
Subject: Re: Smoothing Data with optimal wiener filter
Posted by [Chris\[1\]](#) on Mon, 24 May 2004 17:15:43 GMT
[View Forum Message](#) <> [Reply to Message](#)

Ralf,

The derivatives that your code makes look okay. What makes you think they're wrong? I ran your program and then checked the derivatives against the simplest of all approximations, like this (for the first derivative, analogous for the second):

```
IDL> plot, x, (smoothed_data[1:*] - smoothed_data) / (x[1:*] - x)
IDL> oplot, x, first_diff, linestyle=2
```

In both cases (1st and 2nd derivative), the curves more or less match one another, as should be expected. Where's the problem? Maybe I overlooked something.

Timm

--

Timm Weitkamp <<http://people.web.psi.ch/weitkamp>>

Today at 09:31 -0700, Ralf Schaa wrote:

```
> Hi folks,
>
> i have written a program, which smoothes noisy data with a wiener
> filter. Return values of the program are the smoothed dataset and the
> first and second derivatives.
> However, the smoothing works satisfying, but the derivatives are going
> *very* wrong and i could use some suggestions here.
>
> You find the code below.
> Any suggestions would be appreciate very much!
>
> -Ralf
>
> P.S:
> The program has not an elaborated description as it should have,
> but wo is interested: the program is based on the paper by:
> E.L Kosarev and E. Pantos, J. Phys. E: Vol. 16, 1983
> 'Optimal Smoothing of noisy data by FFT'
> downloadable from http://kapitza.ras.ru/people/kosarev/1983.pdf
>
> A Fortran-code is in Int. J. Biomed. Comput. 37 (1994) 57-76 by F.
> Gazzani
```

```

> 'Comparative assessment of some algorithms for differentiating noisy
> biomechanical data'
>
> Some basics of the topic found in 'Anderssen, Bloomfield' in
> 'Numerische Mathematik' (not german but in english), Vol. 22, pp.
> 157-182, 1974
> 'Numerical Differentiation procedures for non-exact data'
>
>
> The program:
> =====
=====
> (Derivatives are calculated at line 181)
>
> PRO test_smooth_fft
>
> ;;-----
> ;; Create some simple Test-Timeseries
> ;;-----
>
> N = 100 ; Number of Samples
> T = 1.D ; Period
> Delt = T / Double(N) ; Sample interval
> Pi2 = (2.*!DPI) ; 2 x Pi
> OMEGA = Pi2 / (Float(N)*Delt) ; Angular Sample frequency
>
> ; Ordinate
> Y = DbfArr(N)
> FOR t = 0, N-1 DO BEGIN
>   Y[t] = ALOG(t+1) + SIN(t/Pi2)*COS(t/Pi2)
> ENDFOR
>
> ; Abscissa
> X = FindGen(N) * Delt
>
> ; Add noise:
> Dat_O = Y
> Dat = Y + RANDOMN(789421, n)*0.3D
>
>
>
> ;;-----
> ;; Center and Baselinefit of the data
> ;;-----
> Dat_m = Dat - Mean(Dat)
>
> Dat_b = FltArr(N)
> SLOPE = (Dat_m[N-1]-Dat_m[0])/(N-2)

```

```

> FOR j=0, N-1 DO BEGIN
>   Dat_b[j] = Dat_m[j] - Dat_m[0] - SLOPE * (j)
> ENDFOR
>
>
> ;;-----
> ;; Compute fft- / power- spectrum
> ;;-----
> ; Fourier
> Dat_F = FFT(Dat_b) ;HANNING(N, ALPHA=0.56)*
> ; Power
> Dat_P = ABS(Dat_F)^2
>
> ; Frequency for abscissa axis
> ; F = [0.0, 1.0/(N*delt), ... , 1.0/(2.0*delt)]:
> F = FINDGEN(N/2+1) / (N*delt)
>
>
> ;;-----
> ;; Noise spectrum from 'half' to 'full'
> ;; Mind: half means N/4, full means N/2
> ;;-----
> Sigma = Total(Dat_P[(N-1)/4:(N-1)/2])
> ; The noise is assumed to be the second half of the periodogram
> Noise = Sigma / ((N-1)/2 - (N-1)/4)
>
>
> ;;-----
> ;; Get Filtercoeff. according to Kosarev/Pantos
> ;; Find the J0, start search at i=1 (i=0 is the mean)
> ;;-----
> J0 = 2
> FOR i=1, N/4 DO BEGIN
>   Sigma0 = Dat_P[i]
>   Sigma1 = Dat_P[i+1]
>   Sigma2 = Dat_P[i+2]
>   Sigma3 = Dat_P[i+3]
>
>   IF( Sigma0 LT Noise AND $
>     (Sigma1 LT Noise OR $
>       Sigma2 LT Noise OR $
>         Sigma3 LT Noise)) $
>   THEN BEGIN
>     J0 = i
>     BREAK
>   ENDIF
> ENDFOR
>

