High Resolution Observations and Modeling of Small-scale Solar Magnetic Elements

Final Report
NASA SR&T Contract NASW-98008

Palo Alto, California
23 March, 2001

Dr. Thomas E. Berger

Lockheed-Martin Solar and Astrophysics Laboratory
3251 Hanover St.
O/L9-41 B/252
Palo Alto, Ca., 94304-1191

1. Summary

"High Resolution Observations and Modeling of Small-scale Solar Magnetic Elements" was a research contract investigating the radiative transfer and dynamic physics of the smallest observable magnetic structures in the solar photosphere. Due to the lack of a high-resolution visible light satellite instrument for solar studies, all data were acquired using ground-based instrumentation.

The primary goal of the investigation was to understand the formation and evolution of "G-band bright points" in relation to the associated magnetic elements. G-band bright points are small (on the order of 100 km or less in diameter) bright signatures associated with magnetic flux elements in the photosphere. They are seen in the $A^2\Delta$-$X^2\Pi$ 4308 Å molecular bandhead of the CH radical in the solar spectrum and offer the highest spatial resolution and highest contrast "tracers" of small magnetic structure on the Sun.

Although G-band bright points had been previously observed in ground-based images from both the Pic du Midi and Big Bear solar observatories, it was the 1993 observations of Prof. Göran Scharmer at the Swedish Vacuum Solar Telescope (SVST) that brought the attention of many researchers to the subject. From the SVST observations, it was clear that G-band bright points presented a new tool for studying the structure and evolution of the small-scale photospheric magnetic field. However the radiative transfer physics of the bright points remained unclear, making magnetic field studies based on G-band bright points the subject of criticisms as to how the bright points relate to the formation and evolution of the underlying magnetic field.

This investigation was proposed as a follow-on to the Ph.D. thesis work of Dr. Berger which established most of the major observational characteristics of G-band bright points using the SVST as the primary instrument. Originally funded in March of 1998, the three year investigation comprised two major programs:

- An observational program whose goal is to obtain the highest possible resolution movies and spectra of G-band bright points along with simultaneous magnetogram and "continuum" movies in order to study the topology and dynamics of both magnetic elements (identified in the magnetograms) and the bright points seen in G-band. In addition, we study the chromospheric atmospheric response using Ca II K-line imaging.

- A theory/modeling program whose goal is to simulate the radiative transfer characteristics of magnetic elements in the G-band region of the solar spectrum. A full understanding of the physics behind bright point formation will lead to an accurate simulation using existing radiative transfer numerical codes such as MULTI.

Drs. Thomas Berger and Mats Löfdahl (originally of Lockheed Martin Solar and Astrophysics Lab, currently at the Swedish Solar Observatory, Stockholm) conducted the observational program. Dr. Berger was responsible for all data collection and solar physics analysis; Dr. Löfdahl was in charge of developing and implementing the Partitioned Phase Diverse Speckle (PPDS) image restoration

algorithms necessary to achieve diffraction-limited imaging at the SVST. Dr. Don Kiselman of the Swedish Solar Observatory, Stockholm was in charge of the theoretical and modeling efforts. Dr. Robert Rutten of the Sterrekundig Instituut, Utrecht, the Netherlands and Dr. Alan Title of Lockheed Martin Solar and Astrophysics Lab served as senior advisors to the project in the theory and observational programs, respectively.

The following is a brief list of milestones achieved in both the observational and theoretical programs. Details of the items are provided in the Quarterly Reports associated with this contract.

Observational Milestones:

- Observing Campaigns at the SVST:
  1. 12 May - 01 June 1998
  2. 24 Aug - 30 Aug 1998
  3. 15 May - 06 Jun 1999
  4. 09 Sep - 16 Sep 1999
  5. 16 May - 25 May 2000
- PPDS Datasets:
  1. 11-Jun-1997: 4 hours of G-band and 4386 Å continuum
  2. 12-Jun-1997: 6 hours of G-band and 4386 Å continuum
- Other Datasets:
  1. 13-May-1998: G-band, Ca II K-line, and SOUP filtergrams in coordination with TRACE, MDI, and ASP/Sac Peak
  2. 30-May-1998: G-band. Ca II K-line, and SOUP filtergrams in coordination with TRACE, and MDI.
  3. 20-May-2000: G-band. LPSP, TRACE, MDI coordinated campaign.

In terms of scientific output, the 30-May-1998 dataset proved to be the most prolific. This was the dataset that first elucidated the small scale structure and dynamics of the TRACE "moss" phenomenon. Using the SOUP filter Hα filtergrams and magnetograms in a detailed comparison with TRACE Fe IX/X 171 Å movies and Yohkoh/SXT images, we were able to identify for the first time the highly dynamic interaction of bright moss EUV elements (believed to be the conductively heated footpoints of coronal loops) with the cooler chromospheric jets that give moss its reticulated appearance. This research has led to a new avenue in solar transition region exploration and may shed significant light on the dynamics of coronal loop heating, transition region interfacing with the chromosphere, and coronal loop magnetic footpoint anchoring in the photosphere. In addition, the

dataset was used to produce a detailed analysis of the relationship between G-band bright points and the underlying magnetic field using high resolution SOUP magnetograms in concert with the G-band filtergram series.

In terms of high resolution photospheric imaging, the 11 and 12-June 1997 PPDS datasets represent the longest and highest resolution movies ever produced of the solar photosphere. Both movies consist of nearly diffraction-limited images covering an approximately 2 arcminute field-of-view with a frame cadence of 20 seconds. Final reduction of the datasets is ongoing. When complete, these datasets will serve as "ground truth" datasets for the next generation of 3-D MHD models of the solar photosphere and its magnetic field generation and evolution.

Theoretical/Modeling Milestones:

- Synthesized solar G-band spectra using the SCAN-CH line list.
- Modeled the appearance of G-band bright points using "double gaussian" profiles.
- LTE radiation profiles in the G-band calculated for 200 km flux tube models.
- Stokes vector modeling of the SOUP magnetogram response.

In summary, the observational and theoretical/modeling efforts of the program have shown that G-band bright points are a natural by-product of LTE radiative transfer in the magnetized solar photospheric atmosphere. The observational program of this contract has conclusively demonstrated that G-band bright points, properly distinguished from granular edge brightenings, occur only in magnetized regions of the solar photosphere and are co-morphous with the magnetized structures on 0.″5 spatial scales and temporal scales on the order of 10 s. Hydrostatic LTE flux tube models based on standard atmosphere models can reproduce most of the relevant contrast ranges of G-band bright points. However, work by several groups is ongoing in the areas of non-LTE radiative transfer and non-standard atmospheric modeling to determine if there are demonstrable effects. While we have answered the questions that we set out to answer at the beginning of the contract, the research has raised other questions and pointed out areas that require further analysis to fully understand the formation and dynamics of small-scale magnetic elements.

