```
;+
; NAME:
;               DEMO
;
; AUTHORS:
;  Cherrie Kong & Mark B. Cannell.
;  Dept. Physiology, University of Auckland, New Zealand.
;
; DESCRIPTION:
;        This routine demonstrates how to use the MFDA and accessory routines to detect Ca2+ sparks in experimental datasets.
;        This demonstration is intended for line-scanned laser-scanning confocal images, with x-t oriented to x-y
;        respectively and in TIFF format. Use of other formats require the use of the appropriate read routines.
;        The orientation of images is important as the supplied model is oriented in that direction.
;        A simple normalization of the data is carried out by DEMO_NORMALIZER which simply averages all lines as
;        representative of the background. The user is strongly urged to think about what is appropriate normalization!
;        In this demonstration the data is broken into 2048-line chunks. This is not needed for the MFDA per se but simply
;        because it is likely the background will change after so many line scans. If the background changes significantly,
;        a warning is printed -please do not simply ignore this warning -something was happening during the experiment...
;        (Normalization provided by the user usually involves selecting a non-spark region to be used to create the average array
;        required and users can supply their own normalized datasets and models in the MFDA by directly using the FINDOBJECT
;        routine below.) This routine WILL NOT re-enter with a revised model  and repeat the fit
;        -it's only a demonstration of the calling procedures!
;
; CALLING SEQUENCE:
;     DEMO [,NORMDAT,MODEL,RESULTS,REFINED]
;
; INPUTS:
;     The user will be prompted for filename and model parameters.
;
; OUTPUTS:
;     All outputs are optional and allow the user to use these elements as required:
;
;                         NORMDAT:         normalized data array
;                         MODEL:           original model that the user created
;                         RESULTS: array holding results of the MFDA. Each line gives 3 values:
;                                 [0]:     p-value associated with detected location
;                                 [1]:     x-ordinate of detected location
;                                 [2]:     y-ordinate of detected location
;                         REFINED: refined model based on curve-fitting to an average of the detected events
;
; COPYRIGHT:The IDL codes and algorithms in this program are to be used for
;                         non-commerical purposes only. For commerical use, contact the authors.
;-

FORWARD_FUNCTION DEMO_READER,DEMO_CELLNONCELL,KERNELSIZE,DEMO_NORMALIZER
PRO DEMO,normdat,model,results,refined
!EXCEPT = 0      ;kill a bug in Windows implementation of IDL

;INTRODUCTION
        PRINT,''
        PRINT,':::MFDA DEMONSTRATION:::'
        PRINT,''
        PRINT,'This is a demonstration of the Matched Filter Detection Algorithm (MFDA) and is intended for '
        PRINT,'line-scanned laser-scanning confocal images (x-t) in TIFF format. x-t orientation must '
        PRINT,'correspond with x-y, respectively, as per standard definition. '
        PRINT,'Proceed? Y/N ' & proceed = STRUPCASE(GET_KBRD()) ;ask user if they wish to proceed

IF (proceed EQ 'Y') THEN BEGIN
;READ IN DATA & DISPLAY
        PRINT,'' & PRINT,':::STEP 1: READ IN DATA... '

        filename = ''    ;data file
        WHILE (filename EQ '') DO BEGIN   ;query user until appropriate filepath and name entered
                PRINT,'Please type in full TIFF filepath and name  (e.g. C:\images\data.tif) '
                READ,filename
                query = QUERY_TIFF(filename)      ;check file exists and is in TIFF format
                IF (query EQ 0) THEN BEGIN        ;if query unsuccessful
                        filename = ''
                        PRINT,'Bad filename. Please try again.'
                ENDIF ELSE BEGIN
                        data = READ_TIFF(filename)        ;read in TIFF
                        szda = SIZE(data);find size of DATA
                        IF (szda[0] NE 2) THEN BEGIN      ;must be 2D array
```

```
                                        PRINT,'Data is not a 2D array. Do you want to read another file?'  & proceed = STRUPCASE(GET_KBRD())
                                        IF (proceed EQ 'Y') THEN filename = '' ELSE RETURN
                                ENDIF ELSE BEGIN
                                IF (szda[4] LT 10000) THEN BEGIN  ;DATA must have > 10000 pixels
                                        PRINT,'Dataset is too small. Do you want to try again?'  & proceed = STRUPCASE(GET_KBRD())
                                        IF (proceed EQ 'Y') THEN filename = '' ELSE RETURN
                                ENDIF
                                ENDELSE
                        ENDELSE
                ENDWHILE

                IF (szda[1] LT 256) THEN zoom = 256/szda[1] ELSE zoom = 1 ;zoom up to show pixels
                SLIDE_IMAGE,BYTSCL(REBIN(data,szda[1]*zoom,szda[2]*zoom,/SAMPLE)),XVISIBLE=256,YVISIBLE=256,TITLE='Data' ;display DATA in window

;ASK FOR PARAMETERS AND CREATE MODEL
        PRINT,'' & PRINT,':::STEP 2: CREATE A MODEL... '

        peak = 1. & PRINT,'What is its peak change in fluorescence (dF; e.g. 1)? ' & READ,peak
        bkgd = 1. & PRINT,'What is its basal fluorescence (F0; e.g. 1.0)? ' & READ,bkgd
        fwhm = 5. &      PRINT,'What is its FWHM (e.g. 5 px)? ' & READ,fwhm
        rise = 1.8 & PRINT,'What is its rise time (e.g. 1.8 px)? ' & READ,rise
        decay = 28. & PRINT,'What is its decay time (e.g. 28 px)? ' & READ,decay

        model = FLTARR(256,256)   ;create MODEL array
        params = [peak,fwhm,rise,decay,128,128,bkgd]        ;model parameters with centered spark
        MAKESPARK_FUNC,model,params        ;create MODEL spark
        WINDOW,xsize=256*zoom,ysize=256*zoom,title='Model' ;open new window

        TVSCL,REBIN(model,256*zoom,256*zoom,/SAMPLE)        ;display MODEL in new window with same zoom as DATA

;SEARCH FOR MODEL OBJECT IN DATASET USING THE MFDA
        PRINT,'' & PRINT,':::STEP 3: EMPLOY MATCHED FILTER DETECTION ALGORITHM (sigp = 0.001)...'
