

=====

=====

'INTRODUCTION

'• De nombreux laboratoires de recherche en physique utilisent le logiciel de minimisation MINUIT pour ajuster des modèles théoriques sur des données expérimentales.

'Aux environs de 1970, au fur et à mesure de ses développements successifs, le logiciel MINUIT s'était toutefois déjà progressivement transformé en "usine à gaz" : 'un logiciel hyper complet, mais d'utilisation très lourde pour effectuer des actions basiques. La situation a encore abominablement empiré depuis '(l'usine à gaz est devenue carrément "intergalactique"... les fous d'informatique peuvent le vérifier facilement avec une petite recherche sur internet : 'rien que pour en comprendre les rouages élémentaires, il faut se plonger dedans pendant trois jours).

'Dès le début des complications, Berthon et Portes avaient choisi de simplifier la tâche des utilisateurs "ordinaires" en en programmant (en fortran) une version très basique : MINCON ; 'beaucoup plus simple à mettre en oeuvre, mais possédant de nombreuses fonctionnalités dont une prise en compte efficace des calculs d'incertitudes.

'• Dans les années 1980, faute de logiciels équivalents sur Macintosh, j'en avais reprogrammé une version en pascal ; encore plus simplifiée, 'mais j'y avais joint une interface rudimentaire avec un interpréteur de formules et un traceur de graphiques (MINGRAPH ; dont le code est sur mon site).

'L'évolution des ordinateurs et de leurs systèmes d'exploitation nécessite toutefois une mise à niveau incessante des logiciels. Je n'ai hélas que très peu de temps pour assurer cela.

'Or, bien que plusieurs logiciels récents disposent de fonctionnalités semblables, aucun (ni même Maple ou Mathematica, semble-t-il) ne gère efficacement les calculs d'incertitudes et des corrélations.

'L'enseignement de ces dernières s'était d'ailleurs un peu perdu, à tel point qu'il semble que quelques enseignants eux mêmes n'en maîtrisent plus très bien certains aspects.

'• Je tente ici de mettre au point une version Excel de la procédure de minimisation MINIMI.

'J M Laffaille - septembre 2016

=====

=====

Option Explicit 'impose que tous les types soient déclarés (pour vérifier)

'variables globales utilisées par la fenêtre d'appel de Minimi

Public MG_adrFCN As String 'on ne calcule pas FCN, on utilise la cellule où le tableur calcule

Public MG_adrPar(1 To 30) As String 'pour pouvoir agir sur les paramètres dont

```

dépend FCN
Public MG_nPar As Integer
Public MG_Par(1 To 30) As Double 'pour stocker les valeurs initiales et les restaurer
à la fin si nécessaire
Public MG_adrDPar(1 To 30) As String
Public MG_DPar(1 To 30) As Double
Public MG_adrParN(1 To 30) As String
Public MG_ParN(1 To 30) As String
Public MG_step As Double
Public MG_epsi As Double
Public MG_impr As Integer
Public MG_err As Boolean
Public MG_minimisation As Boolean 'vba refuse qu'on ajoute ici une valeur
d'initialisation ! ? mais, par chance, il initialise par défaut la valeur qu'on désire :
False

```

```

Public Function SorInt(NSt As Integer, Npar As Integer, xp() As Double, xpN() As
String, AMin As Double) As String
    'liste des valeurs intermédiaires des paramètres lors de la minimisation
    Dim ii As Integer
    Dim Tlist As String
    Tlist = vbCrLf & vbCrLf & "Paramètres pour le pas numéro : " & NSt & " (Min = " &
CStr(AMin) & " )"
    For ii = 1 To Npar
        Tlist = Tlist & vbCrLf & xpN(ii) & " : " & CStr(xp(ii))
    Next ii
    SorInt = Tlist
End Function