A major unexpected contribution of the contract was the funding research which led to the intial understanding of the TRACE moss phenomenon, opening up a new research avenue for the study of the solar transition region.

2. Publications

We list here the major publications generated under funding by the contract. The list is sorted in chronological order.

1. "Small-Scale Magnetic Flux in the Photosphere: G-band, Magnetogram, and Ca II K-line Comparisons"
   M. G. Löfdahl, T. E. Berger, A. Title
   Oral Presentation, AGU Spring Meeting, Boston, 1998.

2. "Diffusion Measurement Processes"
   T. E. Berger, A. M. Title
   Poster Presentation, AGU Spring Meeting, Boston, 1998.

3. "Joint TRACE and La Palma Observations"
   T. E. Berger, B. de Pontieu, G. B. Scharmer
   Invited talk, AGU Fall Meeting, San Francisco, 1998.

4. "High Resolution Imaging of the Solar Chromosphere/Corona Transition Region"
   T. E. Berger, B. De Pontieu, C. J. Schrijver, and A. M. Title
   Astrophysical Journal Letters, volume 519, pp. L97-100, 1999.

5. "Dynamics of Transition Region Moss"
   T. E. Berger, B. De Pontieu, C. J. Schrijver, and A. M. Title
   Oral Presentation 79.01, AAS Centennial Meeting, Chicago, 2-June-1999.

6. "Structure and Dynamics of Transition Region Moss"
   T. E. Berger, B. De Pontieu, C. J. Schrijver, and A. M. Title
   Poster Presentation, 8th SOHO Workshop, Plasma Dynamics and Diagnostics in the Solar Transition Region and Corona
   22–25 June 1999, Paris, France.

7. "What is Moss?"
   T. E. Berger, B. De Pontieu, L. Fletcher, T. D. Tarbell, C. J. Schrijver, A. M. Title
   Poster Presentation, 2nd TRACE Workshop, 24–26 August 1999, Monterey, California.

8. "What is Moss?"
   T. E. Berger, B. De Pontieu,L. Fletcher, T. D. Tarbell, C. J. Schrijver, A. M. Title
   Solar Physics, 190, 409, 1999.

9. "On the Nature of "Moss" Observed by TRACE"
   P. C. Martens, C. C. Kankelborg, and T. E. Berger
   Astrophysical Journal, 537, 471, 2000.

10. "Small-scale Magnetic Elements"
    T. Berger
    Talk given at the Second Solar-B Science Meeting, ISAS, Japan, Oct. 2000.

11. "On the Relation of G-band Bright Points to the Magnetic Field"
    T. E. Berger and A. M. Title, Astrophysical Journal, accepted for May 10, 2001 issue.

12. "A Five-hour Sequence of High-Resolution Solar Photospheric Images in Two Wavelengths Restored by the use of Phase Diversity"
M.G. Löfdahl, T. E. Berger, and J. H. Seldin, in preparation.

13. "The formation of G-band bright points I. Standard LTE analysis" D. Kiselman, R. Rutten, & Plez, in preparation.

14. "Spatio-Temporal Multiscaling and Turbulent Solar Flows"
J. K. Lawrence, A. C. Cadavid, A. A. Ruzmaikin, and T. E. Berger
Physical Review Letters, in revision.

In addition, the creation of a web-site detailing the Lockheed Martin ground-based solar physics effort was partially funded by the contract: www.lmsal.com/LaPalma.

3. Analysis Tools

A significant number of general image analysis tools were developed under funding by the contract. Below, we list those which are written in the IDL language and are therefore readily adaptable to the SolarSoft analysis environment used by the large majority of the solar physics community. In addition to those tools listed below, Dr. Löfdahl has developed his PPDS codes for general use. These codes are written in the ANA language.

- 2D Discrete Wavelet Transform (DWT) package.
Based on the Numerical Recipes in C algorithm, an ANSI-C code was developed and linked using the IDL DLM method.

- Multi-scale image alignment tools.
IDL code for performing a general affine transformation on images in order to facilitate rigid alignment and scaling was developed.

- Image Destretching tools.
Based on the ANSI-C codes of Louis Strous which are linked to IDL via custom DLMs, IDL code for performing image-to-image uniform grid warping as well as time series destretching was developed.

- Fourier domain filtering tools.
The contract partially funded the ongoing effort to develop efficient FFT algorithms in ANSI-C for linking into IDL via the DLM method. In addition, IDL code for performing p-mode supression in the Fourier domain was developed.

We include the relevant codes as appendices to this report.