```

```

> ;;-----
> ;; Compute straight line extrapolation to log(Dat_P)
> ;;-----
> XY = 0.
> XX = 0.
> S = 0.
> FOR i=1, J0 DO BEGIN
>   XY = XY + i * ALOG(Dat_P[i])
>   XX = XX + i * i
>   S = S + ALOG(Dat_P[i])
> ENDFOR
>
> ; Find parameters A1, B1
> XM = (2. + J0) / 2.
> YM = S / Float(J0)
> A1 = (XY - Float(J0)*XM*YM) / (XX-Float(J0)*XM*XM)
> B1 = YM - A1 * XM
>
>
> ;;-----
> ;; compute J1, the frequency for which straight
> ;; line extrapolation drops 20dB below noise
> ;;-----
> J1 = J0
> FOR i = J0, (N-1)/2 DO BEGIN
>   vgl = EXP(A1+Float(i)+B1)/NOISE
>   IF (vgl LT 0.01) THEN BEGIN
>     J1 = i
>     BREAK
>   ENDIF
> ENDFOR
>
>
> ;;-----
> ;; Compute the Kosarev-Pantos filter windows
> ;; Frequency-ranges:
> ;; 0 -- J0 | J0+1 -- J1 | J1+1 -- N2
> ;;-----
> LOPT = DblArr((N-1)/2 + 1)
> FOR i = 0, J0 DO BEGIN
>   LOPT[i] = Dat_P[i] / (Dat_P[i] + NOISE)
> ENDFOR
> FOR i = J0, J1 DO BEGIN
>   LOPT[i] = EXP(A1*Float(i)+B1) / (EXP(A1*Float(i)+B1) + NOISE)
> ENDFOR
> FOR i = J1+1, (N-1)/2 DO BEGIN
>   LOPT[i] = 0.
> ENDFOR

```

```

>
>
> ;;-----
> ;; Denoise the Spectrum with the filter
> ;; Calculate the first and second derivative (i.e. multplie with iW)
> ;;-----
> Smoothed_Data = FltArr(N)
> First_Diff = FltArr(N)
> Secnd_Diff = FltArr(N)
> Fltr_Spectrum = DcomplexArr(N)
>
> ; first Loop gives differentiation
> FOR diff = 0, 2 DO BEGIN
>
>   FOR j= 1, (N-1)/2 DO BEGIN
>     ; make the filter complex
>     FltrCoef = Complex(LOPT[j],0.)
>     ; differentitation in frequency domain
>
>     iW = Float(j)*OMEGA
>     iW = Complex(0., iW)^diff
>
>     ; multiply spectrum with filter coefficient
>     Fltr_Spectrum[j] = Dat_F[j] * FltrCoef * iW
>     ; copy first half of modified spectrum to last half
>     Fltr_Spectrum[N-j] = Conj(Fltr_Spectrum[j])
>   ENDFOR
>
>   ; Fltr_Spectrum[0] is the mean (?),
>   ; which is not considered above
>   Fltr_Spectrum[0] = Dat_F[0]
>
>
>   ; The derivatives of Fltr_Spectrum[0] are 0
>   IF(diff GT 0) THEN Fltr_Spectrum[0] = Complex(0.,0.)
>
>   ; Inversal fourier transform back in time domain
>   Dat_T = FFT(Fltr_Spectrum,/Inverse)
>   Dat_T[N-1] = Complex(Float(Dat_T[0]), Imaginary(Dat_T[N-1]))
>
>   IF(diff EQ 0) THEN BEGIN
>     ; This ist the smoothed time series (baseline added)
>     FOR j = 0, N-1 DO BEGIN
>       Smoothed_Data[j] = Float(Dat_T[j]) + Dat[0] + SLOPE * Float(j)
>     ENDFOR
>   ENDIF
>
>   IF(diff EQ 1) THEN BEGIN

```

```

> R4 = SLOPE / Delt
> ; The first derivative
> FOR j = 0, N-1 DO BEGIN
>   First_Diff[j] = Float(Dat_T[j]) + R4
> ENDFOR
> ENDIF
>
> IF(diff EQ 2) THEN BEGIN
>   ; The second derivative
>   FOR j = 0, N-1 DO BEGIN
>     Secnd_Diff[j] = Float(Dat_T[j])
>   ENDFOR
> ENDIF
>
> ENDFOR
>
> !P.MULTI=0
> Plot , X, Dat,Psym=4,SymSize=.5
> Oplot, X, Y, LineStyle=4
> Oplot, X, Smoothed_Data,LineStyle=0
>
> ;OPlot,X,First_Diff,,psym=1
>
>
>
> ;savgolFilter = SAVGOL(16, 16, 1, 4)
> ;OPLOT, X, CONVOL(Y, savgolFilter, /EDGE_TRUNCATE),thick=2
>
> ;;-----
> ;; Create log-log plot of power spectrum:
> ;;-----
> ;PLOT, F, ABS(Dat_F(0:N/2))^2, $
> ; YTITLE='Power Spectrum of u(k)', /YLOG, $
> ; XTITLE='Frequency in cycles / second', /XLOG, $
> ; X RANGE=[1.0,1.0/(2.0*delt)], $
> ; XSTYLE=1,LINESTYLE=1
>
> END
>

```