```

```

Public Sub descente(Npar As Integer, xp() As Double, jg As Integer, yy() As Double,
DirIn() As Double, dd() As Double, xs() As Double, AMin As Double, NFCN As
Integer, NPFN As Integer, Y1 As Double, Y2 As Double, aa As Double)
    'suite de pas pour rechercher un minimum dans une direction
    Const al = 3# 'al et be sont utilisées par plusieurs procédures
    Const be = -0.4
    Const NStepQ = 5 'NStepQ est utilisée par plusieurs procédures
    Dim NS1 As Boolean
    Dim NS2 As Boolean
    Dim NS3 As Boolean
    Dim ii As Integer
    Dim NStepD As Integer
    Y1 = 1#
    NS1 = False
    NS2 = False
    NS3 = False
    NStepD = 0
    dd(jg) = 0#
    While (Not NS2)
        For ii = 1 To Npar
            yy(ii) = xp(ii) + DirIn(jg) * xs(jg, ii)

```

```

Next ii
For ii = 1 To Npar
    Range(MG_adrPar(ii)).Value = yy(ii) 'on ne calcule pas FCN, c'est fait par le
tableur : il faut modifier les paramètres
Next ii
Calculate 'puis il faut forcer (au cas où...) le recalcul
aa = Range(MG_adrFCN).Value 'on peut enfin récupérer FCN
NFCN = NFCN + 1
Y2 = AMin - aa
If Y2 < 0 Then
    Dirln(jg) = be * Dirln(jg)
    NStepD = NStepD + 1
    If (NS1 Or NS3) Then
        NS2 = True
    ElseIf (NStepD >= NStepQ) Then
        Y1 = Y2
        NS3 = True
    End If
Elseif ((Y2 = 0) And (Y1 = 0)) Then
    NS2 = True
Else
    AMin = aa
    Y1 = Y2
    NS3 = False
    NStepD = 0
    NPFN = NFCN
    For ii = 1 To Npar
        xp(ii) = yy(ii)
    Next ii
    dd(jg) = dd(jg) + Dirln(jg)
    Dirln(jg) = al * Dirln(jg)
    NS1 = True
End If
Wend
End Sub

```

```

Public Sub parabole(Npar As Integer, xp() As Double, jg As Integer, mieux As
Boolean, yy() As Double, Dirln() As Double, dd() As Double, xs() As Double, AMin As
Double, NFCN As Integer, NPFN As Integer, Y1 As Double, Y2 As Double, aa As
Double)

```

'pour estimer un minimum local par approximation parabolique à partir de trois points

```

Const al = 3# 'al et be sont utilisées par plusieurs procédures

```

```

Const be = -0.4

```

```

Dim sss As Double

```

```

Dim stj As Double

```

```

Dim ii As Integer

```

```

If mieux Then

```

```

    sss = ((al * al) * Y1 + Y2) / (al * Y1 - Y2)

```

```

    stj = 0.5 * Dirln(jg) * sss / (al * be)

```

```

Else
    sss = ((be * be) * Y1 - Y2) / (be * Y1 - Y2)
    stj = 0.5 * Dirln(jg) * sss / (be * be)
End If
If stj <> 0 Then
    For ii = 1 To Npar
        yy(ii) = xp(ii) + stj * xs(jg, ii)
    Next ii
    For ii = 1 To Npar
        Range(MG_adrPar(ii)).Value = yy(ii) 'on ne calcule pas FCN, c'est fait par le
tableur : il faut modifier les paramètres
    Next ii
    Calculate 'puis il faut forcer (au cas où...) le recalcul
    aa = Range(MG_adrFCN).Value 'on peut enfin récupérer FCN
    NFCN = NFCN + 1
    If aa < AMin Then
        AMin = aa
        NPFN = NFCN
        For ii = 1 To Npar
            xp(ii) = yy(ii)
        Next ii
        dd(jg) = dd(jg) + stj
    End If
End If
End Sub

```

```

Public Function errors(Npar As Integer, xp() As Double, ig As Integer, NStepC As
Integer, yy() As Double, invSig2() As Double, pp() As Double, xs() As Double, AMin
As Double, GoToFroid As Boolean) As Boolean

