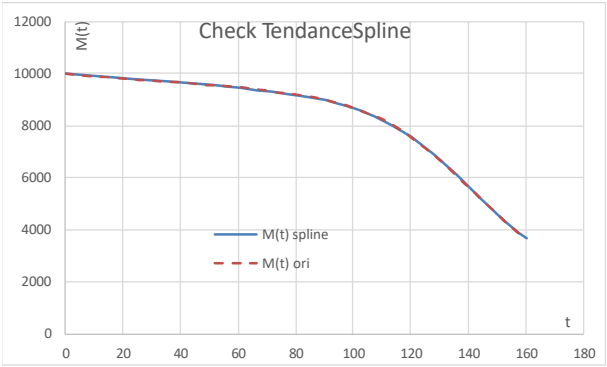
	A	B	C	D	E	F	G	H
20	t	M(t) ori		t	goIsp(t)		t new	M(t) spline
21	0	10000		0.089	9498.5		0	10000
22	8.909	9911.3		6.147	10160.2		0.089	9999.1318
23	17.105	9827.8		11.581	10711.6		6.147	9939.4462
24	25.479	9759.8		17.55	11194.1		11.581	9883.346
25	34.031	9697		22.272	11456		17.55	9823.7031
26	41.425	9639.5		28.864	11614.6		22.272	9784.1103
27	48.998	9571.6		35.635	11518.1		28.864	9734.8681
28	57.55	9488.1		41.069	11304.4		35.635	9685.0889
29	66.102	9383.8		46.592	10994.2		41.069	9642.4825
30	74.298	9279.6		52.294	10608.2		46.592	9593.6543
	…	…					…	…
47	153.318	4249.8		144.053	3185		136.125	6067.5764
48	157.862	3834.6		152.071	3288.3		144.053	5213.4247
49	160	3653		157.506	3440		152.071	4377.5498
50				160	3508.9		157.506	3865.2266
51							160	3653

=**TendanceSpline**($B$21:$B$49; $A$21:$A$49; G21)
and so on, copy down…

Note:
The range Y_known is absolute $B$21:$B$49
The range Xknown is absolute $A$21:$A$49
The cell Xnew is relative address G21
So that, the cells of the sheet Y_known and Xknown are read only one time, even if this function is called 50 times. This represents a lot of time saving.

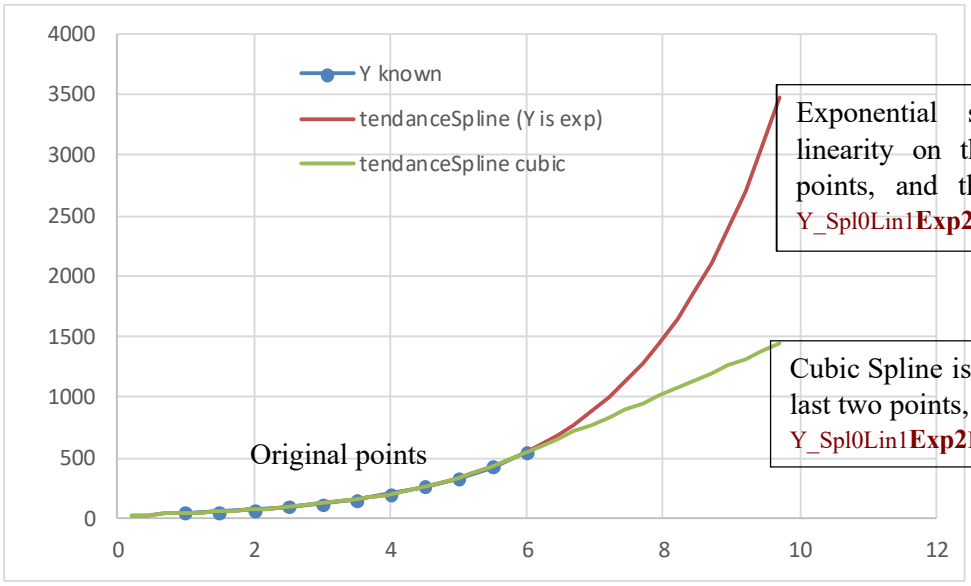
By default, **TendanceSpline** use the spline functions for finding the best Y corresponding to each given X

But if the shape of the function is exponential, better to use interpolation and extrapolation for exponential curves with the optional parameter Y_Spl0Lin1**Exp2**Log3Hyp4 = 2