4. Appendix: Analysis Tool Codes

4.1. IDL Discrete Wavelet Transform System Routine

```
#include <stdio.h>
#include "export.h"
#include "nr.h"
#include "nrutil.h"

#define NRANSI
#define ARRLEN(arr) (sizeof(arr)/sizeof(arr[0]))

#if DLM
IDL_VPTR dwt(int, IDL_VPTR *);
int IDL_Load(void)
{
  static IDL_SYSFUN_DEF function_addr[] = {
    { dwt, "DWT", 3, 3, 0}
  };

  return IDL_AddSystemRoutine(function_addr, IDL_TRUE, 1);
}
#endif

void pwt( float *, unsigned long, int );
void print_vptr( IDL_VPTR, char * );
typedef struct {
  int ncof,ioff,joff;
  float *cc,*cr;
} wavefilt;
wavefilt wfilt;

/*****************************************************************************/

IDL_VPTR dwt( int argc, IDL_VPTR *argv)

     /* Discrete Wavelet Transform. Required inputs:
          a[]: array to be transformed
          filter[]: array of filter coefficients (scaling function).
          isign: isign > 0 does forward transform, <0 does inverse.

        Returns ndim-dimensional discrete wavelet transform of array a.
        If isign = 1, performs forward transform; if isign = -1 performs
        inverse.

        From Numerical Recipes in C, Press et al., 2nd edition. With
```

```
      modifications to adapt to IDL: pwt is the only call possible
      (no separate coding of daub4.c or similar functions). IDL user
      supplies filter coefficients directly via filter[] - there's no
      need for pwtset. NOTE: filter should be unity offset (ie. first
      element should be 0) due to use of unity offset Numerical Recipes
      algorithm.

      Wrapper routine WAVELET.PRO checks inputs and calls this function.

*************************************************************************/
{
  IDL_LONG i1, i2, i3, k, n, ncoef, *nn, nnew, ntot=1;
  IDL_VPTR filter, rfilter, result;
  int i, scale_iteration, x_dim = 1, y_dim = 2, idim;
  float *a, *b, sig=-1.0, *wksp;
  short isign;
  UCHAR atype, ndim, rtype;
  extern wavefilt wfilt;

  static IDL_EZ_ARG arg_struct[] = {  /*parameter definitions */
    {IDL_EZ_DIM_ARRAY, IDL_TYP_B_SIMPLE, IDL_EZ_ACCESS_R, IDL_TYP_UNDEF, 0, 0},  /*input array */
    {IDL_EZ_DIM_ARRAY, IDL_TYP_B_SIMPLE, IDL_EZ_ACCESS_R, IDL_TYP_UNDEF, 0, 0},  /*filter coeffs
    {IDL_EZ_DIM_MASK(0), IDL_TYP_B_SIMPLE, IDL_EZ_ACCESS_R, IDL_TYP_UNDEF, 0, 0}, /*isign flag */
  };
  IDL_EzCallCleanup(argc, argv, arg_struct);
  IDL_EzCall(argc, argv, arg_struct);

  /*get data array */
  /*print_vptr(argv[0],"argv[0]"); */
  a = (float *) argv[0]->value.arr->data;
  nn = argv[0]->value.arr->dim;
  ndim = argv[0]->value.arr->n_dim;

  /*get temporary array for result */
  b = (float *) IDL_MakeTempArray(IDL_TYP_FLOAT,ndim,nn,IDL_BARR_INI_NOP,&result);

  a  -= 1; /*unity offsets from here on out! */
  b  -= 1;
  nn -= 1;

  /*get filter and load coefficients into external structure (replaces pwtset)*/
  /*NOTE: filter should have first element = 0 to handle unity offset. Thus
   a Daubechies 4-filter has 5 coefficients: 0,0.482...                    */
  /*print_vptr(argv[1],"argv[1]"); */
  wfilt.cc = (float *) argv[1]->value.arr->data;
  ncoef = (argv[1]->value.arr->n_elts)-1; /*handle unity offset with the -1*/
```

```
wfilt.ncof = ncoef;
wfilt.cr = (float *)
  IDL_MakeTempArray(IDL_TYP_FLOAT,1,argv[1]->value.arr->dim,IDL_BARR_INI_ZERO,&rfilter);
for (k=1;k<=ncoef;k++) {
  wfilt.cr[ncoef+1-k] = sig*wfilt.cc[k];
  sig = -sig;
}
wfilt.ioff = wfilt.joff = -(ncoef >> 1);

/*for (i=0;i<=ncoef;i++) printf("%f ",wfilt.cc[i]);
printf("\n");
for (i=0;i<=ncoef;i++) printf("%f ",wfilt.cr[i]); */

/* get isign */
isign = argv[2]->value.i;
/*printf("isign=%d\n",isign); */

/*Here starts the NR code, (almost) unmodified from original */
/* CHANGES: significant changes have been made so dwt would do wavelet transforms
 * on 2D images correctly.  The following code is commented.
 */

for (idim=1;idim<=ndim;idim++) ntot *= nn[idim];
wksp = vector(1,ntot);
for (i=1;i<=ntot;i++) b[i]=a[i];  /*copy input array to avoid overwriting */

/* The purpose of scale_iteration is that at each scale, the effective size of the
 * image is decreased by half, basically quartering each time.  In the case of
 * the forward transform, scale iteration is doubled for each iteration, while
 * in the reverse transform, scale iteration starts at a maximum and halves at
 * each iteration.
 */

/*printf("This is the beginning.\n");*/

if (isign >= 0) { /*forward wavelet transform condition */
  scale_iteration = 1;
} else { /*reverse wavelet transform condition*/
  scale_iteration = nn[x_dim]/8;
}

/*printf("hello\n initial scale: %i\n", scale_iteration);*/
while (scale_iteration > 0 && nn[x_dim]/scale_iteration >= 8 && nn[y_dim]/scale_iteration >= 8
  /*printf("scale_iteration: %i\n", scale_iteration);*/

  n = nn[x_dim]/scale_iteration;
```

```
    for (i2 = 0; i2 < ntot/scale_iteration; i2+=nn[y_dim]) {
      for (i3 = i2+1, k=1; k <= n; k++, i3 += 1) {
        wksp[k] = b[i3];
      }
      pwt(wksp,n,isign);
      for (i3 = i2+1, k=1; k <= n; k++, i3 += 1) {
        b[i3]=wksp[k];   /*copy back from workspace */
      }
    }
    /*printf("part 1\n");*/
    n = nn[y_dim]/scale_iteration;
    for (i1 = 1; i1 <= nn[y_dim]/scale_iteration; i1 += 1) {
      for (i3 = i1, k=1; k <= n; k++, i3 += nn[x_dim]) {
        wksp[k] = b[i3];
      }
      pwt(wksp,n,isign);
      for (i3=i1, k=1; k <= n; k++, i3 += nn[x_dim]) {
        b[i3]=wksp[k];   /*copy back from workspace */
      }
    }
    /*printf("part 2\n");*/
    if (isign >= 0) { /*forward wavelet transform condition */
      scale_iteration *= 2;
    } else { /*reverse wavelet transform condition*/
      scale_iteration /= 2;
    }
    /*printf("scale_iteration at the end is : %i\n", scale_iteration);*/
  }
  /*printf("good\n");*/
  free_vector(wksp,1,ntot);
  IDL_EzCallCleanup(argc, argv, arg_struct);
  /*printf("even better\n");*/
  return result;
}

/*************************************************************/

void pwt(float a[], unsigned long n, int isign)

/* Partial wavelet transform. Applies an arbitrary wavelet
   filter to data vector a[1...n] passed in a[] for isign=1;
   for isign=-1, applies inverse transform.
   From Numerical Recipes in C, 2nd edition.

*************************************************************/
{
```

```
 float ai,ai1,*wksp;
 unsigned long i,ii,j,jf,jr,k,n1,ni,nj,nh,nmod;
 extern wavefilt wfilt;

 if (n < 4) return;
 wksp=vector(1,n);
 nmod=wfilt.ncof*n;  /* A positive constant equal to zero mod n */
 n1=n-1;             /* Mask of all bits, since n is a power of 2 */
 nh=n >> 1;
 for (j=1;j<=n;j++) wksp[j]=0.0;
 if (isign >= 0) {   /* Apply filter  */
   for (ii=1,i=1;i<=n;i+=2,ii++) {
     ni=i+nmod+wfilt.ioff;     /*Pointer to be incremented and wrapped around */
     nj=i+nmod+wfilt.joff;
     for (k=1;k<=wfilt.ncof;k++) {
       jf=n1 & (ni+k);        /*we use bit-wise AND to wrap around the pointers */
       jr=n1 & (nj+k);
       wksp[ii] += wfilt.cc[k]*a[jf+1];
       wksp[ii+nh] += wfilt.cr[k]*a[jr+1];
     }
   }
 } else {            /* Apply transpose filter */
   for (ii=1,i=1;i<=n;i+=2,ii++) {
     ai=a[ii];
     ai1=a[ii+nh];
     ni=i+nmod+wfilt.ioff;
     nj=i+nmod+wfilt.joff;
     for (k=1;k<=wfilt.ncof;k++) {
       jf=(n1 & (ni+k))+1;
       jr=(n1 & (nj+k))+1;
