Subject: Need 2-dimensional Y versus X curve fit routine
Posted by jmcfee on Thu, 10 Jul 1997 07:00:00 GMT
View Forum Message <> Reply to Message

CURVEFIT in the IDL built-in library which performs a nonlinear fit of a 1-D data vector Y to a function of a 1-D vector X and a parameter vector A. I am looking for a routine similar except that Y and X are two dimensional (time versus depth). CURVEFIT is based on Bevington's CURVFIT (Marquardt's method). I extended a fortran CURVFIT routine to 2-D some years ago, but hate to repeat the effort in IDL. Has anybody got a 2-D version?

I've already checked the searchable IDL library website with no luck.

Thanks for the help.

--
Dr. John E. McFee -- Usual disclaimer applies.
Threat Detection Group -- "They can kill you -
Defence Research Est. Suffield -- but they're not allowed to eat you."
John.McFee@dres.dnd.ca -- - Chuck Jones
--
Dr. John E. McFee
Defence Research Establishment Suffield
Box 4000, Medicine Hat, AB Canada T1A 8K6
(403) 544-4739 (voice) 544-4704 (fax)

Subject: Re: Need 2-dimensional Y versus X curve fit routine
Posted by thompson on Mon, 14 Jul 1997 07:00:00 GMT
View Forum Message <> Reply to Message

jmcfee@dres.dnd.ca (John McFee) writes:

> CURVEFIT in the IDL built-in library which performs a nonlinear fit of a 1-D
> data vector Y to a function of a 1-D vector X and a parameter vector A. I am
> looking for a routine similar except that Y and X are two dimensional (time
> versus depth). CURVEFIT is based on Bevington's CURVFIT (Marquardt's
> method). I extended a fortran CURVFIT routine to 2-D some years ago, but hate
> to repeat the effort in IDL. Has anybody got a 2-D version?

If the problem is that of fitting to a scalar field which is a function of two variables, e.g. height as a function of two spatial coordinates, then CURVEFIT as it is currently written should be able to handle it. The trick is to pass in your coordinates as a structure with the elements of the structure being the two coordinates that the variable being fitted is dependent on. Mathematically this would be expressed as

Y = F(X1, X2)

and in IDL you could define the structure as

```
 POS = REPLICATE( {X1: 0.0, X2: 0.0}, N_ELEMENTS(X1) )
 POS.X1 = X1
 POS.X2 = X2
```

This procedure could be generalized to any number of dimensions.

If instead your problem is that you have two dependent variables which are both related to the same independent variables, e.g.

```
 Y1 = F1(X1, X2)
 Y2 = F2(X1, X2)
```

and you want to fit to both variables simultaneously, then you would have to modify CURVEFIT.

If the latter is the case, then one routine that might help is a program I once wrote called AMOEBA_F. It's based on the algorithm Amoeba outlined in Numerical Recipes. This should work because you pass in the function to be minimized, i.e. your equivalent of chi-squared, insteading of passing the functions F1 and F2.

This routine may call some other routines, which should be available from

 ftp://sohoftp.nascom.nasa.gov/solarsoft/gen/idl/fitting/

and parallel directories.