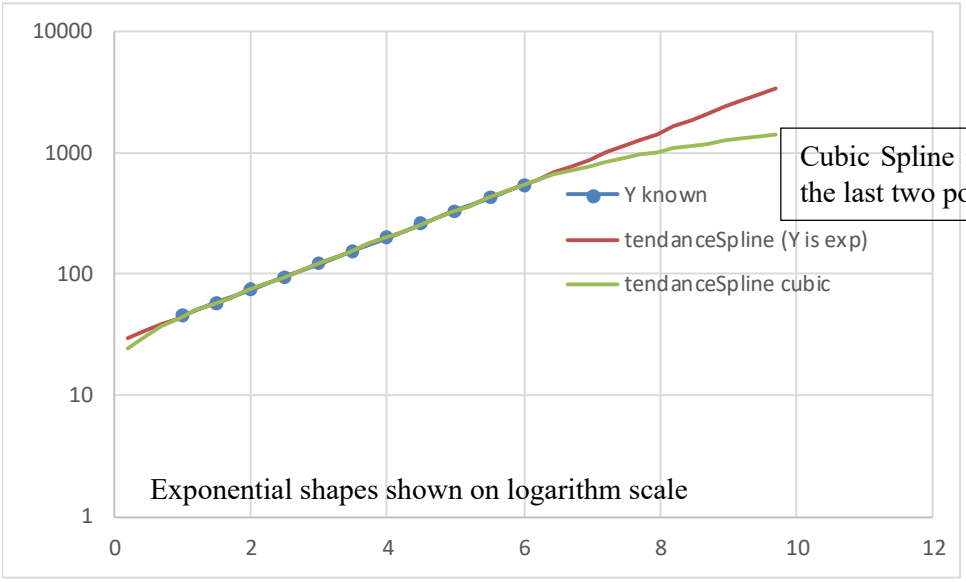
The feature using the optional parameter is shown in the next plots

2.2 Example using the function TendanceSpline with other interpolation/extrapolation functions



Exponential shapes are extrapolated by linearity on the logarithm of the last two points, and then back by an exponential, Y_Spl0Lin1**Exp2**Log3Hyp4 = 2

Cubic Spline is extrapolated by linearity on the last two points,
Y_Spl0Lin1**Exp2**Log3Hyp4 = 0 (default value= Spline)



Cubic Spline is extrapolated by linearity on the last two points, far from the straight line

.

3 Conclusion

TendanceSpline is a good function for simple interpolation from huge set of data, it works fine so far.

4 Listing of the VBA function

' Listings 5.15, June 07, Copyright 1992-2021 Support@KopooS.com, Release 07 mars 2021 20:33:38

