Subject: Re: peak analysis
Posted by Rob.Dimeo on Thu, 04 Jul 2013 11:15:00 GMT
View Forum Message <> Reply to Message

On Tuesday, July 2, 2013 10:20:54 PM UTC-4, 83gre wrote:
> On Friday, April 22, 2005 6:03:37 PM UTC+1, Rob wrote:
>
>> Florian Meyer wrote:
>
>>> Hi,
>
>>> is there a simple way to determine peaks in a dataset (1D) without
>
>>> using CURVEFIT or any other complicated fitting method?
>
>>> Thanks for help.
>
>>> Florian
>
>>
>
>> I have a quick-and-dirty routine that identifies peak positions and
>
>> widths without fitting. It's based on a Savitzky-Golay filter. It
>
>> works well for peaks that don't overlap *much*. It's on the RSI
>
>> contributed web site if you're interested. You can find it at the
>
>> following url
>
>> http://tinyurl.com/a2jje
>
>
>
> Is there an updated link to this routine?

Sorry, no link. But here's the source (apologies for posting the code here).

```
;$Id: get_peak_pos.pro,v 1.7 2004/01/23 13:53:12 dimeo Exp $
;+
; NAME:
;   GET_PEAK_POS
;
; PURPOSE:
;
;  Estimates the approximate locations of peaks in a "peaky"
;  data set.  The return value of the function is an array of
```

```
;  estimates for the peak locations. This function works well
;  for data whose peaks do not overlap significantly.
;
;
; AUTHOR:
;
;    Robert M. Dimeo, Ph.D.
;    NIST Center for Neutron Research
;    100 Bureau Drive
;    Gaithersburg, MD 20899
;    Phone: (301) 975-8135
;    E-mail: robert.dimeo@nist.gov
;    http://www.ncnr.nist.gov/staff/dimeo
;
; CATEGORY:
;
;    Data analysis, model fitting, mathematics
;
; CALLING SEQUENCE:
;
;    PEAK_LOC = GET_PEAK_POS(   X,Y,NPEAKS,                $
;                          INDICES = indices,          $
;                          N_CROSSINGS = n_crossings,    $
;                          DEGREE = degree,            $
;                          NLEFT = nleft,              $
;                          NRIGHT = nright,            $
;                          FWHM = fwhm
;
; INPUT PARAMETERS (required)
;
;   X:        array of x values (independent variable)
;   Y:        array of y values (dependent variable) containing the peaks
;   NPEAKS:    number (integer) specifying the number of peaks to return
;
; INPUT KEYWORD PARAMETERS (optional)
;
;   DEGREE:    Degree of polynomial used in the Savitsky-Golay filter (def: 4)
;   NLEFT:     Number of points to the left of the filtered point over which
;              the filter extends (default: 5)
;   NRIGHT:    Number of points to the right of the filtered point over which
;              the filter extends (default: 5)
;
; OUTPUT KEYWORD PARAMETERS (optional)
;
;   FWHM:      Estimate of the full-width at half maximum for each peak
;   INDICES:   array of indices specifying the index into the x-array
;              describing the peak position,
;              i.e. XPEAKS = INTERPOLATE(X,INDICES)
```

```
;   N_CROSSINGS:  number of zero crossings of the first derivative of the
;                 smoothed function (using a Savitsky-Golay filter).
; COMMON BLOCKS:
;
;   NONE
;
; KNOWN LIMITATIONS:
;
;   This algorithm can get fooled when peaks are too close together.
;
; REQUIREMENTS:
;
;   IDL 5.4 or higher
;
; EXAMPLE:
;
;   See the code appended to the end of this file.
;   IDL> .COMPILE GET_PEAK_POS
;   IDL> TEST_PEAK
;
; DISCLAIMER
;
;   This software is provided as is without any warranty whatsoever.
;   Permission to use, copy, modify, and distribute modified or
;   unmodified copies is granted, provided this disclaimer
;   is included unchanged.