        sigp = 0.001      ;assign value of SIGP
                          ;if SZDA[2]>1024 lines, then DATA will be segmented for analysis
        settings = KERNELSIZE(data,model) ;find overlap region of segments based on border region required for FINDOBJECT
        y_bord = settings[1] + settings[3];overlap region stored in Y_BORD
        normdat = FLTARR(szda[1],szda[2]) ;empty array to store normalized segments in DATA coordinates (for viewing purposes)

        nseg = (szda[2] / 1024) > 1
        print,nseg
        linestodo = szda[2] / nseg

FOR segment=0,nseg-1 DO BEGIN
        endl = (segment+1)*linestodo
        IF (segment GT 0) THEN start = segment * linestodo - 2 * y_bord ELSE start = 0
        datn = data[*,start:endl-1]
        datn = DEMO_NORMALIZER(datn)       ;normalize segment
        szdatn = SIZE(datn)                ;size of segment
        normdat[*,start:endl-1] = datn     ;fill NORMDAT

        ;MFDA DETECTION
        FINDOBJECT,datn,model,sigp,resultsn,detectn,settings ;MFDA
        IF (segment EQ 0) THEN BEGIN       ;create/fill detect and results arrays for first segment
                print,detectn
                detected = detectn
                results = resultsn
        ENDIF ELSE BEGIN ;for second plus segments
                resultsn[2,*] += start             ;adjust detected y-ordinates to full data grid from segment grid
                results = [[results],[resultsn]]
                detected += detectn
        ENDELSE

ENDFOR

;PRINT RESULTS
        PRINT,STRING(13B),'Total number of detected events: ',detected
        PRINT,'Print full results? Y/N ' & proceed = STRUPCASE(GET_KBRD())
        IF (proceed EQ 'Y') THEN PRINT,' P-value            |  X-ordinate   |  Y-ordinate ',STRING(13B),results
        PRINT,'Show detected locations on image? Y/N ' & proceed = STRUPCASE(GET_KBRD())
        IF (proceed EQ 'Y') THEN BEGIN
                vrtbd = settings[3]-1 & hrtbd = settings[2]-1 ;create array that shows border region of no detection
                fdata = normdat                                       ;copy of normdat
                fdata[hrtbd-1:hrtbd,vrtbd:szda[2]-vrtbd] = 0 ;left border
```

```
                fdata[szda[1]-hrtbd-1:szda[1]-hrtbd,vrtbd:szda[2]-vrtbd] = 0 ;right border
                fdata[hrtbd:szda[1]-hrtbd,vrtbd-1:vrtbd] = 0 ;top border
                fdata[hrtbd:szda[1]-hrtbd,szda[2]-vrtbd-1:szda[2]-vrtbd] = 0 ;bottom border
                SLIDE_IMAGE,BYTSCL(fdata),SLIDE_WINDOW=resim,XVISIBLE=256,YVISIBLE=256,TITLE='MFDA: Detected Locations'
                WSET,resim       ;display NORMDAT with detected events
                FOR i=0,detected-1 DO ARROW,results[1,i],results[2,i]-10,results[1,i],results[2,i],THICK=2
        ENDIF

;ANALYZE RESULTS BY FITTING AVERAGE
IF (detected GT 0) THEN BEGIN              ;only give this option if there are detected events!
        PRINT,'' &       PRINT,':::(Optional) Step 4: parameter fitting of the average detected event...'
        PRINT,'Fit detected events by least-squared method to normalized data? Y/N ' & proceed = STRUPCASE(GET_KBRD())

IF (proceed EQ 'Y') THEN BEGIN
        initp = [params[0],params[1],params[3]]    ;initial parameters for fit based on parameters given for the search object (model)
        ANALYZESPARK,normdat,results,detected,initp,parmfit,refined,chi ;least-squares fit routine
        IF (chi EQ 0) THEN BEGIN  ;if fit was successful then print parameters
                PRINT,'Least-squares fit was successful. The fitted parameters are: '
                PRINT,'dF: ',parmfit[0]
                PRINT,'F0 (basal fluorescence): ',parmfit[6]
                PRINT,'FWHM: ',parmfit[1],' px'
                PRINT,'rise time constant: ',parmfit[2],' px'
                PRINT,'decay time constant: ',parmfit[3],' px'
                WINDOW,xs=512,ys=512,title='Refined Model'
                TVSCL,refined[(szda[1]/2-256)>0:(szda[1]/2+255)<(szda[1]-1),(szda[2]/2-256)>0:(szda[2]/2+255)<(szda[2]-1)]
        ENDIF ELSE PRINT,'Least-squares fit was unsuccessful. Try changing the initial parameters.' ;if fit not successful
ENDIF
ENDIF
ENDIF ELSE PRINT,'No selected: MFDA demonstration not initiated.'
PRINT,''
PRINT,':::END OF MFDA DEMONSTRATION:::'
END



;+
; NAME:           DEMO_NORMALIZER
; PURPOSE: This routine is used by DEMO and aids the user in normalizing a line-scanned

;       laser-scanning confocal image prior        to its use in the MFDA. It subtracts the mean of the data and then
;       scales the data. For many experiments this is not the best estimate of background or way to normalize the
;       data, the user should modify this routine as appropriate.
; INPUT: data:            2D data array
; OUTPUT:         data:            normalized 2D data array
; COPYRIGHT:      The IDL6.3 codes and algorithms in this program are to be used for
;                 non-commerical purposes only. For commerical use, contact the authors.