4.1 Public Static Function TendanceSpline(YY_known As Range, XXknown As Range, Xnew As Variant, Optional Y_Spl0Lin1Exp2Log3Hyp4 = 0, Optional XLin1Exp2Log3Hyp4 = 1)

```
"Application.MacroOptions Macro:="TendanceSpline", Description:="Better than Tendance fonction from Excel, full vector input, output vector possible, Manage Spline Lin Log and Exp Ln interpolation, default is Spline Y and linear X", Category:=10
'this function is a cubic Spine for interpolation between the two X cells containing Xnew (after a first unique strict increasing sort has been performed on X and mirrored on Y).
'Attention as for Tendance, Y in first position , type of interpolation Y first                    Traceability:From UtileNewPerso05
'          But instead of the two cell around Xnew needed for Tendance, here a full vector Y_known and Xknown as range can be given as input (generally in absolute address)
'          and this function seach in the large vector Xknown the appropriate two X cells around the Xnew  which gives also the  appropriate two Y cells
'Better than Tendance fonction from Excel
'   thanks to the full vector input in Yknown and Xknown, this function seach in the large vector Xknown
'   no needs of sorted values in Xknown full vector input  (However, this Xknown is sorted inside the function as X() only one time)
'   Extrapolate from the last two point of Xknown if the wanted value is outside the given range
'   Manage cubic Spline, Lin Log, Exp and hyperbolic interpolation on Y axis  (and Lin Log , Exp and hyperbolic interpolation on X axis)  with setting respectively 0, 1 2 3 4. By default cubic Spline on Y and  lin on X
'
'   Example for cubic spline     =TendanceSpline($B$21:$B$49; $A$21:$A$49; G21:G23) where Xnew is the range G21:G23
'
'   Tips: if the Y is exponential, its values are about a staight line when seen in Y log scale  then use  Y_Splin0Lin1Exp2Log3Hyp4=2  as indicated by Exp2  and nothing for the second =TendanceSpline(Y_known ; Xknown;  Xnew; 2)
'
'   probably faster thanTendance fonction from Excel because sub static, and read and sort only once the data from Excel if the Y_range and Xrange are unchanged
'   if the user has several TendanceSpline with different large Y_range and Xrange, better 1) to set Xnew as vertical range, and select a free output range (not a single cell) , type the formula and while still in formula do a control-shift-enter
```

```
' or 2) to make and call several similar TendanceSplineA...B...C etc or 3) better see Public Static Function trouveZ_XY using Const TBMAX = 4 that save all called matrixes in a static sub however <=TBMAX
On Error Resume Next
    Dim Y() As Double, X() As Double, Y_knownMemoAddress As String, XknownMemoAddress As String, Y_SplineMemo As Integer, YpXsumMemo As Double, YpXsum As Double
    Dim Y_known As Variant, Xknown As Variant, Xmat As Variant, var As Double
    Dim ret() As Variant
    Dim i As Long
'*********************************************************************
'Xnew must be a scalar or a sigle column ( nXnew row 1 column) or single row ( 1 row nXnew column), to be mapped for Spline always into a R C matrix  nXnew row 1 column  mat(nXnew,1)
'Case of only one cell for Xnew convert to Xmat(1, 1) = Xnew  Else if Xnew already a range (i.e. a matrix)  Xmat = Xnew
'but if Xnew is a transpose 1 row nXnew column, it is to be mapped into a transposed matrix  nXnew row 1 column  Xmat(nXnew,1)
    nXnew& = 0:  ncol& = 0:  On Error GoTo fauteUbound 'needed for managing the single scalar into matrix
    Xmat = Xnew:     nXnew = UBound(Xmat, 1): ncol = UBound(Xmat, 2) 'produces error for a scalar, so out 0
    On Error GoTo faute

    If ncol = 0 Then 'Case one cell  'only one cell is not a matrix
       If nXnew = 0 Then ReDim Xmat(1 To 1, 1 To 1): Xmat(1, 1) = Xnew: nXnew = 1 Else Stop ': Xmat = Application.Transpose(Xmat): nXnew = ncol 'hum we have ncol = 0  and n>0  TBCheck nXnew,0
    ElseIf ncol > 1 Then 'case single row==> transpose the ncol in nXnew rows
       If nXnew = 1 Then Xmat = Application.Transpose(Xmat): nXnew = ncol Else MsgBox "The Xnew argument of TendanceSplin must be a scalar or a single vector, not a matrix.", vbExclamation, "Spline Evaluation": Stop 'nXnew = 0 for ERROR report
    End If
    ReDim ret(1 To nXnew, 0 To 3)
'*********************************************************************
'check full initialisation needed: change the known inputs, or change lin in Y (memo=1) to cubic Spline in Y (=0)
    nbRows& = YY_known.Count
    YpXsum = Application.Sum(YY_known, XXknown) 'check the total sum of X and Y
    If Y_knownMemoAddress = YY_known.Address And XknownMemoAddress = XXknown.Address And YpXsumMemo = YpXsum And Not (Y_SplineMemo <> 0 And Y_Spl0Lin1Exp2Log3Hyp4 = 0) Then GoTo Skipinit
    'Full initialisation: count and read the Excel sheet, and save in memo the range addresses
fullInit:
        Y_known = YY_known: Xknown = XXknown
        nbY& = UBound(Y_known, 1)
        nbX& = UBound(Xknown, 1)
        nbRows& = nbX&: If nbY < nbX Then nbRows = nbY
        If errWrong = 0 And nbRows < 2 Then errWrong = errWrong + 1: MsgBox "Wrong input TendanceSpline; Y_known and Xknown size  >=2", vbCritical, "Error": TendanceSpline = CVErr(xlErrValue): Exit Function
        If errWrong = 0 And Not nbX = nbY Then errWrong = errWrong + 1: MsgBox "Wrong input TendanceSpline; Y_known and Xknown shall have the same size ", vbInformation, "The minimum is taken"
        ReDim Y(1 To nbY), X(1 To nbX)
        'read
        For i = 1 To nbY: Y(i) = Y_known(i, 1): Next
        For i = 1 To nbRows: X(i) = Xknown(i, 1): Next
        'Sort the X vector by increasing values
        SortKey X(), Y(), 1, nbRows
        If Y_Spl0Lin1Exp2Log3Hyp4 = 0 Then 'Case Spline
'********************************************************************* "VBA implementation of that found on pp 109 Numerical Recipes in Fortran 77, 2001 ISBN 0 521 43064 X (pp 115-116, ' "Numerical Recipes in C", 2nd ed., by Press et al., Cambridge University Press ' 1999/6/25 by David J. Braden)
            Dim EndPtDeriv As Boolean, LowFirstDeriv As Double, HighFirstDeriv As Double, dH As Double, dA As Double, dB As Double, dY2() As Double, jj As Long, lo As Long, fEndpoint As Boolean
            Const EPS = 0.000000000001 'used to check for kinkiness of f" when deriving f"
            UseNatBound = 1E+30: EndPtDeriv = False: LowFirstDeriv = UseNatBound: HighFirstDeriv = UseNatBound
            ReDim dY2(1 To nbRows) As Double    'these are the 2nd derivatives
            Dim n As Long, k As Long, p As Double, qn As Double, sig As Double, un As Double, u() As Double
            n = nbRows
            ReDim u(1 To n - 1) As Double  ' N-1 only
```

```
                If LowFirstDeriv >= UseNatBound Then dY2(1) = 0#: u(1) = 0# Else dY2(1) = -0.5:  u(1) = (3# / (X(2) - X(1))) * ((Y(2) - Y(1)) / (X(2) - X(1)) - LowFirstDeriv)
                For i = 2 To n - 1
                   sig = (X(i) - X(i - 1)) / (X(i + 1) - X(i - 1)) ' 'sig = dx Left/ dx full
                   p = sig * dY2(i - 1) + 2#
                   dY2(i) = (sig - 1#) / p
                   u(i) = (6# * ((Y(i + 1) - Y(i)) / (X(i + 1) - X(i)) - (Y(i) - Y(i - 1)) / (X(i) - X(i - 1))) / (X(i + 1) - X(i - 1)) - sig * u(i - 1)) / p
                Next
                If HighFirstDeriv >= UseNatBound Then qn = 0#: un = 0# Else qn = 0.5: un = (3# / (X(n) - X(n - 1))) * (HighFirstDeriv - (Y(n) - Y(n - 1)) / (X(n) - X(n - 1)))
                dY2(n) = (un - qn * u(n - 1)) / (qn * dY2(n - 1) + 1#) 'with dY2(N - 1) = (sig - 1#) / p
                For k = n - 1 To 1 Step -1: dY2(k) = dY2(k) * dY2(k + 1) + u(k): Next 'with dY2(k) = (sig - 1#) / p
            '*****************************************************************
            End If
    'Save New addresses for reducing later excel read and sort processes
            Y_knownMemoAddress = YY_known.Address
            XknownMemoAddress = XXknown.Address
            YpXsumMemo = YpXsum
            Y_SplineMemo = Y_Spl0Lin1Exp2Log3Hyp4
Skipinit: 'CellAround:
   'settings statics because Optional parameters are not set: we shall first perform a call in order to set it Y_Spl0Lin1Exp2Log3Hyp4 and XLin1Exp2Log3Hyp4
   a = UsrFctY(1, 1, Y_Spl0Lin1Exp2Log3Hyp4):    a = UsrFctX(1, 1, XLin1Exp2Log3Hyp4)          'after one simply use UsrFctY() without any Optional parameters. Here after Skipinit because user can change the flags with no need to run fullInit
   errWrong = 0
  'CellAround large bracket Interp Extrapol
   For jj = 1 To nXnew ' the range of new values Xnew() needing interpolations
      Imin = 0
      var = Xmat(jj, 1)
      If Not Application.IsNumber(var) Then
         For j = 0 To 3: ret(jj, j) = CVErr(xlErrValue): Next
      ElseIf var < X(1) Or var > X(nbRows) Then
         'extrapolation KopooS
         For j = 1 To 3: ret(jj, j) = CVErr(xlErrNum): Next
         If var < X(1) Then Imin = 1: GoSub Evaluate
         If var > X(nbRows) Then Imin = nbRows - 1: GoSub Evaluate
      Else

         lo = Hunt(X, nbRows, var, lo) ': GoTo fast  'find the best brackets in the strick monotonic X()

         Xmin = X(lo): Imin = lo 'at upper bound of table's domain, means that X(nbRows) = var
         If Not Y_Spl0Lin1Exp2Log3Hyp4 = 0 Then 'and absolutely in the case of pure interpolation
               If Imin = 0 Then   'bad init==> need a full init again
                  DoEvents
                  GoTo fullInit
               End If
    'now all is given Exple: linUsrFctX: Ynew = ymin + (ymax - ymin) / (Logd(xmax) - Logd(xmin)) * (Logd(Xnew) - Logd(xmin))
               GoSub Evaluate 'jj,ret, Xmin,Y(),var,UsrFctY,UsrFctX
         ElseIf Y_Spl0Lin1Exp2Log3Hyp4 = 0 Then 'Spline cubic
               Imax = Imin + 1
               dH = X(Imax) - X(Imin): dH2 = dH * dH
               dA = (X(Imax) - var) / dH: dA2 = dA * dA: dA3 = dA2 * dA
               dB = 1 - dA: dB2 = dB * dB: dB3 = dB2 * dB
```

```
            ' calculate Y(s)
            If fEndpoint Then
                ret(jj, 0) = Y(nbRows)
            Else
                ret(jj, 0) = dA * Y(Imin) + dB * Y(Imax) + ((dA3 - dA) * dY2(Imin) + (dB3 - dB) * dY2(Imax)) *
dH2 / 6#
            End If
            For j = 1 To 3: ret(jj, j) = CVErr(xlErrNA): Next
            'GoSub furtherEval
        End If
    End If
  Next 'jj
  TendanceSpline = ret
Exit Function
Evaluate: 'jj,ret, Xmin,var,Y(),UsrFctY,UsrFctX
  Imax = Imin + 1: Xmax = X(Imax):  Xmin = X(Imin): Ymin = Y(Imin):     Ymax = Y(Imax)
  ret(jj,  0)  =  UsrFctY(Y:=(UsrFctY(Ymin)  +  (UsrFctY(Ymax)  -  UsrFctY(Ymin))  /  (UsrFctX(Xmax)  -
UsrFctX(Xmin)) * (UsrFctX(var) - UsrFctX(Xmin))), NormalInverse:=-1)
  For j = 1 To 3: ret(jj, j) = CVErr(xlErrValue): Next 'non significant
  Return
furtherEval: 'only in case of cubic Spline  1st derivative,2nd derivative,3rd derivative 'fEndpoint,EndPtDeriv,
jj,ret(),Imin, Imax, X(), Y(), dY2(), dH, dA,dA2,dB,dB2
  'further evaluations
  fEndpoint = fEndpoint Or (lo = 1 And dB = 0)
  If fEndpoint And Not (EndPtDeriv) Then
        For j = 1 To 3: ret(jj, j) = CVErr(xlErrNA): Next
  Else 'have interior point, or will accept a one-sided derivative
        ret(jj,  1)  =  (Y(Imax)  -  Y(Imin))  /  dH  +  dH  /  6#  *  ((3#  *  dB2  -  1)  *  dY2(Imax)  -  (3#  *  dA2  -  1#)  *
dY2(Imin)) '1st derivative
        ret(jj, 2) = dA * dY2(Imin) + dB * dY2(Imax) '2nd derivative
        If fEndpoint Then '3rd derivative
            ret(jj, 3) = CVErr(xlErrNA)
        Else
            dAt = (dY2(Imax) - dY2(Imin)) / dH 'dA used as temp here
            If var > X(lo) Then ' if in interior, or on a smooth boundary point, all ok
              ret(jj, 3) = dAt
            ElseIf Abs(dAt - (dY2(Imin) - dY2(Imin - 1)) / dH) < EPS Then
              ret(jj, 3) = dAt
            Else
              ret(jj, 3) = CVErr(xlErrNA)
            End If
        End If
  End If
  Return
fauteUbound: 'normal error using Ubound with a single cell or scalar. The output is managed
  Resume Next
faute:
  'Stop
  Resume Next
End Function ' TendanceSpline()
```