```

```

'pour calculer une estimation des incertitudes sur les paramètres
Const al = 3# 'al et be sont utilisées par plusieurs procédures
Const be = -0.4
Dim GoToChaud As Boolean
Dim ii As Integer
Dim jj As Integer
Dim kk As Integer
Dim ff As Double
Dim ee(1 To 2) As Double
GoToChaud = False 'sert au redémarrage à chaud (le GoTo existe en VB, mais
n'est pas utilisé ici pour simplifier la compatibilité)
For kk = 1 To 2 ' on teste de chaque coté pour estimer ff = AMin + p2sur2sigma2
    For jj = 1 To Npar
        yy(jj) = xp(jj) + pp(ig) * xs(ig, jj) * ((-1) ^ kk) 'on teste avant et après
        ' on change le signe d'après kk plutôt que changer le signe de pp
    Next jj
    For ii = 1 To Npar
        Range(MG_adrPar(ii)).Value = yy(ii) 'on ne calcule pas FCN, c'est fait par le
tableur : il faut modifier les paramètres
    Next ii
    Calculate 'puis il faut forcer (au cas où...) le recalcul

```

```

ff = Range(MG_adrFCN).Value 'on peut enfin récupérer FCN
For ii = 1 To Npar
    Range(MG_adrPar(ii)).Value = xp(ii) 'ici on remet en place les paramètres
(minimum trouvé, les décalages ne servent qu'à tester le voisinage)
Next ii
Calculate 'puis il faut forcer (au cas où...) le recalcul
If ff > AMin Then 'le test semble correct de ce coté
    ee(kk) = ff
ElseIf ff < AMin Then 'on a trouvé mieux (par hasard...) ; on recentre et on
repart à froid
    For ii = 1 To Npar
        xp(ii) = yy(ii)
        Range(MG_adrPar(ii)).Value = xp(ii) 'on remet en place les nouvelles
valeurs
    Next ii
    Calculate 'puis il faut forcer (au cas où...) le recalcul
    AMin = ff 'le tableur a recalculé la nouvelle valeur (inutile de la récupérer, on
la connaît déjà)
    pp(ig) = al * pp(ig) 'on augmente le pas car il était probablement petit
(voisinage du minimum)
    GoToFroid = True 'il faut relancer la machinerie par rapport à ce nouveau
minimum
    ElseIf ff = AMin Then ' c'est plat, ou bien le pas est trop petit (le minimum
parabolique n'est pas précis)...
        NStepC = NStepC + 1
        pp(ig) = pp(ig) * (al ^ ((NStepC + 1#) / 2#)) ' ...on va essayer plus loin
        GoToChaud = True ' il faudra tester à nouveau SVP
    Else ' indéterminé ? serait on allé trop loin ?... (normalement on devrait ne
jamais passer ici)
        pp(ig) = be * pp(ig)
        GoToChaud = True ' il faudra tester à nouveau SVP
    End If
    If (GoToFroid Or GoToChaud) Then
        Exit For 'for kk (s'il faut recentrer ou si c'est du premier coté, inutile de
chercher du second car il faut recommencer)
    End If
Next kk
If Not (GoToFroid Or GoToChaud) Then 'terminaison normale (sinon il faudra
recommencer)
    invSig2(ig) = (ee(1) + ee(2) - 2 * AMin) / (pp(ig) * pp(ig)) 'on estime 1/sigma2 (et
non sigma2) car cette estimation peut être nulle, voire négative
End If
errors = GoToChaud
End Function

```

```

Public Function voisinage(Npar As Integer, xp() As Double, NP As Integer, yy() As
Double, invSig2() As Double, pp() As Double, xs() As Double, AMin As Double) As
Boolean

```

'étude du voisinage pour calculer une estimation des incertitudes sur les paramètres