;-

FUNCTION DEMO_NORMALIZER,data
        szda = SIZE(data)
        normdat = data
        offset = MIN(data) - 1.0                                                ;estimate offset of image
        normdat = data - FLOAT(offset)                                          ;remove offset
        avgline = REBIN(normdat,szda[1],1)                                      ;average of all lines
        avg = FLOAT(REBIN(avgline,szda[1],szda[2]))                  ;resize average array to data size for division
        normdat = normdat/FLOAT(avg)                                            ;normalize
        RETURN,normdat
END



;+
; NAME:  FINDOBJECT
; PURPOSE:        This routine computes the MFDA: it detects objects against a model object defined by
;                         the user. It returns a -log(P-value) map, P-values and coordinates
;                         of local P-value minima for detected Ca2+ spark events.
; INPUTS:         data:    2D array containing search space, usually normalized data
;                         model:   object to be detected, usually a synthetic Ca2+ spark
;                         sigp:    significant P-value level
; OUTPUTS:        results: array holding output of MFDA, with length of second dimension equal to number
;                                         of detected events (unless no sparks are detected)

;                                 [0]: pvalue
```

```
;                                   [1]: x-ordinate
;                                   [2]: y-ordinate
;                          detected:number of detected events
;                          settings:settings calculated for MFDA
;                                   [0]: shar[0]
;                                   [1]: shar[1]
;                                   [2]: area[0]
;                                   [3]: area[1]
; COPYRIGHT:The IDL6.3 codes and algorithms in this program are to be used for
;                          non-commerical purposes only. For commerical use, contact the authors.
;-

FORWARD_FUNCTION FFTCORREL,RSTOP_CALC,CORREL_FIND,STATMAP,NEXTSP,KERNELSIZE
PRO FINDOBJECT,data,model,sigp,results,detected,settings
!EXCEPT = 0      ;kill a bug in Windows implementation of IDL
ON_ERROR,1

;NON-STATIONARY BEHAVIOR CHECK & WARNING
        DEMO_NONSTATWARNING,data

;CREATE INTERMEDIATE & FINAL RESULTS ARRAYS FOR MFDA
        szda = SIZE(data)         ;dimensions of DATA
        smo = SIZE(model);dimensions of MODEL
        results = FLTARR(3)

;FFT CORRELATION & FIND RSTOP (GIVEN USER-DEFINED SIGP)
        correl = FFTCORREL(data,model)    ;initital cross-correlation to find
        rstop = RSTOP_CALC(data,model,correl)              ;calculate rstop

;FIND APPROPRIATE KERNEL SIZE
        settings = KERNELSIZE(data,model)
        shar = settings[0:1]
        area = settings[2:3]

;SPEARMAN RANK CORRELATION
        detected = 0     ;initiate detection count
WHILE (MAX(correl) GT rstop) DO BEGIN              ;begin detection
        pre = CORREL_FIND(correl,area,shar)        ;find tentative spark locations
        res = STATMAP(data,model,area,shar,sigp,pre)       ;non-parametric correlation
        IF (res[0] LE sigp) THEN BEGIN                                 ;if P-value < SIGP
                IF (detected EQ 0) THEN results = res ELSE results = [[results],[res]] ;fill results array
                detected += 1    ;increase detection count
        ENDIF            ;empty array for output in stand-alone program
        correl = NEXTSP(correl,model,pre,area)     ;remove detected event from FFT correlation array
ENDWHILE
END



;+
; NAME:           DEMO_NONSTATWARNING
; PURPOSE:        This routine is used by DEMO and warns the user of
;                          non-stationery behavior that may affect MFDA performance. The routine averages
;                          line-scan data in the x-dimension to produce the time-dependent average signal.
;                          This signal is then reversed in time and compared to itself and R computed. A cut-off
;                          of 0.2 is used to detect weak trends in the data. YMMV.
; INPUT: data     :        2D data array
; OUTPUT:         status   :        0 = non-stationary behavior not detected
;                                   1 = non-stationary behaior detected and warning is printed
; COPYRIGHT:The IDL6.3 codes and algorithms in this program are to be used for
;                          non-commerical purposes only. For commerical use, contact the authors.
;-

PRO DEMO_NONSTATWARNING,data,status
        szda = SIZE(data)
        timeavg = REFORM(REBIN(FLOAT(data),1,szda[2]))
        result = CORRELATE(timeavg,REVERSE(timeavg))
        IF (ABS(result) GT 0.2) THEN BEGIN
                PRINT,'Warning: Non-stationary behavior detected! '
                status = 1
        ENDIF ELSE status = 0
END
```

```
;+
; NAME:  FFTCORREL
; PURPOSE:       This routine correlates two datasets by FFT
; INPUTS:        imA:     data to be convolved
;                         imB:     second dataset or 'PSF' (centred and enlarged)
; KEYWORDS: AUTO:         computes auto-correlation of A
; REFERENCE:This IDL6.3 routine has been modified from CONVOLVE, originally written by
;                         Frank Varosi, NASA/GSFC 1992.
;-

FUNCTION FFTCORREL,imA,imB,AUTO=auto
        A = imA - MEAN(imA)               ;mean subtraction to relate to Pearson's
        sza = SIZE(A)                     ;dimensions of A
        sc = sza/2
        na = N_ELEMENTS(A)
        aFT = FFT(A,-1 ) ;FFT of A
IF KEYWORD_SET(auto) THEN BEGIN
        acor = SHIFT(na*REAL_PART(FFT(aFT*CONJ(aFT),1)),sc[1],sc[2]) ;autocorrelation of A
        normsiga = (na-1) * VARIANCE(A)
        acor = acor / FLOAT(normsiga)             ;normalization to relate to Pearson's
        RETURN, acor
ENDIF ELSE BEGIN
        szb = SIZE(imB)
        B = imB - MEAN(imB)       ;mean subtraction to relate to Pearson's
        psf = (sc - szb/2) > 0    ;center B in new array
        s = (szb/2 - sc) > 0              ;scaling of B
        s2 = (s + sza-1) < (szb-1)
        bFT = CONJ(A)*0
        bFT[psf[1],psf[2]] = B[s[1]:s2[1],s[2]:s2[2]]
        bFT = FFT(bft,-1,/OVERWRITE)
        cor = na * REAL_PART(FFT(aFT * CONJ(bFT),1))       ;cross-correlation
        sc = sc + (sza MOD 2)
        cor = SHIFT(cor,sc[1],sc[2])      ;return to correct position
        normsig = (na-1) * STDEV(A) * STDEV(B)
        cor = cor / FLOAT(normsig)        ;normalization to relate to Pearson's
        RETURN,cor
ENDELSE
END



;+
; NAME:          RSTOP_CALC
; PURPOSE:       This routine calculates the appropriate value of RSTOP by correlating scrambled data with
;                the model to find an upper-bound for correlation values obtained with noise of the same
;                energy as the normalized data itself.