Bill

```
 ============================================================
==================
 PRO AMOEBA_F,FNAME,PARAM,ACCURACY=ACCURACY,MAX_ITER=MAX_ITER, $
  LAMBDA=LAMBDA0,CHISQR=CHISQR,N_ITER=ITER,NOPRINT=NOPRINT
;+
; Project     : SOHO - CDS
;
; Name        : AMOEBA_F
;
; Purpose     : Reiteratively minimizes an arbitrary function
;
; Explanation : Minimizes an arbitrary function via a least-squares reiterative
;  technique.
;
;  The procedure used is taken from Numerical Recipes.
;
; Use         : AMOEBA_F, FNAME, PARAM
```

```
;
; Inputs      : FNAME = Name of function to be minimized (string variable).
;  PARAM = Parameters of fit.  Passed as first guess.  Returned
;     as fitted values.
;
; Opt. Inputs : None.
;
; Outputs     : PARAM = Parameters of fit.  See note above.
;
; Opt. Outputs: None.
;
; Keywords    : ACCURACY = Accuracy to cut off at.  Defaults to 1E-5.
;  MAX_ITER = Maximum number of reiterations.  Defaults to 20.
;  LAMBDA  = Initial step sizes for PARAM, or if scalar then
;      fraction of PARAM.  Defaults to 1E-2.
;  NOPRINT  = If set, then no printout is generated.
;  CHISQR  = Returned value of chi-squared.  Only relevant if
;      ERROR passed explicitly.
;  N_ITER  = Number of iterations used.
;
; Calls       : None.
;
; Common      : None
;
; Restrictions: The user defined function is passed by name as a character
;  string in the variable FNAME.  The function must have the form.
;
;        Y = F(PARAM)
;
;  where PARAM is the vector containing the parameters of the fit.
;
;  Unless LAMBDA is passed as an array, the initial guess for
;  PARAM must not contain any zeroes.
;
; Side effects: None.
;
; Category    : Utilities, Curve_Fitting
;
; Prev. Hist. : William Thompson, August, 1989.
;  William Thompson, June 1991, modified to use keywords.
;
; Written     : William Thompson, GSFC, August 1989
;
; Modified    : Version 1, William Thompson, GSFC, 9 January 1995
;   Incorporated into CDS library
;  Version 2, William Thompson, GSFC, 6 October 1995
;   Fixed typo.
;
```

```
; Version     : Version 2, 6 October 1995
;-
;
 ON_ERROR,2
;
;  Check the number of parameters passed.
;
 IF N_PARAMS(0) NE 2 THEN BEGIN
  PRINT,' This procedure must be called with two parameters:'
  PRINT,'                  FNAME, PARAMS'
  RETURN
 ENDIF
;
;  Set up the default parameters.
;
 ACCUR = 1E-5
 IF N_ELEMENTS(ACCURACY) EQ 1 THEN $
  IF ACCURACY GT 0 THEN ACCUR = ACCURACY
;
 IF N_ELEMENTS(MAX_ITER) EQ 1 THEN NMAX = MAX_ITER ELSE NMAX = 20
;
 LAMBDA = 1E-2
 IF N_ELEMENTS(LAMBDA0) EQ 1 THEN $
  IF LAMBDA0 GT 0 THEN LAMBDA = LAMBDA0
;
;  Define NPAR from the input array.  Start ITER at zero.
;
 NPAR = N_ELEMENTS(PARAM)
 IF NPAR EQ 1 THEN PARAM = MAKE_ARRAY(PARAM)
 ITER = 0
;
;  Check the array or scalar LAMBDA.
;
 NLAMB = N_ELEMENTS(LAMBDA)
 IF (NLAMB NE NPAR) AND (NLAMB NE 1) THEN BEGIN
  PRINT,'*** LAMBDA must have 1 or ' + FIX(NPAR) + $
   ' elements, routine AMOEBA_F.'
  RETURN
 ENDIF
;
;  Calculate the array of parameters to start with.
;
 P = PARAM # REPLICATE(1.,NPAR+1)
 FOR I = 0,NPAR-1 DO BEGIN
  IF NLAMB EQ NPAR THEN P(I,I+1) = PARAM(I) + LAMBDA(I) $