;
; MODIFICATION HISTORY:
;
;  Written 5/07/03 (RMD)
;
;  Removed the main loop over each data point to increase the execution
;   speed: RMD (01/05/04)
;-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; ;;;;;;;;;;;;;;;;;;;;
function GET_PEAK_POS,  x,y,npeaks,              $
                   indices = indices,        $
                   n_crossings = n_crossings, $
                   degree = degree,          $
                   nleft = nleft,            $
                   nright = nright,          $
                   fwhm = fwhm

n_crossings = 0 & xpeaks = 0 & fwhm = 0
if n_elements(nleft) eq 0 then nleft = 5
if n_elements(nright) eq 0 then nright = nleft
if n_elements(degree) eq 0 then degree = 4
nx = n_elements(x)
```

```
nright = (nright < (fix(0.5*nx)-1)) > 1
nleft = nright
; Create an array containing the the first derivative of the
; smoothed data using a Savitsky-Golay filter.
savgol_deriv_filter = savgol(nleft,nright,1,degree)
yd = convol(y,savgol_deriv_filter,/edge_truncate)
ny = n_elements(yd)
;value_sign = 2*(yd gt (machar()).eps) - 1
value_sign = 2*(yd gt 0d) - 1
indices = 0
; Determine the number of zero crossings
diff_sign = value_sign[1:ny-1]-value_sign[0:ny-2]
wh_cross = where((diff_sign eq 2) or (diff_sign eq -2),n_crossings)

indices = 0.5*(2*wh_cross-1)

no_width = 0
if n_crossings gt 0 then begin
; Ok, now which ones of these are peaks?
   ymax = interpolate(y,indices)
   ymin = min(ymax)
   for i = 0,npeaks-1 do begin
     this_max = max(ymax,max_index)
     if i eq 0 then best_index = indices[max_index] else $
      best_index = [best_index,indices[max_index]]
     ymax[max_index] = ymin
   endfor
   indices = best_index
   xpeaks = interpolate(x,indices)
; Now calculate the FWHM of each peak
   if arg_present(fwhm) then begin
     for i = 0,npeaks-1 do begin
      full_height = y[fix(indices[i])]
      half_height = 0.5*full_height
      ; Descend down the peak until you get lower than the half height
      elevation = full_height
      increment = 0
      while elevation gt half_height do begin
         ; go down the right side of the peak
         elevation = y[fix(indices[i])+increment]
         increment = increment+1
         no_width = 0
         if (fix(indices[i])+increment) gt (ny-1) then begin
           no_width = 1
           goto, no_width_found
         endif
      endwhile
      no_width_found:
```

```
      if no_width then width = 2.0*(x[ny-1]-xpeaks[i]) else $
         width = 2.0*(x[fix(indices[i])+increment-1]-xpeaks[i])
      if i eq 0 then fwhm = width else fwhm = [fwhm,width]
     endfor
   endif
endif
return,xpeaks
end
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; ;;;;;;;;;;;;;;;;;;;;
;;;;;BEGIN  EXAMPLE/DEMO;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; ;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; ;;;;;;;;;;;;;;;;;;;;
function fp_gaussian,x,area,center,fwhm
sig = fwhm/2.354
y = (area/sqrt(2.0*!dpi*sig^2))*exp(-0.5*((x-center)/sig)^2)
return,y
end
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; ;;;;;;;;;;;;;;;;;;;;
pro test_peak
; Synthesize some noisy "peaky" data
!except = 0
nx = 200 & xlo = -10.0 & xhi = 10.0 & dx = (xhi-xlo)/(nx-1.0)
x = xlo+dx*findgen(nx)
area = 1500.0 & center = 0.0 & fwhm = 1.0 & bg = 0.1*area
et = 4.0
ygauss = fp_gaussian(x,area,center,fwhm) + bg + $
         fp_gaussian(x,0.15*area,et,fwhm) + $
         fp_gaussian(x,0.15*area,-et,fwhm)
ydata = fltarr(nx)
for i = 0,nx-1 do begin
   ydata[i] = randomn(s,1,poisson = ygauss[i])
endfor
yerr = sqrt(ydata)

f = 1.e-3
ydata = ydata*f
yerr = yerr*f

plot,x,ydata,psym = -4;,yrange = [-50.0,1.2*max(ydata+yerr)],/ysty
errplot,x,ydata-yerr,ydata+yerr,width = 0.0
print,'Press a key to find the peaks'
r = get_kbrd(1)

; Find the peaks
npeaks = 3  ; number of peaks (assumed known)
xpeaks = get_peak_pos(x,ydata-min(ydata),npeaks,fwhm = fwhm,indices = indices)
ymax = interpolate(ydata-min(ydata),indices)
print,'Peak location estimates: ',xpeaks
print,'Peak width estimates: ',fwhm
```

```
; Put lines on the data indicating where the peak estimates are located
if n_elements(xpeaks) gt 1 then begin
   for i = 0,n_elements(xpeaks)-1 do begin
     plots,[xpeaks[i],xpeaks[i]],!y.crange,/data
     lo = xpeaks[i]-0.5*fwhm[i] & hi = xpeaks[i]+0.5*fwhm[i]
     plots,[lo,hi],0.5*[ymax[i],ymax[i]]+min(ydata),/data
   endfor
endif
end
```