4.2 Static Function UsrFctX(X, Optional NormalInverse As Variant, Optional Typ As Variant) 'From UtileNewPerso05

```
"Application.MacroOptions Macro:="UsrFctX", Description:=" dedicated  to macro TendanceSpline X", Category:=10
'Intentionally (and exceptionally) the Optional parameters are not set: this is to simplify the use once the static parameters have been set once
'BUT we shall first perform a call in order to set the static values of this function
```

```
'NormalInverse >0 for normal function  <=0 for inverse function
On Error Resume Next 'mainly case error when no optional are given
   call_number = call_number + 1
   If Not IsMissing(NormalInverse) Then Inversee = NormalInverse '
   If Not IsMissing(Typ) Then Typee = Typ
   If Inversee = 0 Then
        If call_number < 2 Then MsgBox "Alert : initialisation missing in sub function UsrFctX ", vbCritical,
"ERROR UsrFctX" 'displayed only once
        UsrFctX = 0: Exit Function
   End If
   Select Case Typee
        Case Is = 0, 1 'linear   If there is more than one value in a single list, the values are separated by commas
           If Inversee > 0 Then UsrFctX = X Else UsrFctX = X
        Case Is = 2 'log decimal  10 power for exponential curve
           If Inversee > 0 Then UsrFctX = Log(X) / Log(10#) Else UsrFctX = 10 ^ X 'log decimal
        Case Is = 3 'exp Log Neperien for log curve
           If Inversee > 0 Then UsrFctX = Exp(X) Else UsrFctX = Log(X)  'exponentiel
        Case Is = 4  'ln Hyperbolic
           If Inversee > 0 Then UsrFctX = 1 / (X) Else UsrFctX = 1 / (X) ' Hyperbolic
   End Select
End Function ' UsrFctX()
```