Dim ig As Integer 'pour transmettre le numéro du paramètre à la sous-procédure

```

Dim NStepC As Integer
Dim GoToFroid As Boolean
Dim GoToChaud As Boolean

GoToFroid = False 'sert au redémarrage à froid
For ig = 1 To NP
  NStepC = 0
  GoToChaud = True 'sert au redémarrage à chaud
  Do
    GoToChaud = errors(Npar, xp(), ig, NStepC, yy(), invSig2(), pp(), xs(), AMin,
GoToFroid)
    If GoToFroid Then
      Exit Do 'inutile de continuer ici en cas de redémarrage à froid
    End If
    Loop Until (Not GoToChaud And invSig2(ig) > 0)
    If Not GoToFroid Then 'on passe en cas de redémarrage à froid
      pp(ig) = 1 / Sqr(invSig2(ig)) 'pp est une estimation de sigma
    End If
  Next ig
  voisinage = GoToFroid
End Function

```

```

Public Function incertitudes(Npar As Integer, xp() As Double, wt() As Double, NP As
Integer, yy() As Double, invSig2() As Double, pp() As Double, xs() As Double, eet()
As Double, AMin As Double, ErrorsOK As Boolean) As String
  'pour calculer une estimation des incertitudes sur les paramètres
  Dim ii As Integer
  Dim jj As Integer
  Dim kk As Integer
  Dim ll As Integer
  Dim mm As Integer
  Dim GoToFroid As Boolean
  Dim NEST As Integer
  Dim NESTMx As Integer
  Dim ppt(1 To 31) As Double
  Dim suffisant As Boolean
  Dim NElist As String
  Dim compar As Double
  Dim Ncompar As Integer
  invSig2(1) = invSig2(NP + 1) 'calcul d'incertitudes sans dérivées (on fait ce qu'on
peut)
  For ii = 1 To NP
    If invSig2(ii) > 1e-08 Then 'on prend une marge de sécurité par rapport à 0
pour ne pas surstimer pp (en évitant d'être trop restrictif)
      pp(ii) = 1# / Sqr(invSig2(ii)) 'pp correspond à sigma
    Else
      pp(ii) = 0.01 'on initialise en fonction des données disponibles (normalement
on ne passe pas ici si le minimum est bien trouvé)
    End If
    compar = 1# 'il semble prudent de comparer l'estimation aux suggestions

```

```

wt(kk) de l'utilisateur
Ncompar = 0
For kk = 1 To Npar 'on compare la projection de pp(ii) sur chaque xp(kk) au
wt(kk) correspondant
  If (wt(kk) <> 0) And (xs(ii, kk) <> 0) Then
    compar = compar * Abs(pp(ii) * xs(ii, kk) / wt(kk)) 'négatifs possibles
    Ncompar = Ncompar + 1
  End If
Next kk
compar = compar ^ (1# / Ncompar) 'on calcule le coefficient de comparaison
moyen
compar = compar ^ (1 - 4 / (6 + Abs(Log(compar)))) 'modérateur si compar est
très différent de 1
pp(ii) = pp(ii) / compar 'proposition rectifiée
Next ii
NEStMx = 5
For jj = 1 To NEStMx ' on traite le calcul plusieurs fois pour améliorer
  For ii = 1 To NP
    ppt(ii) = pp(ii)
  Next ii

  GoToFroid = True 'pour redémarrage à froid
  Do While GoToFroid
    GoToFroid = voisinage(Npar, xp(), NP, yy(), invSig2(), pp(), xs(), AMin)
  Loop

  NESt = jj
  suffisant = True
  For ii = 1 To NP
    If (pp(ii) < 0.95 * ppt(ii) Or pp(ii) > 1.05 * ppt(ii)) Then
      suffisant = False
      Exit For
    End If
  Next ii
  If suffisant Then
    NElist = vbCrLf & "Calcul des incertitudes en " & CStr(NESt) & " étapes"
    Exit For
  ElseIf NESt = NEStMx Then
    NElist = vbCrLf & "Approximation des incertitudes en " & CStr(NESt) & "
étapes (NEStMx atteint)"
  End If
Next jj ' fin de répétition (deux fois suffisent souvent : le processus converge
rapidement)
For ll = 1 To Npar
  For mm = ll To Npar
    eet(ll, mm) = 0
    For ii = 1 To NP
      eet(ll, mm) = eet(ll, mm) + xs(ii, ll) * xs(ii, mm) / invSig2(ii)
    Next ii
    eet(ll, mm) = 2 * eet(ll, mm) ' eet = sigma2 pour les paramètres réels
    eet(mm, ll) = eet(ll, mm)
  Next mm
Next ll