   ELSE P(I,I+1) = PARAM(I) * (1. + LAMBDA)
 ENDFOR
;
```

```
;  Initialize the array containing chi-squared.
;
 CHI = 0.*[0,PARAM]
 FOR I = 0,NPAR DO BEGIN
  TEST = EXECUTE('CHI(I) = ' + FNAME + '(P(*,I))')
 ENDFOR
;
;  Print the header.
;
 IF (NMAX GT 0) AND (NOT KEYWORD_SET(NOPRINT)) THEN BEGIN
  PRINT,FORMAT="(1H1,3X,'I',10X,'log',20X,'log',16X,'I')"
  PRINT,FORMAT="(1X,I4,9X,'ERROR',17X,'CHISQR',7X,I9)",NMAX,NMAX
  PRINT,FORMAT="(2X,4(1H-),5X,10(1H-),5X,25(1H-),4X,4(1H-))"
 ENDIF
;
;  Starting point for all iterations.  Find the highest, next highest, and
;  lowest values of CHI.
;
ITERATE:
 ITER = ITER + 1
 ILO = 0
 IF CHI(0) GT CHI(1) THEN BEGIN
  IHI  = 0
  INHI = 1
 END ELSE BEGIN
  IHI  = 1
  INHI = 0
 ENDELSE
 FOR I = 0,NPAR DO BEGIN
  IF CHI(I) LT CHI(ILO) THEN ILO = I
  IF CHI(I) GT CHI(IHI) THEN BEGIN
   INHI = IHI
   IHI  = I
  END ELSE IF (CHI(I) GT CHI(INHI)) AND (I NE IHI) THEN INHI = I
 ENDFOR
;
;  Find the average of P for all points other than IHI.
;
 PBAR = 0*PARAM
 FOR I = 0,NPAR DO IF I NE IHI THEN PBAR = PBAR + P(*,I)
 PBAR = PBAR / NPAR
;
;  Reflect from the high point through PBAR.
;
 PR = 2*PBAR - P(*,IHI)
 TEST = EXECUTE('CR = ' + FNAME + '(PR)')
;
;  If an improvement was achieved, try an additional extrapolation.
```

```
;
 IF CR LE CHI(ILO) THEN BEGIN
  PRR = 2*PR - PBAR
  TEST = EXECUTE('CRR = ' + FNAME + '(PRR)')
;
;  If an additional improvement was achieved, use the second extrapolation.
;
  IF CRR LT CHI(ILO) THEN BEGIN
   P(0,IHI) = PRR
   CHI(IHI) = CRR
;
;  Otherwise use the first extrapolation (reflection).
;
  END ELSE BEGIN
   P(0,IHI) = PR
   CHI(IHI) = CR
  ENDELSE
;
;  If the reflection yields a value between the highest value and the next
;  highest value, use it.
;
 END ELSE IF CR GE CHI(INHI) THEN BEGIN
  IF CR LT CHI(IHI) THEN BEGIN
   P(0,IHI) = PR
   CHI(IHI) = CR
  ENDIF
;
;  Look for an intermediate lower point.  If it's an improvement use it.
;
  PRR = (P(*,IHI) + PBAR) / 2.
  TEST = EXECUTE('CRR = ' + FNAME + '(PRR)')
  IF CRR LT CHI(IHI) THEN BEGIN
   P(0,IHI) = PRR
   CHI(IHI) = CRR
;
;  Nothing seems to help.  Contract about the lowest point.
;
  END ELSE BEGIN
   FOR I = 0,NPAR DO IF I NE ILO THEN BEGIN
    PR = (P(*,I) + P(*,ILO)) / 2.
    P(0,I) = PR
    TEST = EXECUTE('CHI(I) = ' + FNAME + '(PRR)')
   ENDIF
  ENDELSE
;
;  The reflection yields an intermediate value.  Use it.
;
 END ELSE BEGIN
```

```
  P(0,IHI) = PR
  CHI(IHI) = CR
 ENDELSE
;
;  Print out the iteration information.
;
 ERROR = TOTAL( (P(*,IHI) - P(*,ILO))^2 / (P(*,IHI)^2 + P(*,ILO)^2) )
 BANG_C = !C
 CHI_HI = ABS(MAX(CHI))
 CHI_LO = ABS(MIN(CHI))
 I_LOW = !C
 !C = BANG_C
 IF ERROR  GT 0 THEN ERRORLG = ALOG10(ERROR)/2 ELSE ERRORLG = -999
 IF CHI_HI GT 0 THEN CHILOG  = ALOG10(CHI_HI)  ELSE CHILOG  = -999
 IF CHI_LO GT 0 THEN CLOLOG  = ALOG10(CHI_LO)  ELSE CLOLOG  = -999
 FORMAT = "(I5,3F15.5,I8)"
 IF NOT KEYWORD_SET(NOPRINT) THEN PRINT,FORMAT=FORMAT,ITER,ERRORLG, $
  CHILOG,CLOLOG,ITER
 IF (ERROR GT ACCUR^2) AND (ITER LT NMAX) THEN GOTO,ITERATE
;
;  The program has either converged or reached its maximum number of
;  iterations.
;
 PARAM  = P(*,I_LOW)
 CHISQR = CHI(I_LOW)
 END
```