       wksp[jf] += wfilt.cc[k]*ai;
       wksp[jr] += wfilt.cr[k]*ai1;
     }
    }
  }
  for (j=1;j<=n;j++) a[j]=wksp[j];    /*copy the results back from workspace */
  free_vector(wksp,1,n);
}
#undef NRANSI


;=============================================================

FUNCTION wavelet, array, filter, isign, OVERWRITE=overwrite

;Wrapper function for wavelet transform implemented in C.
```

```
;Applies the wavelet filter FILTER, specified by a string
; corresponding to values in WT_FILTER.PRO, to the N-dimensional
; array passed in ARRAY.

; DWT is composed of c-code routines wtn and pwt adapted,
;  and modified heavily, from Numerical Recipes 2nd edition.
;===============================================================

ON_ERROR,2

if N_PARAMS() lt 3 then begin
  MESSAGE,'Wavelet transform requires three arguments: array, filter string, and isign'
  RETURN,-1
end

;Check ARRAY:
sz  = SIZE(array)
badnews = 'All dimensions of ARRAY must be powers of 2'
for i=1,sz(0) do $
if not ispow2(sz(i)) then  begin
    MESSAGE,badnews
    RETURN,-1
end
if sz(3) ne 4 then array = FLOAT(array)  ;This step is crucial since the C-code uses
; sizeof(float) offsets to do the transform steps.

;Check filter: Numerical Recipes algorithm needs unity offset vectors:
filter = STRUPCASE(filter)
coeff=wt_filter(filter)
if coeff(0) ne -1 then filt = [0.,coeff] else RETURN,coeff
nn = [sz(1)]
for i=2,sz(0) do nn = [nn,sz(i)]
nmin = MIN(nn)
if N_ELEMENTS(filt)-1 gt nmin then begin
  MESSAGE,'Filter too long for data array in at least one dimension'
  RETURN,-1
end
;help,filt
;for i=0,N_ELEMENTS(filt)-1 do print,filt(i)

;Check isign
case 1 of
  isign gt 0: isign = 1
  isign lt 0: isign = -1
else: begin
```

```
  MESSAGE,'specify ISIGN > 0 for forward transform, ISIGN < 0 for inverse'
  RETURN,-1
end
end

;All okay - do transform in C:
result = dwt(array,FLOAT(filt),isign)  ;THE FLOAT(filt) IS CRUCIAL!!

if KEYWORD_SET(overwrite) then begin
  array = result
  RETURN,1
end else RETURN,result

END


;=============================================================================
FUNCTION wt_filter, filter


;Look-up table cased on string FILTER. Returns the
; wavelet filter coefficients in a double array COEFFS.

;Coefficients gotten from Mathematica Wavelet Explorer (varying precision ones)
; and from Numerical Recipes (const. precision ones).
;=============================================================================

filter = STRUPCASE(filter)
CASE filter of

    'HAAR': coeff = REPLICATE(1./SQRT(2.0D),2)

    'DAUB4': coeff = FLOAT([0.4829629131445341,$
                            0.8365163037378079,$
                            0.2241438680420134,$
                           -0.1294095225512604])

    'DAUB6': coeff = FLOAT([0.3326705529500828,$
                            0.8068915093110929,$
                            0.4598775021184916,$
                           -0.1350110200102547,$
                           -0.08544127388202671,$
                            0.03522629188570956])

    'DAUB8': coeff = FLOAT([0.2303778133088966,$
                            0.7148465705529158,$
```

```
                          0.6308807679298591,$
                         -0.02798376941686012,$
                         -0.1870348117190932,$
                          0.03084138183556081,$
                          0.03288301166688522,$
                         -0.01059740178506904])

    'DAUB10': coeff = FLOAT([0.1601023979741925,$
                             0.6038292697971881,$
                             0.7243085284377716,$
                             0.1384281459013218,$
                            -0.2422948870663802,$
                            -0.03224486958463779,$
                             0.07757149384004565,$
                            -0.006241490212798174,$
                            -0.01258075199908195,$
                             0.003335725285473758])

    'DAUB12': coeff = FLOAT([0.111540743350,$
                             0.494623890398,$
                             0.751133908021,$
                             0.315250351709,$
                            -0.226264693965,$
                            -0.129766867567,$
                             0.097501605587,$
                             0.027522865530,$
                            -0.031582039318,$
                             0.000553842201,$
                             0.004777257511,$
                            -0.001077301085])

    'DAUB20': coeff = FLOAT([0.026670057901,$
                             0.188176800078,$
                             0.527201188932,$
                             0.688459039454,$
                             0.281172343661,$
                            -0.249846424327,$
                            -0.195946274377,$
                             0.127369340336,$
                             0.093057364604,$
                            -0.071394147166,$
                            -0.029457536822,$
                             0.033212674059,$
                             0.003606553567,$
                            -0.010733175483,$
                             0.001395351747,$
```

```
                          0.001992405295,$
                         -0.000685856695,$
                         -0.000116466855,$
                          0.000093588670,$
                         -0.000013264203])

'MEYER0': coeff = FLOAT([0.003691882686403731,$
                          0.004541394739570804,$
                         -0.02000702924793551,$
                          0.0225516658829227,$
                          0.04050242501133709,$
                         -0.1000351462396785,$
                         -0.057691904077159,$
                          0.4316322262631767,$
                          0.771509959510067,$
                          0.4316322262631767,$
                         -0.057691904077159,$
                         -0.1000351462396785,$
                          0.04050242501133709,$
                          0.0225516658829227,$
                         -0.02000702924793551,$
                          0.004541394739570804,$
                          0.003691882686403731])

'MEYER1': coeff = FLOAT([0.01260563175115622,$
                         -0.01472281058330518,$
                         -0.02431045382096553,$
                          0.04603894635662135,$
                          0.03661850739611902,$
                         -0.1194679203378185,$
                         -0.04603454889385984,$
                          0.4391769769258334,$
                          0.7566719914327569,$
                          0.4391769769258334,$
                         -0.04603454889385984,$
                         -0.1194679203378185,$
                          0.03661850739611902,$
                          0.04603894635662135,$
                         -0.02431045382096553,$
                         -0.01472281058330518,$
                          0.01260563175115622])

else: begin
PRINT,'Unsupported filter type in WT_FILTER.PRO'
PRINT,'Supported filter types:'
PRINT,'   Haar'
```

```
    PRINT,'   Daub4, Daub6, Daub8, Daub10, Daub12, Daub20'
    PRINT,'   Meyer0, Meyer1'
    coeff = -1
    end

END

RETURN, coeff
END
```