```

```

    Next mm
Next ll ' calculs d'incertitudes sans dérivées
For ii = 1 To Npar
    pp(ii) = 0 'l'incertitude est nulle pour les paramètres non ajustés
    If eet(ii, ii) > 0 Then
        pp(ii) = Sqr(eet(ii, ii))
    End If
Next ii
ErrorsOK = True
incertitudes = NElist
End Function
*****

Public Function minimi(Npar As Integer, xpp() As Double, xpN() As String, wtt() As
Double, theStep As Double, epsi As Double, iimp As Integer, wErrors As Boolean) As
String
    'recherche le minimum d'une fonction (généralement un chi2)
    'le paramètre "step" est ici nommé "theStep" car le mot "step" est réservé comme
mot clé du visual basic
    Const al = 3# 'al et be sont utilisées par plusieurs procédures
    Const be = -0.4
    Const NStepQ = 5 'NStepQ est utilisée par plusieurs procédures
    Dim yy(1 To 30) As Double
    Dim xp(1 To 30) As Double 'on veut conserver les valeurs initiales pour restaurer si
besoin (on modifie par les cellules)
    Dim wt(1 To 30) As Double
    Dim Dirln(1 To 31) As Double
    Dim dd(1 To 31) As Double
    Dim invSig2(1 To 31) As Double
    Dim pp(1 To 31) As Double
    Dim xs(1 To 31, 1 To 31) As Double
    Dim eet(1 To 31, 1 To 31) As Double
    Dim AMin As Double
    Dim NFCN As Integer
    Dim NPFN As Integer
    Dim Y1 As Double
    Dim Y2 As Double
    Dim aa As Double
    Dim ErrorsOK As Boolean
    Dim jg As Integer 'pour transmettre le numéro du paramètre aux sous-procédures
    Dim NP As Integer
    Dim NEq As Integer
    Dim avr As Double
    Dim av As Double
    Dim ast As Double
    Dim ame As Double
    Dim dp As Double
    Dim AM As Double
    Dim NSt As Integer
    Dim NP1 As Integer
    Dim JM As Integer

```



```

Dim ii As Integer
Dim jj As Integer
Dim kk As Integer
Dim ExitMin As Boolean
Dim xn As Double
Dim dir As Double
Dim impr As Integer
Dim liste As String
Dim Elist As String
Dim NElist As String
Dim iarg As Integer
Dim jarg As Integer
For iarg = 1 To 30 'initialisation des variables globales
    yy(iarg) = 0# 'pour le principe : pour déboguer, doit logiquement être nul tant
qu'on n'en a pas besoin
    xp(iarg) = 0#
    wt(iarg) = 0#
Next iarg
For iarg = 1 To 31
    DirIn(iarg) = 0#
    dd(iarg) = 0#
    invSig2(iarg) = 0#
    pp(iarg) = 0#
    For jarg = 1 To 31
        xs(iarg, jarg) = 0#
        eet(iarg, jarg) = 0#
    Next jarg
Next iarg
AMin = 1000#
NFCN = 0
NPFN = 0
Y1 = 0#
Y2 = 0#
aa = 0#
ErrorsOK = False
liste = "MINIMI (minimisation sans dérivées) MINIMI"
liste = liste & Chr(10) & "Nombre de paramètres : " & CStr(Npar) 'vbCrLf équivaut
à Chr(10)
For ii = 1 To Npar
    xp(ii) = xpp(ii) 'valeurs initiales des paramètres
    wt(ii) = wtt(ii) 'valeurs initiales des incertitudes (algébriques pour éventuellement
orienter la recherche)
Next ii
NEq = 0 'Initialisation, premier appel de FCN
AMin = Range(MG_adrFCN).Value 'on ne calcule pas FCN, c'est fait par le
tableur : ici on se contente de récupérer la valeur initiale
NFCN = 1
NPFN = NFCN
aa = AMin
NSt = 0
avr = AMin