4.3 Static Function UsrFctY(Y, Optional NormalInverse As Variant, Optional Typ As Variant) 'From UtileNewPerso05

```
"Application.MacroOptions Macro:="UsrFctY", Description:=" idem UsrFctX but for Y", Category:=10
On Error Resume Next 'mainly case error when no optional are given
   call_number = call_number + 1
   If Not IsMissing(NormalInverse) Then Inversee = NormalInverse '
   If Not IsMissing(Typ) Then Typee = Typ
   If Inversee = 0 Then
        If call_number < 2 Then MsgBox "Alert : initialisation missing in sub function UsrFctX ", vbCritical,
"ERROR UsrFctX" 'displayed only once
        UsrFctY = 0: Exit Function
   End If
   Select Case Typee
        Case Is = 0, 1  'linear
           If Inversee > 0 Then UsrFctY = Y Else UsrFctY = Y
        Case Is = 2 'log decimal   10 power  for exponential curve
           If Inversee > 0 Then UsrFctY = Log(Y) / Log(10#) Else UsrFctY = 10 ^ Y 'log decimal
        Case Is = 3 'exp Log Neperien  for log curve
           If Inversee > 0 Then UsrFctY = Exp(Y) Else UsrFctY = Log(Y)  'exponentiel
         Case Is = 4 'Hyperbolic
           If Inversee > 0 Then UsrFctY = 1 / (Y) Else UsrFctY = 1 / (Y) ' Hyperbolic
   End Select
End Function ' UsrFctY()
```