4.2. IDL Image Alignment Tools

```
;=================================================

FUNCTION AFFINE, image, mx, my, sx, theta, xc, yc,$
                 INTERP=interp, CUBIC=cubic, MISSING=missing

;=================================================
;+
; NAME:
;       AFFINE
;
; PURPOSE:
;       Apply the affine transformation given by the input parameters
;       to IMAGE.
;
; CATEGORY:
;       Z3 - Image processing, geometric transforms, image registration.
;
; CALLING SEQUENCE:
;       transformed_image = AFFINE(image,mx,my,sx,theta,xc,yc)
;
; INPUTS:
;       IMAGE:  The image to be transformed. Must be 2-D.
;       MX, MY: Magnification factors in x and y directions.
;       SX:     Horizontal shear term.
;       THETA:  Rotation angle in DEGREES. THETA > 0 => counterclockwise rotation.
;       XC, YC: Center of rotation if THETA ne 0 else translation.
;
; KEYWORDS:
;       INTERP: Set this keyword for bilinear interpolation.  If this keyword
;               is set to 0 or omitted, nearest neighbor sampling is used.
```

```
;              Note that setting this keyword is the same as using the
;              ROT_INT User Library function.  This change (and others)
;              essentially makes ROT_INT obsolete.
;
;       CUBIC: If specified and non-zero, "Cubic convolution"
;              interpolation is used.  This is a more
;              accurate, but more time-consuming, form of interpolation.
;              CUBIC has no effect when used with 3 dimensional arrays.
;              If this parameter is negative and non-zero, it specifies the
;              value of the cubic interpolation parameter as described
;              in the INTERPOLATE function.  Valid ranges are -1 <= Cubic < 0.
;              Positive non-zero values of CUBIC (e.g. specifying /CUBIC)
;              produce the default value of the interpolation parameter
;              which is -1.0.
;
;     MISSING: The data value to substitute for pixels in the output image
;              that map outside the input image.
;
; OUTPUTS:
;       NONE
;
; RETURNS:
;       TIMAGE: the affine transformation of input image IMAGE.
;
; COMMON BLOCKS:
;       None.
;
; SIDE EFFECTS:
;       None.
;
; RESTRICTIONS:
;       None.
;
; PROCEDURE:
;       Uses POLY_2D to warp the input image according to the
;       given parameters.
;
;       See:    Image Processing for Scientific Applications
;               Bernd J\"ahne
;               CRC Press, 1997, Chapter 8.
;
;       Same as ROT.PRO but includes shear term and /PIVOT is assumed.
;
; MODIFICATION HISTORY:
;       T. Berger, LMATC, 26-February-1998.
;       T. Berger, LMATC, 25-June-1998: made THETAR internal variable.
```

```
;-


ON_ERROR,2

sz = SIZE(image)
if sz[0] ne 2 then begin
    MESSAGE,'Input image must be 2-D'
    RETURN,-1
end

if N_PARAMS() eq 1 then mx = 1.D0 else mx = DOUBLE(mx)
if N_PARAMS() le 2 then my = 1.D0 else my = DOUBLE(my)
if N_PARAMS() le 3 then sx = 0.D0 else sx = DOUBLE(sx)
trans=0
if N_PARAMS() le 4 then begin
    theta = 0.D0
    trans = 1
end else thetar = DOUBLE(theta*!DTOR)

if thetar eq 0.0 then trans=1
if N_PARAMS() le 5 then begin
    if thetar eq 0 then begin
        xc = 0
        yc = 0
    end else begin
        xc = DOUBLE(sz[1]/2.)
        yc = DOUBLE(sz[2]/2.)
    end
end else xc = DOUBLE(xc)
if N_PARAMS() le 6 and thetar eq 0 then yc = 0
if N_PARAMS() le 6 and thetar ne 0 then yc = DOUBLE(sz[1]/2.) else yc = DOUBLE(yc)

mxy = mx*my

if trans then P00 = xc else  $
  P00 = ( (-my*xc + mx*sx*yc)*COS(thetar) + mxy*xc + mx*yc*SIN(thetar) )/mxy
P10 = -sx*COS(thetar)/my - SIN(thetar)/my
P01 = COS(thetar)/mx
P11 = 0.
P = [P00,P10,P01,P11]

if trans then Q00 = yc else $
  Q00 = ( mxy*yc - mx*yc*COS(thetar) + (-my*xc+mx*sx*yc)*SIN(thetar) )/mxy
Q10 = COS(thetar)/my - sx*SIN(thetar)/my
Q01 = SIN(thetar)/mx
```

```
Q11 = 0.
Q = [Q00,Q10,Q01,Q11]

i=0
if KEYWORD_SET(interp) then i=1 ;bilinear
if N_ELEMENTS(cubic) eq 0 then cubic = 0

if N_ELEMENTS(missing) eq 0 then return,POLY_2D(image,P,Q,i,CUBIC=cubic) else $
  return,POLY_2D(image,P,Q,i,CUBIC=cubic,MISSING=missing)

END

;================================================

PRO AFFINE_SOLVE, xin, xpin, $
            mx, my, sx, theta, xc, yc,$
            VERBOSE=blab

;================================================
;+
; NAME:
;       AFFINE_SOLVE
;
; PURPOSE:
;       Calculate the parameters of a general affine image
;       transformation given a set of points from two images:
;       one of the images is assumed to be the reference image,
;       the other is assumed to be an image translated, rotated,
;       scaled, and possibly sheared relative to the reference image.
;
;       The form of the general transformation is affine:
;       X = tranformed coordinates = [T+ M S R T-] X'
;       where, in homogeneous coordinates,
;
;             X  = TRANSPOSE[x, y, 1]: test image vector
;             T+ = [[1,0,x0],[0,1,y0],[0,0,1]]: translatation of (0,0) back to (x0,y0)
;             M  = [[mx,0,0],[0,my,0],[0,0,1]]:scale
;             S  = [[1,sx,0],[0,1,0],[0,0,1]]: horizontal shear
;             R  = [[cos(t),-sin(t),0],[sin(t),cos(t),0],[0,0,1]]:
;                  rotate clockwise by angle t about origin.
;             T- = [[1,0,-x0],[0,1,-y0],[0,0,1]]:center of rotation to (0,0)
;             X' = TRANSPOSE[x',y',1]: reference image vector
;
; CATEGORY:
;       Z3 - Image processing, geometric transforms, image registration.
;
```

```
; CALLING SEQUENCE:
;       AFFINE_SOLVE, xin,xrefin,sx,sy,s,theta,x0,y0
;
; INPUTS:
;       XIN:    2xN dimensional array of points taken from image1