; INPUTS:        data:    2D array holding dataset
;                model:   2D array holding model object
;                correl:  2D array holding the correlation between data and model (given by FFTCORREL)
; OUTPUTS:       rstop:   the appropriate value for RSTOP
; COPYRIGHT:     The IDL6.3 codes and algorithms in this program are to be used for
;                non-commerical purposes only. For commerical use, contact the authors.
;-

FUNCTION RSTOP_CALC,data,model,correl
        szda = SIZE(data);find size of data array
        noise_est = REFORM(data(SORT(RANDOMU(seed,szda[4]))),szda[1],szda[2])
        cor_noise = FFTCORREL(noise_est,model)     ;cross-correlate noise with model
        RETURN,MEAN(cor_noise) + 6.0 * STDEV(cor_noise)    ;six standard deviations from mean correlation
END



;+
; NAME:          KERNELSIZE
; PURPOSE:       This routine finds the appropriate kernel size to use in STATMAP
; INPUTS:        data:    2D array holding dataset
;                         model:   2D array holding model object
; OUTPUTS:       shar:    half of shift area for centre of model object
;                                 [0]: x-dimension
;                                 [1]: y-dimension
```

```
;                         area:    2-element array defining kernel size for Spearman Rank Correlation
;                                  [0]: half WINA in x-dimension
;                                  [1]: half WINA in y-dimension
; COPYRIGHT:The IDL6.3 codes and algorithms in this program are to be used for
;                         non-commerical purposes only. For commerical use, contact the authors.
;-

FUNCTION KERNELSIZE,data,model
        szda = SIZE(data);find size of data
        smo = SIZE(model);find size of model
        shar = [5,5]              ;half shift area to test for most likely spark location
        peakloc = ARRAY_INDICES(model,WHERE(model EQ MAX(model))) ;find location of peak
        halfflu = ((MAX(model)-MIN(model))/2.+MIN(model)) ;half-max fluorescence value
        fwhm = N_ELEMENTS(WHERE(model[*,peakloc[1]] GT halfflu));find FWHM
        fdhm = N_ELEMENTS(WHERE(model[peakloc[0],*] GT halfflu));find full-duration half maximum (FDHM)
        area = [1.5*fwhm,1.5*fdhm]                         ;kernel size is 3 times FWHM in x and 3 times FWHD in y
        ulimx = ROUND((szda[1] - 2 * shar[0]) / 8 - 0.5);kernel cannot be larger than one quarter data size (minus shift region)
        ulimy = ROUND((szda[2] - 2 * shar[1]) / 8 - 0.5)
        llimx = 5        ;kernel should not be smaller than that which gives enough pixels for
        llimy = 5        ;Student's t distribution approximation (Spearman Rank Correlation test)
IF (area[0] GE ulimx) THEN area[0] = ulimx - 1
IF (area[0] LE llimx) THEN area[0] = llimx
IF (area[1] GE ulimy) THEN area[1] = ulimy - 1
IF (area[1] LE llimy) THEN area[1] = llimx
        RETURN,[shar,area]
END



;+
; NAME:           CORREL_FIND
; PURPOSE:        This routine finds the first maximum cross-correlation coefficent and
;                         returns its x,y coordinates on the original data grid
; INPUTS:         correl:  2D array holding R of data cross-correlated with model
;                         area: defines kernel size for Spearman Rank Correlation
;                                  [0]: half WINA in x-dimension
;                                  [1]: half WINA in y-dimension
;                         shar: half of shift area for centre of model object
;                                  [0]: x-dimension
;                                  [1]: y-dimension
; OUTPUTS:        pre:      tentative location for detected Ca2+ spark
;                                  [0]: tentative x-ordinate
;                                  [1]: tentative y-ordinate
; COPYRIGHT:The IDL6.3 codes and algorithms in this program are to be used for
;                         non-commerical purposes only. For commerical use, contact the authors.
;-

FUNCTION CORREL_FIND,correl,area,shar
        sco = SIZE(correl)        ;dimensions of CORREL
        mask = BYTARR(sco[1],sco[2])       ;create mask to allow edge-space for STATMAP
        mask[shar[0]+area[0]:sco[1]-shar[0]-area[0]-1,shar[1]+area[1]:sco[2]-shar[1]-area[1]-1] = 1
        correl = mask * correl             ;apply mask
        RETURN,ARRAY_INDICES(correl,WHERE(correl EQ MAX(correl))) ;find first maximum R value location
END



;+
; NAME:  STATMAP
; PURPOSE:        This routine cross-correlates regions of interest with a model object
;                         using the Spearman Rho Rank method. It returns a map showing the
;                         - log ( P-value ) of the regions and the final locations of detected events.
; INPUTS:         data:    2D array containing search space, usually normalized data
;                         model:   2D array containing model object, usually a synthetic CLSM spark
;                         area:    2-element array defining kernel size for Spearman Rank Correlation
;                                  [0]: half WINA in x-dimension
;                                  [1]: half WINA in y-dimension
;                         shar:     half of shift area for centre of model object
;                                  [0]: x-dimension
;                                  [1]: y-dimension
;                         pre:      tentative location for detected Ca2+ spark
;                                  [0]: tentative x-ordinate
;                                  [1]: tentative y-ordinate
;
```

```
; OUTPUTS:       xy:              final location for detected Ca2+ spark
;                                 [0]: final x-ordinate
;                                 [1]: final y-ordinate
; COPYRIGHT:The IDL6.3 codes and algorithms in this program are to be used for
;                        non-commerical purposes only. For commerical use, contact the authors.