```

```

NP = 0 'nombre des paramètres effectivement ajustés
For ii = 1 To Npar
  If wt(ii) <> 0 Then
    NP = NP + 1
    For jj = 1 To Npar
      xs(NP, jj) = 0#
    Next jj
    invSig2(NP) = 0#
    xs(NP, ii) = 1#
    Dirln(NP) = wt(ii) * theStep
  End If
Next ii
liste = liste & vbCrLf & "Nombre de paramètres effectifs : " & CStr(NP)
liste = liste & vbCrLf & "Taille des pas : " & CStr(theStep)
liste = liste & vbCrLf & "Précision : " & CStr(eps)
If wErrors Then
  liste = liste & vbCrLf & "Analyse des incertitudes pour un chi2"
End If
liste = liste & vbCrLf & vbCrLf & "Valeurs initiales [ Pas relatif ]"
For ii = 1 To Npar
  liste = liste & vbCrLf & xpN(ii) & " : " & CStr(xp(ii))
  liste = liste & " [ D" & xpN(ii) & " : " & CStr(wt(ii)) & " ]"
Next ii
liste = liste & vbCrLf & vbCrLf & "Premier calcul de la quantité minimisée : Min = "
& CStr(AMin)
NP1 = NP + 1 'pour éviter de le recalculer trop souvent (une fois terminé le calcul
de NP)
If iimp = 0 Then
  impr = NP1 'on peut faire comme on veut, mais c'est généralement bien ainsi
Else
  impr = iimp
End If
invSig2(NP1) = 0#
ExitMin = False 'Début du voyage, on y est pour un bout de temps...
Elist = "" 'pour indiquer le diagnostic de fin
Do While Not ExitMin
'##### minimisation
  For ii = 1 To Npar
    xs(NP1, ii) = xs(1, ii)
  Next ii
  Dirln(NP1) = Dirln(1)
  AM = 0
  JM = 0
  For jg = 1 To NP1 '##### série de NP1
steps, au travail !
    Call descente(Npar, xp(), jg, yy(), Dirln(), dd(), xs(), AMin, NFCN, NPFN, Y1,
Y2, aa)
    If Y1 < 0 And Y2 < 0 Then
      Call parabole(Npar, xp(), jg, False, yy(), Dirln(), dd(), xs(), AMin, NFCN,
NPFN, Y1, Y2, aa)
    ElseIf Not (Y1 = 0 And Y2 = 0) Then

```

```

    Call parabole(Npar, xp(), jg, True, yy(), Dirln(), dd(), xs(), AMin, NFCN,
NPFN, Y1, Y2, aa)
End If
NSt = NSt + 1
avr = avr - AMin
avr = AMin
If jg < 2 Then 'pas d'amélioration possible au premier essai
    ast = AMin
Elseif av > AM Then
    AM = av
    JM = jg
End If
If (NSt Mod impr) = 0 Then 'valeurs intermédiaires
    liste = liste & SorInt(NSt, Npar, xp(), xpN(), AMin)
End If
Next jg ##### la série est terminée, on
agite les mains...
ame = AMin - ast
If ame >= 0 Then
    ExitMin = True 'déclenche la fin du while
    Elist = vbCrLf & "Le minimum n'a pas été amélioré à la dernière étape"
    Exit Do 'provoque la sortie directe du while
End If
If epsi + ame < 0 Then
    NEq = -1
End If
NEq = NEq + 1
If NEq >= NStepQ Then
    ExitMin = True 'déclenche la fin du while
    Elist = vbCrLf & "Le minimum n'a pas été amélioré de " & CStr(epsi) & "
après chacune des " & CStr(NStepQ) & " dernières étapes"
    Exit Do 'provoque la sortie directe du while
End If
For kk = 1 To Npar
    pp(kk) = 0#
    For ii = 2 To NP1
        pp(kk) = pp(kk) + dd(ii) * xs(ii, kk)
    Next ii
Next kk
dp = 1#
For ii = 1 To Npar
    yy(ii) = xp(ii) + pp(ii)
Next ii
For ii = 1 To Npar
    Range(MG_adrPar(ii)).Value = yy(ii) 'on ne calcule pas FCN, c'est fait par le
tableur : il faut modifier les paramètres
Next ii
Calculate 'puis il faut forcer (au cas où...) le recalcul
aa = Range(MG_adrFCN).Value 'on peut enfin récupérer FCN
NFCN = NFCN + 1
While aa < AMin