Subject: Re: Need 2-dimensional Y versus X curve fit routine
Posted by thompson on Mon, 14 Jul 1997 07:00:00 GMT
View Forum Message <> Reply to Message

jmcfee@dres.dnd.ca (John McFee) writes:

> CURVEFIT in the IDL built-in library which performs a nonlinear fit of a 1-D
> data vector Y to a function of a 1-D vector X and a parameter vector A. I am
> looking for a routine similar except that Y and X are two dimensional (time
> versus depth). CURVEFIT is based on Bevington's CURVFIT (Marquardt's
> method). I extended a fortran CURVFIT routine to 2-D some years ago, but hate
> to repeat the effort in IDL. Has anybody got a 2-D version?

If the problem is that of fitting to a scalar field which is a function of two
variables, e.g. height as a function of two spatial coordinates, then CURVEFIT
as it is currently written should be able to handle it. The trick is to pass
in your coordinates as a structure with the elements of the structure being the
two coordinates that the variable being fitted is dependent on. Mathematically

this would be expressed as

 Y = F(X1, X2)

and in IDL you could define the structure as

 POS = REPLICATE({X1: 0.0, X2: 0.0}, N_ELEMENTS(X1))
 POS.X1 = X1
 POS.X2 = X2

This procedure could be generalized to any number of dimensions.

If instead your problem is that you have two dependent variables which are both related to the same independent variables, e.g.

 Y1 = F1(X1, X2)
 Y2 = F2(X1, X2)

and you want to fit to both variables simultaneously, then you would have to modify CURVEFIT.

If the latter is the case, then one routine that might help is a program I once wrote called AMOEBA_F. It's based on the algorithm Amoeba outlined in Numerical Recipes. This should work because you pass in the function to be minimized, i.e. your equivalent of chi-squared, insteading of passing the functions F1 and F2.

This routine may call some other routines, which should be available from

 ftp://sohoftp.nascom.nasa.gov/solarsoft/gen/idl/fitting/

and parallel directories.