;               which correspond to the same points in the reference image.
;               Xi = XIN(0,*)
;               Yi = XIN(1,*)
;               N is the number of points.
;
;       XPIN:   2xN dimensional array of points from the "reference image"
;               which correspond to points in the image.
;
; KEYWORDS:
;       VERBOSE: If set, print the transformation elements to the screen.
;
; OUTPUTS:
;       MX, MY: Magnification factors in x and y axes, respectively.
;
;       SX:      Horizontal shearing factor.
;
;       THETA:  Rotation angle in degrees.
;
;       XC,YC:  Center of rotation vector elements OR translation
;               vector elements.
;
; COMMON BLOCKS:
;       None.
;
; SIDE EFFECTS:
;       None.
;
; RESTRICTIONS:
;       N, the number of matched points in the transformed and reference
;       images should be large (greater than 20), should be taken from
;       widely spaced locations in the image field-of-view, and should
;       be measured to within 1-pixel for greatest accuracy.
;
;       Off-center rotation and translation require a two-stage approach
;       for image registration. i.e. in the first stage, apply the parameters
;       given by this routine to the test image. A second set of points
;       is then selected from the image and the reference image, and
;       a second run of this program should output a final translation
;       to be applied to the test image to bring it in registration with
;       the reference image. This is tested for and the user is alerted.
;
```

```
; PROCEDURE:
;       Using least squares estimation, determine the elements
;       of the general affine transformation (rotation and/or scaling
;       and/or translation and/or shearing) of an image onto a reference
;       image.
;
;       See:    Image Processing for Scientific Applications
;               Bernd J\"ahne
;               CRC Press, 1997, Chapter 8.
;
;       Use AFFINE.PRO (or ROT.PRO if no shear is found) to apply the
;       transformation to the test image after computing them with this routine.
;
; MODIFICATION HISTORY:
;       Written: T. Berger, LMATC, 24-Feb-1998.
;       Added no rotation/translation test. TEB, 2-March-98.
;-
ON_ERROR,2

x = xin
xp = xpin

n1 = (SIZE(x))[1]
n2 = (SIZE(xp))[1]
if n2 ne n1 then begin
    MESSAGE,'Number of points must be the same in both input arrays'
    RETURN
end

y = DOUBLE(x[*,1])
x = DOUBLE(x[*,0])
yp = DOUBLE(xp[*,1])
xp = DOUBLE(xp[*,0])


;Least squares solution for matrix elements: see Notebook 5, p. 132.
AT = DBLARR(2*n1,6)
zerow = [REPLICATE(0,n1)]
onerow = [REPLICATE(1,n1)]
AT[INDGEN(n1)*2,*] = [[x],[y],onerow,zerow,zerow,zerow]
AT[INDGEN(n1)*2+1,*] = [zerow,zerow,zerow,[x],[y],onerow]
A = TRANSPOSE(AT)
b = TRANSPOSE(DBLARR(2*n1))
b[INDGEN(n1)*2] = xp
b[INDGEN(n1)*2+1] = yp
xb = INVERT(AT##A,/DOUBLE)##AT##REFORM(b)
```

```
;Solve for transformation elements:
theta = -ATAN(xb[3],xb[4])
mx = xb[3]*(xb[1]*xb[3]-xb[0]*xb[4])/SIN(theta)/(xb[3]^2+xb[4]^2)
my = -xb[3]/SIN(theta)
sx = (xb[0]*xb[3]+xb[1]*xb[4])/(xb[0]*xb[4]-xb[1]*xb[3])
;Translation solution:
det = xb[0]*xb[4]-xb[1]*xb[3]
denom = (xb[0]+xb[4]) - det - 1.
xc = -(xb[2] - xb[2]*xb[4]+xb[1]*xb[5])/denom
yc = -(xb[2]*xb[3]+xb[5]-xb[0]*xb[5])/denom

trans=0
if ABS(theta) lt 5e-03 then begin  ;no rotation - use simple translation solve:
    trans=1
    xc = xb[2]
    yc = xb[5]
end

;Return the transformation FROM x TO xp:
; ie. rotate image by theta degrees counterclockwise to get to reference image
theta = theta/!dtor
if theta gt 0 then dir='clockwise' else dir='counterclockwise'

if KEYWORD_SET(blab) then begin

    stheta = STRCOMPRESS(ABS(theta),/re)
    sxc = STRCOMPRESS(xc,/re)
    syc = STRCOMPRESS(yc,/re)
    smx = STRCOMPRESS(mx,/re)
    smy = STRCOMPRESS(my,/re)
    ssx = STRCOMPRESS(sx,/re)
    PRINT,''
    PRINT,'Relative to the reference image, the test image is:'

    if not trans then begin
        PRINT,'     Rotated ',stheta,' degrees ',dir
        PRINT,'     with the center of rotation at'
        PRINT,'          xc = ',sxc
        PRINT,'          yc = ',syc
    end else begin
        PRINT,'     Rotated ',stheta,' degrees ',dir
        PRINT,'     Translated by'
        PRINT,'          dx = ',sxc
        PRINT,'          dy = ',syc
    end
```

```
    PRINT,'     Scaled by a factor of ',smx,' horizontally'
    PRINT,'     Scaled by a factor of ',smy,' vertically'
    PRINT,'     Sheared horizontally by a factor of = ',ssx
    PRINT,''
end

RETURN
END

;+
; NAME
;       IMAGE_REGISTER
; PURPOSE:
;       Widget-based program to determine the geometric registration
;       between two images of the same scene. Uses affine transformations
;       to calculate relative scaling, rotation, displacement, and shear.
;
; CALLING SEQUENCE:
;       image_register,im1,im2,x,xp
; INPUTS:
;       im1 - image 1, defined as the "reference image"
;       im2 - image 2, image to be aligned/scaled/rotated to image 1.
;
; OUTPUT:
;       x   - vector of points [2,N] where [0,N] are the x-values of
;             reference tie points selected by the user and [1,N} are
;             the y-values.
;       xp  - vector of identical points located by the user in image2.
;
; KEYWORDS:
;       center - not implemented yet.
;
;       output - STRING filename. If set, program writes X and XP to
;                the file specified by the string filename.
;
; USEAGE:
;       Upon calling, a widget with four display areas opens. In the
;       left half of the widget the reference image (im1) is displayed
;       with a 4X zoom in the zoom window. In the right half, im2 is
;       similarly displayed. The user selects a point in im1 using