4.4 Function Hunt(X() As Double, nbRows As Long, var As Double, lo As Long) ' lo=Hunt(X,nbRows,var,lo) ... VBA implementation of that found on pp 112 Numerical Recipes in Fortran 77, 2001 ISBN 0 521 43064 X

```
"Application.MacroOptions Macro:="Hunt", Description:="Similar to Excel Match (RechercheV), output the first index lo such X(lo) <=var<= X(lo+1)", Category:=10
' Hunting better brackets (that is searching better brackets from previous position lo as initial guess) then use bisection for finding the smallest bracket
  'Given an array X(1:nbRows), and given a value var, returns a value lo such that var is between X(lo) and X(lo+1). X(1:nbRows) must be monotonic, either increasing or decreasing.
  'lo=0 or lo=nbRows is returned to indicate that var is out of range. lo is on input is taken as the initial guess; and lo is on output for the solution lo too.
   Dim inc As Long, high As Long, jm As Long, ascnd As Boolean, flag As Boolean
   On Error GoTo faute
'search better brackets, that is Hunting better brackets
   ascnd = (X(nbRows) >= X(1)): If (lo <= 0 Or lo > nbRows) Then lo = 0: high = nbRows + 1: GoTo bissec3 'Input guess not
useful. Go immediately to bisection.
   inc = 1 'Set the hunting increment.
```

```
    If (var >= X(lo)) = ascnd Then 'Hunt up:
        Do
            high = lo + inc: If high > nbRows Then high = nbRows + 1: Exit Do 'Done hunting brackets, since off end of table.
            flag = (var >= X(high)) = ascnd: If flag Then lo = high: inc = inc + inc 'double the increment and try again.
        Loop Until Not flag
    Else ': Hunt down:
        high = lo
        Do
            lo = high - inc: If lo < 1 Then lo = 0: Exit Do 'Done hunting brackets, since off end of table.
            flag = (var < X(lo)) = ascnd: If flag Then high = lo: inc = inc + inc 'so double the increment and try again.
        Loop Until Not flag 'Done hunting, value bracketed.
    End If 'Hunt is done, so begin the final bisection phase:
bissec3: 'final bisection phase for finding the smallest bracket
    Do
       If (high - lo = 1) Then ' smallest bracket found
          If var = X(nbRows) Then lo = nbRows - 1
          If var = X(1) Then lo = 1
          Hunt = lo: Exit Do 'Return 'done lo has been found
       End If
       jm = (high + lo) / 2: If (var >= X(jm)) = ascnd Then lo = jm Else high = jm
    Loop
    Exit Function
faute:
    Stop
    Resume Next
End Function ' Hunt()
```

