
Subject: Re: Smoothing Spline -- any existing efficient routines?

Posted by [Nikola](#) on Mon, 16 Aug 2010 10:11:02 GMT

[View Forum Message](#) <> [Reply to Message](#)

You can try this one. It's not very nicely coded - I wrote it as an exercise in my early IDL days - but it might work.

```
;+
; NAME:
; SPLINECOEFF
;
; PURPOSE:
; This procedure computes coefficients of cubic splines
; for a given observational set and smoothing parameter
; lambda. The method is coded according to Pollock D.S.G.
; (1999), "A Handbook of Time-Series Analysis, Signal
; Processing and Dynamics, Academic Press", San Diego
;
; CATEGORY:
; Data processing
;
; CALLING SEQUENCE:
; COEFFS = SPLINECOEFF([X,] Y, [SIGMA], LAMBDA=LAMBDA)
;
; INPUTS:
; X = 1D Array (independent variable)
; Y = 1D Array (function)
; SIGMA = 1D Array (weight of each measurement) By default
; all the measurements are of the same weight.
;
; KEYWORDS:
; LAMBDA = Smoothing parameter (It can be determined
; empirically, by the LS method or by cross-
; validation, eg. see book of Pollock.) LAMBDA
; equals 0 results in a cubic spline interpolation.
; In the other extreme, for a very large LAMBDA
; the result is smoothing by a linear function.
;
; COMMENT:
;
; EXAMPLE:
; X = ....
; Y = ....
; Coeffs = SPLINECOEFF(X, Y, LAMBDA = 1.d5)
; Y1 = N_ELEMENTS(Y) - 1
; X1 = X(0:N_ELEMENTS(Y)-2)
; FOR i = 0, N_ELEMENTS(Y)-2 DO Y1(i) = Coeff.D(I) + $
; Coeff.C(I) * (X(I+1)-X(I)) + $
```

```

;          Coeff.B(I) * (X(I+1)-X(I))^2 + $
;          Coeff.A(I) * (X(I+1)-X(I))^3
; PLOT, X, Y, PSYM = 3
; OPLOT, X1, Y1
;
; OUTPUTS:
; COEFFS: Structure of 4 arrays (A, B, C & D) containing
;          the coefficients of a spline between each two of
;          the given measurements.
;
; MODIFICATION HISTORY:
; Written by: NV (Jan2006)
;           # as a function, NV (Mar2007)
;-
FUNCTION SPLINECOEFF, XX, YY, SS, LAMBDA = LAMBDA

CASE N_PARAMS() OF
 1: BEGIN
  Y = XX
  X = INDGEN(N_ELEMENTS(Y))
  SIGM = FLTARR(N_ELEMENTS(Y))+1
END
2: BEGIN
  Y = YY
  X = XX
  SIGM = FLTARR(N_ELEMENTS(Y))+1
END
3: BEGIN
  Y = YY
  X = XX
  SIGM = SS
END
ELSE: MESSAGE, 'Wrong number of arguments'
ENDCASE

NUM = SIZE(X, /N_ELEMENTS)
N = NUM-1
IF NOT(KEYWORD_SET(LAMBDA)) THEN MESSAGE, 'Parameter lambda is not
defined.'

; Definition of the help variables
H = DBLARR(NUM) & R = DBLARR(NUM) & F = DBLARR(NUM) & P = DBLARR(NUM)
Q = DBLARR(NUM) & U = DBLARR(NUM) & V = DBLARR(NUM) & W = DBLARR(NUM)
; Definition of the unknown coefficients
A = DBLARR(NUM) & B = DBLARR(NUM) & C = DBLARR(NUM) & D = DBLARR(NUM)

; Computation of the starting values
H(0) = X(1) - X(0)

```

```

R(0) = 3.D/H(0)

; Computation of all H, R, F, P & Q
FOR I = 1, N - 1 DO BEGIN
  H(I) = X(I+1) - X(I)
  R(I) = 3.D/H(I)
  F(I) = -(R(I-1) + R(I))
  P(I) = 2.D * (X(I+1) - X(I-1))
  Q(I) = 3.D * (Y(I+1) - Y(I))/H(I) - 3.D * (Y(I) - Y(I-1))/H(I-1)
ENDFOR

; Compute diagonals of the matrix: W + LAMBDA T' SIGMA T
FOR I = 1, N - 1 DO BEGIN
  U(I) = R(I-1)^2 * SIGM(I-1) + F(I)^2 * SIGM(I) + R(I)^2 * SIGM(I+1)
  U(I) = LAMBDA * U(I) + P(I)
  V(I) = F(I) * R(I) * SIGM(I) + R(I) * F(I+1) * SIGM(I+1)
  V(I) = LAMBDA * V(I) + H(I)
  W(I) = LAMBDA * R(I) * R(I+1) * SIGM(I+1)
ENDFOR

; Decomposition in the form L' D L
V(1) = V(1)/U(1)
W(1) = W(1)/U(1)

FOR J = 2, N-1 DO BEGIN
  U(J) = U(J) - U(J-2) * W(J-2)^2 - U(J-1) * V(J-1)^2
  V(J) = (V(J) - U(J-1) * V(J-1) * W(J-1))/U(J)
  W(J) = W(J)/U(J)
ENDFOR

; Gaussian eliminations to solve Lx = T'y
Q(0) = 0.D
FOR J = 2, N-1 DO Q(J) = Q(J) - V(J-1) * Q(J-1) - W(J-2) * Q(J-2)
FOR J = 1, N-1 DO Q(J) = Q(J)/U(J)

; Gaussian eliminations to solve L'c = D^{-1}x
Q(N-2) = Q(N-2) - V(N-2)*Q(N-1)
FOR J = N-3, 1, -1 DO Q(J) = Q(J) - V(J) * Q(J+1) - W(J) * Q(J+2)

; Coefficients in the first segment
D(0) = Y(0) - LAMBDA * R(0) * Q(1) * SIGM(0)
D(1) = Y(1) - LAMBDA * (F(1) * Q(1) + R(1) * Q(2)) * SIGM(0)
A(0) = Q(1)/(3.D * H(0))
B(0) = 0.D
C(0) = (D(1) - D(0))/H(0) - Q(1) * H(0)/3.D

; Other coefficients
FOR J = 1, N-1 DO BEGIN

```

```
A(J) = (Q(J+1)-Q(J))/(3.D * H(J))
B(J) = Q(J)
C(J) = (Q(J) + Q(J-1)) * H(J-1) + C(J-1)
D(J) = R(J-1) * Q(J-1) + F(J) * Q(J) + R(J) * Q(J+1)
D(J) = Y(J) - LAMBDA * D(J) * SIGM(J)
ENDFOR
D(N) = Y(N) - LAMBDA * R(N-1) * Q(N-1) * SIGM(N)
```

```
SplCoeff = {A:DBLARR(NUM), B:DBLARR(NUM), C:DBLARR(NUM),
D:DBLARR(NUM)}
```

```
SplCoeff.A = A
```

```
SplCoeff.B = B
```

```
SplCoeff.C = C
```

```
SplCoeff.D = D
```

```
RETURN, SPLCOEFF
```

```
END
```