;-

FORWARD_FUNCTION MINP
FUNCTION STATMAP,data,model,area,shar,sigp,pre
        ssp = SIZE(model);define model kernel, ARC
        arc = model[ssp[1]/2-area[0]:ssp[1]/2+area[0]-1,ssp[2]/2-area[1]:ssp[2]/2+area[1]-1]
        ind = WHERE(arc GT MIN(model))     ;use only pixels with intensity above basal fluorescence
        arc = arc[ind]
        pmap = FLTARR(2*(area[0]+shar[0]),2*(area[1]+shar[1]),2);array to hold individual rho- and P-value maps
        pmap[*,*,1] = 1. ;default P-value in map is 1.0
FOR xim = pre[0]-shar[0],pre[0]+shar[0]-1 DO BEGIN  ;shift in X over area (defined by SHAR) around PRE
        FOR yim = pre[1]-shar[1],pre[1]+shar[1]-1 DO BEGIN ;shift in Y over area (defined by SHAR) around PRE
                aoi = data[xim-area[0]:xim+area[0]-1,yim-area[1]:yim+area[1]-1] ;define AOI in DATA
                aoi = aoi[ind]   ;use IND pixels only
        pmap[xim-pre[0]+shar[0]+area[0],yim-pre[1]+shar[1]+area[1],*] = R_CORRELATE(aoi,arc) ;Spearman Rank Correlation
        ENDFOR
ENDFOR
        RETURN, MINP(area,shar,pmap,pre)  ;find XY by locating minimum P-value
END



;+
; NAME:  MINP
; PURPOSE:       This routine returns the coordinates of the center of
;                        the detected object as determined by the Spearman Rank Rho Correlation,
;                        used in STATMAP.
; INPUTS:        area:    2-element array defining kernel size for Spearman Rank Correlation
;                                 [0]: half WINA in x-dimension
;                                 [1]: half WINA in y-dimension
;                        shar:     half of shift area for centre of model object
;                                 [0]: x-dimension
;                                 [1]: y-dimension
;                        pmap:     P-value map for individual object
;                        pre:      tentative location for detected Ca2+ spark
;                                 [0]: tentative x-ordinate
;                                 [1]: tentative y-ordinate
; OUTPUT:        res:     [pvalue,x,y], where
;                                 pvalue = minimum P-value in pmap
;                                 x = final x-ordinate
;                                 y = final y-ordinate
; COPYRIGHT:The IDL6.3 codes and algorithms in this program are to be used for
;                        non-commerical purposes only. For commerical use, contact the authors.
;-

FUNCTION MINP,area,shar,pmap,pre
        pvalue = MIN(pmap[*,*,1]) ;minimum value in PMAP[*,*,1] is the pvalue
        locxy = ARRAY_INDICES(pmap[*,*,1],WHERE(pmap[*,*,1] EQ pvalue)) ;locate pvalue
IF (N_ELEMENTS(locxy) GT 2) THEN BEGIN     ;if more than one pixel contains minimum take center
        locxy[0,0] = MEAN(locxy[0,*])
        locxy[1,0] = MEAN(locxy[1,*])
ENDIF
        locxy[0,0] = pre[0] - area[0] - shar[0] + locxy[0,0] ;convert coordinates from PMAP grid to DATA grid
        locxy[1,0] = pre[1] - area[1] - shar[1] + locxy[1,0]
        RETURN,[pvalue,locxy[*,0]]        ;return pvalue and coordinates
END



;+
; NAME:  NEXTSP
; PURPOSE:       This routine removes a found object in the initial correlation
;                        array to prepare it for subsequent object searches. Removal uses
;                        the model object auto-correlation.
; INPUTS:        correl:  initial 2D FFT correlation array
;                        model:    2D array holding model object
;                        xy:               initial location for detected Ca2+ spark
```

```
;                                   [0]: final x-ordinate
;                                   [1]: final y-ordinate
;                          area:    2-element array defining kernel size for Spearman Rank Correlation
;                                   [0]: half WINA in x-dimension
;                                   [1]: half WINA in y-dimension
; OUTPUTS:       correl:  amended 2D FFT correlation array
; COPYRIGHT:The IDL6.3 codes and algorithms in this program are to be used for
;                          non-commerical purposes only. For commerical use, contact the authors.
;-

FUNCTION NEXTSP,correl,model,xy,area
        ssp = SIZE(model);dimensions of MODEL
        sco = SIZE(correl)        ;dimensions of CORREL
        spmask = SMOOTH(FFTCORREL(model,/AUTO),2*area[0],/EDGE) > 0
;note autocorrelate model as term for subtraction and blur to account for differences in detected objects
        spmask = spmask * MAX(correl) / MAX(spmask)                 ;scale amplitude to cross-correlation
IF (sco[1] LT ssp[1]) THEN spmask = spmask[ssp[1]/2-sco[1]/2:ssp[1]/2-sco[1]/2+sco[1]-1,*] ;if x-dimension of model is > data
IF (sco[2] LT ssp[2]) THEN spmask = spmask[*,ssp[2]/2-sco[2]/2:ssp[2]/2-sco[2]/2+sco[2]-1] ;if y-dimension of model is > data
        mask = FLTARR(sco[1],sco[2])      ;create MASK array to size of CORREL
        ssp = SIZE(spmask)        ;dimensions of SPMASK
        mask[sco[1]/2-ssp[1]/2:sco[1]/2-ssp[1]/2+ssp[1]-1,sco[2]/2-ssp[2]/2:sco[2]/2+ssp[2]-ssp[2]/2-1]=spmask ;fill MASK
        mask = SHIFT(mask,xy[0]-sco[1]/2-1,xy[1]-sco[2]/2-1) ;position correctly
        RETURN,correl - mask      ;remove detected event
END



;+
; NAME:  MAKESPARK_FUNC
; PURPOSE:       This routine generates a synthetic CLSM Ca2+ spark in a 2D array
; INPUT:         arr:      2D floating point array to hold Ca2+ spark
;                p: 7-element floating point array holding initial spark parameters
;                          [0]: dF
;                          [1]: FWHM (px)
;                          [2]: tau rise (px)
;                          [3]: tau decay (px)
;                          [4]: model center (x-ordinate), =< array x-dimensions
;                          [5]: model center (y-ordinate), =< array y-dimensions
;                          [6]: F0
; OUTPUT:        arr:      2D floating point array holding Ca2+ spark
;                          start:    (optional) start time (px) of spark time course
;                          tpeak:    (optional) time to peak (px)
;                          thalf:    (optional) half-time to decay (px)
; COPYRIGHT:The IDL6.3 codes and algorithms in this program are to be used for
;                          non-commerical purposes only. For commerical use, contact the authors.