;       the mouse and the left button. The user then finds this exact
;       same point in im2 and selects it using the mouse. For a robust
;       solution, usually more than 10 points are required.
;
;       If an incorrect selection is made, use the EDIT>Undo Selection
;       menu item.
```

```
;
;       To see the points selected use the DISPLAY>selections menu
;       item.
;
;       To print the current solution in the active IDL window, use
;       the TRANSFORM>affine menu selection. Repeat this step periodically
;       during the selection process to see if the solution is converging.
;       In some cases, mis-paired points may cause the solution to
;       converge to erroneous values; this requires a restart of the
;       procedure.
;
; NOTES:
;       Uses AFFINE_SOLVE.PRO to determine the affine transformation
;
; REVISION HISTORY:
;       Written    T. Berger              20-April-1998
;-
;=============================================================
       PRO Image_Register_event, event

;The main event handler called by XManager. "event" is a
;structure returned by a the WIDGET_EVENT function within
;XManager.
;=============================================================

COMMON im_register,       $
  im1, im1_x0, im1_y0,    $     ;Image 1 and its dimensions.
  im2, im2_x0, im2_y0,    $     ;Image 2 and its dimensions.
  xyvis1, xyvis2,         $     ;Coordinates of draw windows (LL corner).
  magn1, magn2,           $     ;Current magnification of images 1 and 2.
  draw1id,draw2id,        $     ;Window IDs for display widgets.
  det1id, det2id,         $     ;Window IDs for detail displays.
  winsz,                  $     ;Size of the draw windows.
  outfile,unitnum,        $     ;Output file flag and unit number.
  mouseflag,xymouse,      $     ;Flag and position of mouse.
  x1, y1, x1flag,         $     ;Selected points in image 1 and flag.
  x2, y2, x2flag,         $     ;ditto for image 2.
  xr, xpr,                $     ;Returned arrays.
  detsz                         ;Size of detail windows

COMMON widget_ids,        $
  irbase,                 $     ;Main base ID.
  draw1, draw2,           $     ;Drawing areas.
  detail1, detail2,       $     ;Detail view areas.
  text1x, text1y,         $     ;Text output areas for image 1.
  text2x, text2y                ;ditto for image 2.
```

```
WIDGET_CONTROL, event.id, GET_UVALUE=uval

;print,'Event: ', event
;print,'uval = ', uval

case uval of

  'menu': case event.value of

     'File.Save': begin


     end
     'File.Save As': begin


     end
     'File.Exit': begin        ;close the points pairs output file if still open.
         if outfile eq 1 then CLOSE,unitnum
         WIDGET_CONTROL, event.top, /DESTROY
     end
     'Edit.Undo Selection': begin
         if x1flag eq 1 then begin
             nel = (SIZE(x1))[1]
             badx = x1[nel-1]
             bady = y1[nel-1]
             PRINT,badx,bady,': Selection removed. Reselect in Image 1.'
             x1 = x1[0:nel-2]
             y1 = y1[0:nel-2]
             x2flag = 1
         end else begin
             nel = (SIZE(x2))[1]
             badx = x2[nel-1]
             bady = y2[nel-2]
             PRINT,badx,bady,': Selection removed. Reselect in Image 2.'
             x2 = x2[0:nel-2]
             y2 = y2[0:nel-2]
             x1flag  = 1
         end
     end
     'Display.Selections.Print': begin
         npts = (SIZE(x1))[1]
         if npts gt 1 then begin
             x1t = x1[1:*]
             y1t = y1[1:*]
```

```
        x2t = x2[1:*]
        y2t = y2[1:*]
        PRINT,''
        PRINT,'Selections so far:'
        PRINT,''
        PRINT,'      IMAGE 1                    IMAGE 2'
        for i=0,npts-2 do begin
            sx1 = STRCOMPRESS(x1t[i],/re)
            sy1 = STRCOMPRESS(y1t[i],/re)
            sx2 = STRCOMPRESS(x2t[i],/re)
            sy2 = STRCOMPRESS(y2t[i],/re)
            PRINT,'( ',sx1,', ',sy1,' )','     ( ',sx2,', ',sy2,' )'
        end
        PRINT,''
    end else PRINT,'No points selected yet.'
end
'Display.Selections.Circle': begin
    npts1 = (SIZE(x1))[1]
    npts2 = (SIZE(x2))[1]
    if npts1 gt 1 and npts2 gt 1 then begin
        x1t = x1[1:*]
        y1t = y1[1:*]
        x2t = x2[1:*]
        y2t = y2[1:*]
        WSET,draw1id
        for i=0,npts1-2 do TVCIRCLE,10,x1t[i]*magn1,y1t[i]*magn1,color=200
        WSET,draw2id
        for i=0,npts2-2 do TVCIRCLE,10,x2t[i]*magn2,y2t[i]*magn2,color=200
    end else PRINT,'Not enough points selected yet.'
end
'Display.Selections.Highlight': begin
    npts1 = (SIZE(x1))[1]
    npts2 = (SIZE(x2))[1]
    if npts1 gt 1 and npts2 gt 1 then begin
        x1t = x1[1:*]
        y1t = y1[1:*]
        x2t = x2[1:*]
        y2t = y2[1:*]
        WSET,draw1id
        for i=0,npts1-2 do PLOTS,[x1t[i]*magn1],[y1t[i]*magn1], $
          psym=4,thi=2,syms=0.2*magn1,color=200,/dev
        WSET,draw2id
        for i=0,npts2-2 do PLOTS,[x2t[i]*magn2],[y2t[i]*magn2], $
          psym=4,thi=2,syms=0.2*magn2,color=200,/dev
    end else PRINT,'Not enough points selected yet.'
end
```

```
      'Display.Color Table.Xloadct...': XLOADCT
      'Display.Color Table.XPalette...': XPALLETE
      'Transform.Affine': begin
          npts = N_ELEMENTS(x1)
          if npts gt 1 then begin
              x1t = x1[1:*]
              y1t = y1[1:*]
              x2t = x2[1:*]
              y2t = y2[1:*]
              x = [[x1t],[y1t]]
              xp = [[x2t],[y2t]]
              affine_solve,x,xp,/verb
          end else PRINT,'No points selected yet.'
      end
  end

  'menu1': case event.value of

      'Zoom.1:4': begin