4.5 Sub SortKey(Tabl() As Double, subKey() As Double, LB As Long, UB As Long, Optional MaxRecur = 10000)

```
'Tabl : a numeric list to be sorted, only from LowerBound to UpperBound
' subKey : a sub key which will reflect the sort done on Tabl only ... normally just a parmetric Index increasing from LB to UB
'        which is used to sort other tables linked to Tabl  by sub key : table1(subKey(i)) corresponds to the new sorted Tbl(i)
'
'Dapres une idée publiée par Auteur : Jhon Fullspeed Version : 09/08/2004 Tri Quick-sort sans appel récursif
Dim RangeR()  As Long
ReDim RangeR(0 To MaxRecur, 0 To 1)      " Recursivity Table  Table qui remplace la recursivité et améliore CONSIDERABLEMENT les performances
'10000  (abusif Jhon Fullspeed: le dimensionnement à 100 correspond au nombre maximal d'appel en recursivité normalement 10 devraient suffire mais avec 100 on doit pouvoir trier quelque milliards d'infos...)
Dim Permut, PermutSK
Dim memoLevel As Long, memoRecursiv As Long, memoSubset As Long, memoPermutation As Long
    On Error GoTo faute
    If UB <= LB Then Exit Sub 'tri canonique à zero ou 1 élément
    'first check if already strictly increasing
    For i = LB + 1 To UB
       If Tabl(i) <= Tabl(i - 1) Then Exit For
    Next
    If Not i <= UB Then Exit Sub ' ===> done
'case a sort must be performed
    subSetN = 1 ' level of sub sets = N
    RangeR(subSetN, 0) = LB                " Auteur : Jhon Fullspeed Version : 09/08/2004 Tri Quick-sort sans appel récursif IMPORTANT je ne trie pas l'élément ZERO Il suffit normalement de metrre IndexMini à zero pour le trier
    RangeR(subSetN, 1) = UB            " Nombre d'elements à trier
    memoLevel = 4:    memoRecursiv = 0:    memoSubset = 0:    memoPermutation = 0
    Do 'recursivité simulée: la fonction tri permet de découper l'ensemble à trier en 2 parties: partie plus petite que la valeur du pivot et partie plus grande
       'ensuite on s'ocupe de la partie plus petite, et on sauvegarde les index de la partie plus grande en fonction de subSetN
       'ensuite on continue avec un éventuel nouveau découpage, qui ici se réalise en incrémentant le subSetN et sauvegarde les index
```

```
'et ainsi de suite.. jusqu'a ce que l'ensemble se réduise à 1 ou zéro élément.
'puis on prend en compte les index sauvegardés dans le dernier subSetN
  IndexINF = RangeR(subSetN, 0)                " Utilisation unique des index qui ont été Sauvegardés au niveau le plus grand actuel
  IndexSUP = RangeR(subSetN, 1)
  subSetN = subSetN - 1                        "prépararation du niveau en dessous  ou pour ré-écrire de nouveaux index au niveau actuel
  memoRecursiv = memoRecursiv + 1
  Do ' size subset
    jUp = IndexINF                 " le quick sort trie le fichier par morceau de plus en plus petits
    jDown = IndexSUP
    IndexPivot = Int((IndexINF + IndexSUP) / 2)        " on divise le nombre d'élements choisi par deux
    ValeurPivot = Tabl(IndexPivot)          " On mémorise la valeur pivot pour le critère de tri de l'élement actif
    memoSubset = memoSubset + 1
    Do 'pour permutation function of Critere
      Do While (Tabl(jUp) < ValeurPivot)   " Partant du debut de la portion selectionnée on, recherche le
        jUp = jUp + 1                  " premier element plus grand ou égal à la ValeurPivot dans cet intervalle
      Loop
      Do While (Tabl(jDown) > ValeurPivot)  " maintenant on en recherche un plus petit ou égal partant de la fin
        jDown = jDown - 1
      Loop
      'nota: l'index croissant ou décroissant peut prendre la valeur IndexPivot du pivot...
      memoPermutation = memoPermutation + 1
      If jUp <= jDown Then " quand on a trouvé, on permute avec resa_Chiffre
        "l'autre concept du quick sort est que plus la  permutation est lointaine plus elle est efficace:il vaut mieux permuter le premier
        " avec le dernier que le premier avec le second il a plus de chance d'être à sa place. Je laisse les mathématiciens le démontrer....
        If jUp < jDown Then
          Permut = Tabl(jDown)
          Tabl(jDown) = Tabl(jUp)
          Tabl(jUp) = Permut
          'subKey now
          PermutSK = subKey(jDown)
          subKey(jDown) = subKey(jUp)
          subKey(jUp) = PermutSK
        End If
        'Passage aux éléments suivant pour continuer la comparaison
        jUp = jUp + 1
        jDown = jDown - 1 'On ne verifie pas les bornes car il y a le loop until apres
      "nota: mais puisque IndexPivot était au mileu de la zone, avec les incrémentations on va sortir à coup sur AC.
      End If
    Loop Until jUp > jDown " on rétrécie la portion de recherche jusqu'à son arrivée à 1 ou 0 élement. On a terminé le rangement par rapport au pivot.
    If jUp < IndexSUP Then  " Et c'est la qu'on recursive en se servant de la table RangeR
'"    If Not (IndexSUP = jUp + 1 And Tabl(IndexSUP) = Tabl(jUp)) Then
      subSetN = subSetN + 1 " pour écrire ou ré-écrire de nouveaux index
      RangeR(subSetN, 0) = jUp 'garde en mémoire les index du reste à faire
      RangeR(subSetN, 1) = IndexSUP
```

```
                If subSetN > MaxRecur Then MsgBox "Attention MaxRecur atteint", vbCritical, "ERROR MaxRecur" 'en fait avec MaxRecur=100 on devrait pouvoir traiter 2^100 soit 10^30 lignes... donc a priori pas de pb...
'''             End If
            End If
            IndexSUP = jDown 'on s'occcupe de la partie basse (partie plus petite que la valeur du pivot)
        Loop Until IndexINF >= IndexSUP  'l'ensemble est réduit à 1 ou zéro élément.
    Loop Until subSetN < 1 'il ne reste aucune zone non traitée  et la c'est FINI.
Exit Sub
faute:
    Stop
    Resume Next
End Sub ' Evaluate  'jj,ret, Xmin,Y()
```

5 Reference Spline

3.3 Cubic Spline Interpolation

Numerical Recipes
in Fortran 77
The Art of Scientific Computing
Second Edition

Given a tabulated function $y_i = y(x_i)$, $i = 1...N$, focus attention on one particular interval, between x_j and x_{j+1}. Linear interpolation in that interval gives the interpolation formula

$$y = Ay_j + By_{j+1} \tag{3.3.1}$$

where

$$A \equiv \frac{x_{j+1} - x}{x_{j+1} - x_j} \qquad B \equiv 1 - A = \frac{x - x_j}{x_{j+1} - x_j} \tag{3.3.2}$$

Equations (3.3.1) and (3.3.2) are a special case of the general Lagrange interpolation formula (3.1.1).