;-

PRO MAKESPARK_FUNC,arr,p,start,tpeak,thalf
!EXCEPT=0
        sz = SIZE(arr)            ;dimensions of ARR
        y = FINDGEN(sz[2]) - P[5] ;generate time axis with shift
;amplitude(t) function
        ampt = EXP(-EXP(-y/P[2])-y/P[3])  ;sigmoidal rising phase and exponential decay phase
        ampt = ampt * P[0] / EXP(P[2]*(ALOG(P[2]/P[3])-1.)/P[3]);scale amplitude
        start = WHERE(ampt GE 1E-20) & start = start[0]     ;find start of rise
        maxp = -P[2]*ALOG(P[2]/P[3]) + P[5]                         ;find location of peak
        endhalf = WHERE(ampt GE (P[0]/2.))                          ;find time of half decay
        tpeak = maxp - start                                        ;find time to peak
        thalf = endhalf[N_ELEMENTS(endhalf)-1] - maxp       ;find time of half decay
;sigma(t) function
        sigma = P[1] / 2.355      ;convert FWHM to sigma
        sigmat = ((y-start+P[5])^2+0.0001)^0.25    ;sigma increases with positive and negative square root of time
        sigmat = sigmat * sigma / sigmat[maxp]     ;scale sigma at peak
                                                    ;fluorescence(x,t) function
        FOR i =0,sz[1]-1 DO       arr[i,*] = ampt * EXP(-(i-P[4])^2/(2.*sigmat^2)) + P[6]     ;create spark
END



;+
; NAME:  MAKESPARK
; PURPOSE:       This routine utilizes the MAKESPARK_FUNC routine to generate a
;                          synthetic CLSM Ca2+ spark. This adapts MAKESPARK_FUNC for its use in
```

```
;                          CURVEFIT (a 1D non-linear least squares fitting routine within the IDL
;                          package), also providing analytical partial derivatives.
; INPUT:         x:                 1D independent variable, where
;                                  [0]: size of data array in x
;                                  [1]: size of data array in y
;                                  [2:x[0]+1]: independent variable x
;                                  [x[0]+2:*]: independent variable y
;                          p:                7-element floating point array holding initial spark parameters
;                                  [0]: dF
;                                  [1]: FWHM (px)
;                                  [2]: tau rise (px)
;                                  [3]: tau decay (px)
;                                  [4]: model center (x-ordinate)
;                                  [5]: model center (y-ordinate)
;                                  [6]: F0
; OUTPUT:        z:                 1D data array
;                          pder: (optional) 2D array of partial derivatives
; COPYRIGHT:The IDL6.3 codes and algorithms in this program are to be used for
;                          non-commerical purposes only. For commerical use, contact the authors.
;-

PRO MAKESPARK,x,p,z,pder
        nx = LONG(x[0]) & ny = LONG(x[1])                              ;data size in x- and y-dimensions
        n = nx * ny                                                            ;no. elements in dataset
        z = FLTARR(nx,ny)                                              ;create empty array for spark


MAKESPARK_FUNC,z,p,start
;call spark routine
        z = REFORM(z,n,/OVERWRITE)                                             ;2D to 1D spark
IF (N_PARAMS() GE 4) THEN BEGIN                                                ;partial derivatives
        xp = (x[2:nx+1]) # REPLICATE(1,ny)                             ;x independent variables
        yp = REPLICATE(1,nx) # (x[nx+2:*])                             ;y independent variables
        a = P[0] / EXP( (P[2]/P[3]) * (ALOG(P[2]/P[3])-1.) )
        b = EXP(-(yp - P[5])/P[2])
        c = (yp-P[5])/P[3]
        d1 = P[2] * ALOG(P[3]/P[2]) + P[5] - start
        d2 = (xp-P[4])/P[1]
        d = 0.5 * d2^2 * (2.355*(d1^2+0.0001)^0.25/((yp-start)^2+0.0001)^0.25)^2
        e = EXP(-b-c-d)
        pder[*,0] = a*e / P[0]
        pder[*,1] = a*e*2.*d/P[1]
        pder[*,2] = -a*e*(((ALOG(P[2]/P[3])-1.)/P[3]+(1/P[3]))+((yp-P[5])*b/P[2]^2+0.5*((d2*2.355)^2*d1* $
(ALOG(P[3]/P[2])-1.)/((d1^2+0.0001)^0.5*((yp-start)^2+0.0001)^0.5) )) )
        pder[*,3] = -a*e*((-P[2]*(ALOG(P[2]/P[3])-1.)/P[3]^2 - P[2]/P[3]^2)-(c/P[3]-0.5*(d2*2.355)^2*d1* $
P[2]/(P[3]*((d1^2+0.0001)^0.5*((yp-start)^2+0.0001)^0.5))))
        pder[*,4] = a*e*(xp-P[4])/((P[1]*((yp-start)^2+0.0001)^0.25/(2.355*((P[2]*ALOG(P[3]/P[2])+P[5]-start)^2+0.0001)^0.25))^2)
        pder[*,5] = a*((-b/P[2])+1/P[3]-d)*e
        pder[*,6] = 1.