```

```

Amin = aa
dp = 2 * dp
For ii = 1 To Npar
    xp(ii) = yy(ii)
    pp(ii) = 2 * pp(ii)
    yy(ii) = xp(ii) + pp(ii)
Next ii
For ii = 1 To Npar
    Range(MG_adrPar(ii)).Value = yy(ii) 'on ne calcule pas FCN, c'est fait par
le tableur : il faut modifier les paramètres
Next ii
Calculate 'puis il faut forcer (au cas où...) le recalcul
aa = Range(MG_adrFCN).Value 'on peut enfin récupérer FCN
NFCN = NFCN + 1
Wend
dp = (aa + ast - 2 * Amin) / (2 * dp * dp)
If dp <= AM Then
    If JM <= NP Then
        For ii = JM To NP
            invSig2(ii) = invSig2(ii + 1)
            Dirln(ii) = Dirln(ii + 1)
            For jj = 1 To Npar
                xs(ii, jj) = xs(ii + 1, jj)
            Next jj
        Next ii
    End If
    xn = 0
    For kk = 1 To Npar
        xs(1, kk) = pp(kk)
        xn = xn + xs(1, kk) * xs(1, kk)
    Next kk
    dir = Sqr(xn)
    Dirln(1) = dir
    invSig2(NP1) = 2 * dp
    For kk = 1 To Npar
        xs(1, kk) = xs(1, kk) / dir
    Next kk
End If
Loop '##### while - minimisation (retour au début et
on recommence...)
ErrorsOK = False
If (wErrors And (NSt >= Npar * NP)) Then
    NElist = incertitudes(Npar, xp(), wt(), NP, yy(), invSig2(), pp(), xs(), eet(), Amin,
ErrorsOK)
End If
'encore quelques impressions avant d'aller se coucher (la journée a été dure)
liste = liste & vbCrLf & vbCrLf & "La minimisation est terminée"
liste = liste & Elist
liste = liste & vbCrLf & vbCrLf & "La plus faible valeur est : Min = " & CStr(Amin)
liste = liste & " ( pour l'entrée : " & CStr(NPFN) & " )"
If Not ErrorsOK Then

```

```

If wErrors Then
    liste = liste & vbCrLf & vbCrLf & "Le nombre de pas est insuffisant pour
calculer les incertitudes"
End If
liste = liste & SorInt(NSt, Npar, xp(), xpN(), AMin) 'impression des paramètres
sans incertitudes
Else
    liste = liste & NElist
    liste = liste & vbCrLf & vbCrLf & "Paramètres [ Déviations standard ]"
    For ii = 1 To Npar
        liste = liste & vbCrLf & CStr(xpN(ii)) & " : " & CStr(xp(ii))
        liste = liste & " [ D" & CStr(xpN(ii)) & " : " & CStr(pp(ii)) & " ]"
        For jj = 1 To ii - 1
            If pp(ii) * pp(jj) <> 0 Then
                liste = liste & vbCrLf & "Cov[" & CStr(xpN(ii)) & ", "
                liste = liste & CStr(xpN(jj)) & "]" : "
                liste = liste & CStr(eet(ii, jj)) & " ; Cor["
                liste = liste & CStr(xpN(ii)) & ", " & CStr(xpN(jj))
                liste = liste & "]" : " & CStr(eet(ii, jj) / (pp(ii) * pp(jj)))
            End If
        Next jj
    Next ii
End If
liste = liste & vbCrLf & vbCrLf & "Statistique de la minimisation : nombre d'entrées
= " & CStr(NFCN)
liste = liste & " ; nombre de pas = " & CStr(NSt)
'c'est fini, bonsoir...
minimi = liste
End Function

```