Bill

```
 ============================================================
==================
 PRO AMOEBA_F,FNAME,PARAM,ACCURACY=ACCURACY,MAX_ITER=MAX_ITER, $
  LAMBDA=LAMBDA0,CHISQR=CHISQR,N_ITER=ITER,NOPRINT=NOPRINT
;+
; Project     : SOHO - CDS
;
; Name        : AMOEBA_F
;
; Purpose     : Reiteratively minimizes an arbitrary function
;
; explanation : minimizes an arbitrary function via a least-squares reiterative
;  technique.
```

```
;
;  the procedure used is taken from numerical recipes.
;
; use         : amoeba_f, fname, param
;
; inputs      : fname = name of function to be minimized (string variable).
;  param = parameters of fit.  passed as first guess.  returned
;     as fitted values.
;
; opt. inputs : none.
;
; outputs     : param = parameters of fit.  see note above.
;
; opt. outputs: none.
;
; keywords    : accuracy = accuracy to cut off at.  defaults to 1e-5.
;  max_iter = maximum number of reiterations.  defaults to 20.
;  lambda  = initial step sizes for param, or if scalar then
;      fraction of param.  defaults to 1e-2.
;  noprint  = if set, then no printout is generated.
;  chisqr  = returned value of chi-squared.  only relevant if
;      error passed explicitly.
;  n_iter  = number of iterations used.
;
; calls       : none.
;
; common      : none
;
; restrictions: the user defined function is passed by name as a character
;  string in the variable fname.  the function must have the form.
;
;       y = f(param)
;
;  where param is the vector containing the parameters of the fit.
;
;  unless lambda is passed as an array, the initial guess for
;  param must not contain any zeroes.
;
; side effects: none.
;
; category    : utilities, curve_fitting
;
; prev. hist. : william thompson, august, 1989.
;  william thompson, june 1991, modified to use keywords.
;
; written     : william thompson, gsfc, august 1989
;
; modified    : version 1, william thompson, gsfc, 9 january 1995
```

```
;   incorporated into cds library
;  version 2, william thompson, gsfc, 6 october 1995
;   fixed typo.
;
; version     : version 2, 6 october 1995
;-
;
 on_error,2
;
;  check the number of parameters passed.
;
 if n_params(0) ne 2 then begin
  print,' this procedure must be called with two parameters:'
  print,'                  fname, params'
  return
 endif
;
;  set up the default parameters.
;
 accur = 1e-5
 if n_elements(accuracy) eq 1 then $
  if accuracy gt 0 then accur = accuracy
;
 if n_elements(max_iter) eq 1 then nmax = max_iter else nmax = 20
;
 lambda = 1e-2
 if n_elements(lambda0) eq 1 then $
  if lambda0 gt 0 then lambda = lambda0
;
;  define npar from the input array.  start iter at zero.
;
 npar = n_elements(param)
 if npar eq 1 then param = make_array(param)
 iter = 0
;
;  check the array or scalar lambda.
;
 nlamb = n_elements(lambda)
 if (nlamb ne npar) and (nlamb ne 1) then begin
  print,'*** lambda must have 1 or ' + fix(npar) + $
   ' elements, routine amoeba_f.'
  return
 endif
;
;  calculate the array of parameters to start with.
;
 p = param # replicate(1.,npar+1)
 for i = 0,npar-1 do begin
```

```
  if nlamb eq npar then p(i,i+1) = param(i) + lambda(i) $
   else p(i,i+1) = param(i) * (1. + lambda)
 endfor
;
;  initialize the array containing chi-squared.
;
 chi = 0.*[0,param]
 for i = 0,npar do begin
  test = execute('chi(i) = ' + fname + '(p(*,i))')
 endfor
;
;  print the header.
;
 if (nmax gt 0) and (not keyword_set(noprint)) then begin
  print,format="(1h1,3x,'i',10x,'log',20x,'log',16x,'i')"
  print,format="(1x,i4,9x,'error',17x,'chisqr',7x,i9)",nmax,nmax
  print,format="(2x,4(1h-),5x,10(1h-),5x,25(1h-),4x,4(1h-))"
 endif
;
;  starting point for all iterations.  find the highest, next highest, and
;  lowest values of chi.
;
iterate:
 iter = iter + 1
 ilo = 0
 if chi(0) gt chi(1) then begin
  ihi  = 0
  inhi = 1
 end else begin
  ihi  = 1
  inhi = 0
 endelse
 for i = 0,npar do begin
  if chi(i) lt chi(ilo) then ilo = i
  if chi(i) gt chi(ihi) then begin
   inhi = ihi
   ihi  = i
  end else if (chi(i) gt chi(inhi)) and (i ne ihi) then inhi = i
 endfor
;
;  find the average of p for all points other than ihi.
;
 pbar = 0*param
 for i = 0,npar do if i ne ihi then pbar = pbar + p(*,i)
 pbar = pbar / npar
;
;  reflect from the high point through pbar.
;
```

```
 pr = 2*pbar - p(*,ihi)
 test = execute('cr = ' + fname + '(pr)')
;
;  if an improvement was achieved, try an additional extrapolation.
;
 if cr le chi(ilo) then begin
  prr = 2*pr - pbar
  test = execute('crr = ' + fname + '(prr)')
;
;  if an additional improvement was achieved, use the second extrapolation.
;
  if crr lt chi(ilo) then begin
   p(0,ihi) = prr
   chi(ihi) = crr
;
;  otherwise use the first extrapolation (reflection).
;
  end else begin
   p(0,ihi) = pr
   chi(ihi) = cr
  endelse
;
;  if the reflection yields a value between the highest value and the next
;  highest value, use it.
;
 end else if cr ge chi(inhi) then begin
  if cr lt chi(ihi) then begin
   p(0,ihi) = pr
   chi(ihi) = cr
  endif
;
;  look for an intermediate lower point.  if it's an improvement use it.
;
  prr = (p(*,ihi) + pbar) / 2.
  test = execute('crr = ' + fname + '(prr)')
  if crr lt chi(ihi) then begin
   p(0,ihi) = prr
   chi(ihi) = crr
;
;  nothing seems to help.  contract about the lowest point.
;
  end else begin
   for i = 0,npar do if i ne ilo then begin
    pr = (p(*,i) + p(*,ilo)) / 2.
    p(0,i) = pr
    test = execute('chi(i) = ' + fname + '(prr)')
   endif
  endelse
```

```
;
;  the reflection yields an intermediate value.  use it.
;
 end else begin
  p(0,ihi) = pr
  chi(ihi) = cr
 endelse
;
;  print out the iteration information.
;
 error = total( (p(*,ihi) - p(*,ilo))^2 / (p(*,ihi)^2 + p(*,ilo)^2) )
 bang_c = !c
 chi_hi = abs(max(chi))
 chi_lo = abs(min(chi))
 i_low = !c
 !c = bang_c
 if error  gt 0 then errorlg = alog10(error)/2 else errorlg = -999
 if chi_hi gt 0 then chilog  = alog10(chi_hi)  else chilog  = -999
 if chi_lo gt 0 then clolog  = alog10(chi_lo)  else clolog  = -999
 format = "(i5,3f15.5,i8)"
 if not keyword_set(noprint) then print,format=format,iter,errorlg, $
  chilog,clolog,iter
 if (error gt accur^2) and (iter lt nmax) then goto,iterate
;
;  the program has either converged or reached its maximum number of
;  iterations.
;
 param  = p(*,i_low)
 chisqr = chi(i_low)
 end
```