ENDIF
END



;+
; NAME:          ANALYZESPARK
; PURPOSE:       This routine fits parameters of Ca2+ spark using defined model/function by calling CURVEFIT and estimating
;                initial parameters with those used for finding sparks in FINDOBJECT and with locations
;                output from FINDOBJECT.
; INPUT: data:    data array
;                          mfdares: array holding output of MFDA, with length of second dimension equal to number
;                                            of detected events (unless no sparks are detected)
;                                  [0]: pvalue
;                                  [1]: x-ordinate
;                                  [2]: y-ordinate
;                          p:                initial parameters for fit
;                                  [0]: dF
;                                  [1]: FWHM at peak (px)
;                                  [2]: decay time constant (px)
; OUTPUT:        parmfit: fitted parameters of a spark
;                                  [0]: dF
;                                  [1]: FWHM at peak (px)
;                                  [2]: rise time constant (px)
```

```
;                                   [3]: decay time constant (px)
;                                   [4]: spark x-ordinate
;                                   [5]: spark y-ordinate
;                                   [6]: F0
;                 refined: optional output of 2D refined model array with fitted parameters
;                 chi:      status of curvefit, 0=success, 1=failed as Chi-square increased without bounds, 2=failed to converge in max. iterations
; COPYRIGHT: The IDL6.3 codes and algorithms in this program are to be used for non-commerical purposes only. For commerical use, contact the authors.
;-

PRO ANALYZESPARK,data,mfdares,detected,p,parmfit,refined,chi

;ORGANISE DATA FOR CURVE-FITTING
        szda = SIZE(data)         ;DATA dimensions
        p = ABS(p)                ;ensure no negative values
        nx = (4. * FIX(P[1])) < (szda[2]/4 - 1)    ;fit includes data within 4 times expected FWHM
        ny = (4. * FIX(P[2]) + 20) < (szda[2]/4 - 1)        ;fit includes data 20 px prior to detected location to 4 times expected half decay
        n = nx * ny                                         ;size of dataset to be fitted
        savg = FLTARR(nx,ny)
        count = 0
        FOR i=0,detected-1 DO BEGIN
                temp_x = mfdares[1,i] & temp_y = mfdares[2,i]
                IF (temp_x GT nx) AND (temp_y GT ny) AND (temp_x LT (szda[1]-nx)) AND (temp_y LT (szda[2]-ny)) THEN BEGIN ;events too close to edge
                        temp = SHIFT(data,szda[1]/2-mfdares[1,i],20-mfdares[2,i]) ;centering detected objects in savg
                        IF (count EQ 0) THEN savg = temp[szda[1]/2-nx/2:szda[1]/2+nx-nx/2-1,0:ny-1] ELSE $
                         savg = [[[savg]],[[temp[szda[1]/2-nx/2:szda[1]/2+nx-nx/2-1,0:ny-1]]]] ;cut out region of interest as defined by nx and ny
                        count += 1
                ENDIF
        ENDFOR
        savg = REBIN(savg,nx,ny,1)        ;signal average all detected objects

;CURVE-FITTING
        x = FINDGEN(nx) & y = FINDGEN(ny)                            ;generate independent variables
        parm = [P[0],P[1],0.1*P[2],P[2],nx/2,20,MEAN(data)] ;initial parameter estimates
        result = CURVEFIT([nx,ny,x,y],REFORM(savg,n),weight,parm,FUNCTION_NAME="MAKESPARK",STATUS=chi) ;fitting with no weighting
        parmfit = [parm[0],parm[1],parm[2],parm[3],szda[1]/2,szda[2]/2,parm[6]]      ;fitted parameters in parmfit

IF (parm[0] LT 0) OR (parm[1] LT 0) OR (parm[2] LT 0) OR (parm[3] LT 0) OR (parm[4] LT 0) OR (parm[5] LT 0) OR (parm[6] LT 0) THEN BEGIN
        chi = 1                   ;negative numbers as unsuccessful fit
        refined = INTARR(szda[1],szda[2])
ENDIF ELSE BEGIN
;GENERATE REFINED MODEL
        refined = FLTARR(szda[1],szda[2]) ;create array for refined model
        MAKESPARK_FUNC,refined,parmfit    ;generate model and print time course characteristics
ENDELSE
END



;+
; NAME:  MAKEGAUSS2D_FUNC
; PURPOSE:       This routine generates a 2D Gaussian in a 2D array
; INPUT:         array:    2D floating point array to hold 2D Gaussian
;                         p:                 6-element floating point array holding initial 2D Gaussian parameters
;                                   [0]: dF
;                                   [1]: FWHM in x (px)
;                                   [2]: FWHM in y (px)
;                                   [3]: model center (x-ordinate), =< array x-dimensions
;                                   [4]: model center (y-ordinate), =< array y-dimensions
;                                   [5]: F0
; OUTPUT:        array:    2D floating point array holding Ca2+ spark
; COPYRIGHT:The IDL6.3 codes and algorithms in this program are to be used for
;                         non-commerical purposes only. For commerical use, contact the authors.
;-

PRO MAKEGAUSS2D_FUNC,array,p
        sz = SIZE(array)          ;find dimensions of array
        nx = LONG(sz[1])          ;x-dimension
        ny = LONG(sz[2])          ;y-dimension
        MAKEGAUSS2D,[nx,ny,FINDGEN(nx),FINDGEN(ny)],p,z     ;call 2D Gaussian function (requires 1D index)
        array = REFORM(z,nx,ny)   ;reform array to 2D
END
```

```
;+
; NAME:  MAKEGAUSS2D
; PURPOSE:       This routine utilises the MAKEGAUSS2D_FUNC routine to generate a
;                        2D Gaussian. This provides analytical partial derivatives and can be used with
;                        CURVEFIT (a 1D non-linear least squares fitting routine within the IDL
;                        package).