Since it is (piecewise) linear, equation (3.3.1) has zero second derivative in the interior of each interval, and an undefined, or infinite, second derivative at the abscissas x_j. The goal of cubic spline interpolation is to get an interpolation formula that is smooth in the first derivative, and continuous in the second derivative, both within an interval and at its boundaries.

Suppose, contrary to fact, that in addition to the tabulated values of y_i, we also have tabulated values for the function's second derivatives, y'', that is, a set of numbers y''_i. Then, within each interval, we can add to the right-hand side of equation (3.3.1) a cubic polynomial whose second derivative varies linearly from a value y''_j on the left to a value y''_{j+1} on the right. Doing so, we will have the desired continuous second derivative. If we also construct the cubic polynomial to have zero *values* at x_j and x_{j+1}, then adding it in will not spoil the agreement with the tabulated functional values y_j and y_{j+1} at the endpoints x_j and x_{j+1}.

A little side calculation shows that there is only one way to arrange this construction, namely replacing (3.3.1) by

$$y = Ay_j + By_{j+1} + Cy''_j + Dy''_{j+1} \tag{3.3.3}$$

where A and B are defined in (3.3.2) and

$$C \equiv \frac{1}{6}(A^3 - A)(x_{j+1} - x_j)^2 \qquad D \equiv \frac{1}{6}(B^3 - B)(x_{j+1} - x_j)^2 \tag{3.3.4}$$

Notice that the dependence on the independent variable x in equations (3.3.3) and (3.3.4) is entirely through the linear x-dependence of A and B, and (through A and B) the cubic x-dependence of C and D.

We can readily check that y'' is in fact the second derivative of the new interpolating polynomial. We take derivatives of equation (3.3.3) with respect to x, using the definitions of A, B, C, D to compute $dA/dx, dB/dx, dC/dx$, and dD/dx. The result is

$$\frac{dy}{dx} = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{3A^2 - 1}{6}(x_{j+1} - x_j)y''_j + \frac{3B^2 - 1}{6}(x_{j+1} - x_j)y''_{j+1} \tag{3.3.5}$$

for the first derivative, and

$$\frac{d^2y}{dx^2} = Ay''_j + By''_{j+1} \tag{3.3.6}$$

for the second derivative. Since $A = 1$ at x_j, $A = 0$ at x_{j+1}, while B is just the other way around, (3.3.6) shows that y'' is just the tabulated second derivative, and also that the second derivative will be continuous across (e.g.) the boundary between the two intervals (x_{j-1}, x_j) and (x_j, x_{j+1}).

The only problem now is that we supposed the y''_i's to be known, when, actually, they are not. However, we have not yet required that the *first* derivative, computed from equation (3.3.5), be continuous across the boundary between two intervals. The key idea of a cubic spline is to require this continuity and to use it to get equations for the second derivatives y''_i.

The required equations are obtained by setting equation (3.3.5) evaluated for $x = x_j$ in the interval (x_{j-1}, x_j) equal to the same equation evaluated for $x = x_j$ but in the interval (x_j, x_{j+1}). With some rearrangement, this gives (for $j = 2, \ldots, N-1$)

$$\frac{x_j - x_{j-1}}{6}y''_{j-1} + \frac{x_{j+1} - x_{j-1}}{3}y''_j + \frac{x_{j+1} - x_j}{6}y''_{j+1} = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{y_j - y_{j-1}}{x_j - x_{j-1}} \tag{3.3.7}$$

These are $N - 2$ linear equations in the N unknowns $y''_i, i = 1, \ldots, N$. Therefore there is a two-parameter family of possible solutions.

For a unique solution, we need to specify two further conditions, typically taken as boundary conditions at x_1 and x_N. The most common ways of doing this are either

- set one or both of y''_1 and y''_N equal to zero, giving the so-called *natural cubic spline*, which has zero second derivative on one or both of its boundaries, or
- set either of y''_1 and y''_N to values calculated from equation (3.3.5) so as to make the first derivative of the interpolating function have a specified value on either or both boundaries.

One reason that cubic splines are especially practical is that the set of equations (3.3.7), along with the two additional boundary conditions, are not only linear, but also *tridiagonal*. Each y''_j is coupled only to its nearest neighbors at $j \pm 1$. Therefore, the equations can be solved in $O(N)$ operations by the tridiagonal algorithm (§2.4). That algorithm is concise enough to build right into the spline calculational routine. This makes the routine not completely transparent as an implementation of (3.3.7), so we encourage you to study it carefully, comparing with `tridag` (§2.4).