; INPUT:         x:              1D independent variable, where
;                                [0]: size of data array in x
;                                [1]: size of data array in y
;                                [2:x[0]+1]: independent variable x
;                                [x[0]+2:*]: independent variable y
;                        p:               6-element floating point array holding initial 2D Gaussian parameters
;                                [0]: dF
;                                [1]: FWHM in x (px)
;                                [2]: FWHM in y (px)
;                                [3]: model center (x-ordinate)
;                                [4]: model center (y-ordinate)
;                                [5]: F0
; OUTPUT:        z:              1D data array
;                        pder: optional 2D array of partial derivatives
; REFERENCE:Modified from IDL6.3 GAUSS2DFIT routine.
;-

PRO      MAKEGAUSS2D,x,p,z,pder
         nx = LONG(X[0]) & ny = LONG(X[1])                              ;x- and y-dimensions
         xp = (X[2:nx+1]-p[3]) # REPLICATE(1.0/P[1],ny)      ;expand X values
         yp = REPLICATE(1.0/P[2],nx) # (x[nx+2:*]-P[4])      ;expand Y values
         n = nx * ny                                                                                        ;total number of elements
         u = REFORM(EXP(-0.5 * (xp^2 + yp^2)),n)                       ;make exp() term 1D
         z = P[5] + P[0] * u
IF (N_PARAMS() GE 4) THEN BEGIN                                                 ;partial derivatives
         s = 0.0 & c = 1.0
         pder = FLTARR(n,N_ELEMENTS(P))
         pder[*,0] = u
         u = P[0] * u                                                           ;common term
         pder[*,1] = u * xp^2 / P[1]
         pder[*,2] = u * yp^2 / P[2]
         pder[*,3] = u * (c/P[1] * xp + s/P[2] * yp)
         pder[*,4] = u * (-s/P[1] * xp + c/P[2] * yp)
         pder[*,5] = 1.0
ENDIF
END



;+
; NAME:          ANALYZEGAUSS
; PURPOSE:       This routine fits parameters of a 2D Gaussian using defined model/function by calling CURVEFIT and estimating
;                        initial parameters with those used for finding sparks in FINDOBJECT and with locations
;                        output from FINDOBJECT.
; INPUT: data:    data array
;                        mfdares: array holding output of MFDA, with length of second dimension equal to number
;                                         of detected events (unless no sparks are detected)
;                                [0]: pvalue
;                                [1]: x-ordinate
;                                [2]: y-ordinate
;                        p:               initial parameters for fit
;                                [0]: dF
;                                [1]: FWHM in x (px)
;                                [2]: FWHM in y (px)
; OUTPUT:        parmfit: fitted parameters of a spark
;                                [0]: dF
;                                [1]: FWHM in x (px)
;                                [2]: FWHM in y (px)
;                                [3]: object x-ordinate
;                                [4]: object y-ordinate
;                                [5]: F0
;                        refined: optional output of 2D refined model array with fitted parameters
;                        chi:     status of curvefit, 0=success, 1=failed as Chi-square increased without bounds, 2=failed to converge in max. iterations
; COPYRIGHT: The IDL6.3 codes and algorithms in this program are to be used for non-commerical purposes only. For commerical use, contact the authors.
;-
```

```
PRO ANALYZEGAUSS,data,mfdares,detected,p,parmfit,refined,chi

;ORGANISE DATA FOR CURVE-FITTING
szda = SIZE(data)                                                             ;data dimensions
p = ABS(p)                                                                                      ;ensure no negative values
nx = ROUND(6. * P[1] + 0.5) < (szda[1]/4 - 1)      ;fit area includes 6 times estimated FWHMx
ny = ROUND(6. * P[2] + 0.5)       < (szda[2]/4 - 1);fit area includes 6 times estimated FWHMy
n = nx * ny                                                                                     ;total number of elements
count = 0
savg = FLTARR(nx,ny)
FOR i=0,detected-1 DO BEGIN
        temp_x = mfdares[1,i] & temp_y = mfdares[2,i]
        IF (temp_x GT nx) AND (temp_y GT ny) AND (temp_x LT (szda[1]-nx)) AND (temp_y LT (szda[2]-ny)) THEN BEGIN ;events too close to edge
                temp = SHIFT(data,szda[1]/2-mfdares[1,i],szda[2]/2-mfdares[2,i]) ;centering detected objects in savg
                IF (count EQ 0) THEN savg = temp[szda[1]/2-nx/2:szda[1]/2+nx-nx/2-1,szda[2]/2-ny/2:szda[2]/2+ny-ny/2-1] ELSE $
                 savg = [[[savg]],[[temp[szda[1]/2-nx/2:szda[1]/2+nx-nx/2-1,szda[2]/2-ny/2:szda[2]/2+ny-ny/2-1]]]]
                count += 1
        ENDIF
ENDFOR
savg = REBIN(savg,nx,ny,1)                                                    ;signal average all detected objects

;CURVE-FITTING
x = FINDGEN(nx) & y = FINDGEN(ny)                            ;generate independent variables
parm = [P[0],P[1],P[2],nx/2,ny/2,MEAN(data)]       ;initial parameter estimates
result = CURVEFIT([nx,ny,x,y],REFORM(savg,n),weight,parm,function_name="MAKEGAUSS2D",STATUS=chi) ;fitting with no weighting
parmfit = [parm[0],parm[1],parm[2],szda[1]/2,szda[2]/2,parm[5]] ;fitted parameters in parmfit

IF (parm[0] LT 0) OR (parm[1] LT 0) OR (parm[2] LT 0) OR (parm[3] LT 0) OR (parm[4] LT 0) OR (parm[5] LT 0) THEN BEGIN
        chi = 1                                                               ;negative parameters as unsuccessful fit
        refined = INTARR(szda[1],szda[2])
ENDIF ELSE BEGIN
;GENERATE REFINED MODEL
        refined = FLTARR(szda[1],szda[2])                    ;create array for refined model
        MAKEGAUSS2D_FUNC,refined,parmfit                     ;generate model and print time course characteristics
ENDELSE
END
```