          IR_redraw, draw1, 0.25
      end
      'Zoom.1:2': begin
          IR_redraw, draw1, 0.5
      end
      'Zoom.1:1': begin
          IR_redraw, draw1, 1.0
      end
      'Zoom.2:1': begin
          IR_redraw, draw1, 2.0
      end
      'Zoom.4:1': begin
          IR_redraw, draw1, 4.0
      end
      'Zoom.8:1': begin
          IR_redraw, draw1, 8.0
      end
      'Move.Image Center':begin
          xyvis1[0] = im1_x0/2*magn1-winsz/2
          xyvis1[1] = im1_y0/2*magn1-winsz/2
          WIDGET_CONTROL,draw1,SET_DRAW_VIEW=xyvis1
      end
      'Move.Mouse...':begin
          PRINT,'Image centers on next mouse down...'
          mouseflag = 1
      end
  end
```

```
'menu2': case event.value of

    'zoom.1:4': begin
        IR_redraw, draw2, 4
    end
    'Zoom.1:2': begin
        IR_redraw, draw2, 0.5
    end
    'Zoom.1:1': begin
        IR_redraw, draw2, 1.0
    end
    'Zoom.2:1': begin
        IR_redraw, draw2, 2.0
    end
    'Zoom.4:1': begin
        IR_redraw, draw2, 4.0
    end
    'Zoom.8:1': begin
        IR_redraw, draw2, 8.0
    end
    'Move.Image Center':begin
        xyvis2[0] = im2_x0/2*magn2-winsz/2
        xyvis2[1] = im2_y0/2*magn2-winsz/2
        WIDGET_CONTROL,draw2,SET_DRAW_VIEW=xyvis2
    end
    'Move.Mouse...':begin
        PRINT,'Image centers on next mouse down...'
        mouseflag = 1
    end
end

'draw1': begin
    xm = FLOAT(event.x)/magn1
    ym = FLOAT(event.y)/magn1
    case event.type of

        0:begin ;only register point if previous pair is done and move.mouse is inactive.
            if x2flag eq 1 and mouseflag eq 0 then begin
                x2flag = 0
                PRINT,'Image 1: ',xm,ym
                x1 = [x1,xm]
                y1 = [y1,ym]
                x1flag = 1
            end else if mouseflag eq 0 then $
              PRINT,'Please select the previous point in Image 2.'
```

```
          end

          2: begin
              IR_detail,detail1, xm, ym
              strx = STRCOMPRESS(xm,/re)
              stry = STRCOMPRESS(ym,/re)
              WIDGET_CONTROL,text1x,SET_VALUE=strx
              WIDGET_CONTROL,text1y,SET_VALUE=stry
          end
          else:

      end
  end

  'draw2': begin
      xm = FLOAT(event.x)/magn2
      ym = FLOAT(event.y)/magn2
      case event.type of

          0:begin
            if x1flag eq 1 and mouseflag eq 0 then begin
                x1flag = 0
                PRINT,'Image 2: ',xm,ym
                PRINT,''
                x2 = [x2,xm]
                y2 = [y2,ym]
                x2flag = 1
            end else if mouseflag eq 0 then  $
              PRINT,'Please select a new point in Image 1.'
        end

        2: begin
            IR_detail,detail2,xm,ym
            strx = STRCOMPRESS(xm,/re)
            stry = STRCOMPRESS(ym,/re)
            WIDGET_CONTROL,text2x,SET_VALUE=strx
            WIDGET_CONTROL,text2y,SET_VALUE=stry
        end

        else:
      end
  end


end ;case uval
```

```
RETURN
END
;=============================================================
        PRO IR_Redraw, wid, mag, XCENTER=xcen, YCENTER=ycen

; Redispalys image after changes to MAG or ORIGIN
;=============================================================

COMMON im_register,       $
  im1, im1_x0, im1_y0,    $     ;Image 1 and its dimensions.
  im2, im2_x0, im2_y0,    $     ;Image 2 and its dimensions.
  xyvis1, xyvis2,         $     ;Coordinates of draw windows (LL corner).
  magn1, magn2,           $     ;Current magnification of images 1 and 2.
  draw1id,draw2id,        $     ;Window IDs for display widgets.
  det1id, det2id,         $     ;Window IDs for detail displays.
  winsz,                  $     ;Size of the draw windows.
  outfile,unitnum,        $     ;Output file flag and unit number.
  mouseflag,xymouse,      $     ;Flag and position of mouse.
  x1, y1, x1flag,         $     ;Selected points in image 1 and flag.
  x2, y2, x2flag,         $     ;ditto for image 2.
  xr, xpr,                $     ;Returned arrays.
  detsz                         ;Size of detail windows

COMMON widget_ids,        $
  irbase,                 $     ;Main base ID.
  draw1, draw2,           $     ;Drawing areas.
  detail1, detail2,       $     ;Detail view areas.
  text1x, text1y,         $     ;Text output areas for image 1.
  text2x, text2y                ;ditto for image 2.

if N_ELEMENTS(mag) eq 0 then mag = 1
if wid eq draw1 then begin
    wnum = draw1id
    xyvis = xyvis1
    factor = mag/magn1
    zim = im1
    xs = im1_x0
    ys = im1_y0
end else begin
    wnum = draw2id
    xyvis = xyvis2
    factor = mag/magn2
    zim = im2
    xs = im2_x0
    ys = im2_y0
```

```
end
;find center of window on current image or set request:
WIDGET_CONTROL, wid, GET_DRAW_VIEW=xyvis
if KEYWORD_SET(xcen) then xc = xcen else xc = xyvis[0]+winsz/2
if KEYWORD_SET(ycen) then yc = ycen else yc = xyvis[1]+winsz/2
;PRINT,xc,yc

;display zoomed image
WIDGET_CONTROL, wid, DRAW_XSIZE=xs*mag, DRAW_YSIZE=ys*mag
WSET, wnum
WIDGET_CONTROL,/HOURGLASS
TVSCL, confac(zim,mag)
xc = xc*factor
yc = yc*factor
xyvis = [xc-winsz/2,yc-winsz/2]
WIDGET_CONTROL, wid, SET_DRAW_VIEW=xyvis

;update common block:
if wid eq draw1 then begin
    magn1 = mag
    xyvis1 = xyvis
end else begin
    magn2 = mag
    xyvis2 = xyvis
end

RETURN
END

;=============================================================
        PRO IR_Detail, wid, xm, ym

; Dispalys magnified image in detail window.
; Constant update from mouse motion in main window.
;=============================================================

COMMON im_register,       $
  im1, im1_x0, im1_y0,    $     ;Image 1 and its dimensions.
  im2, im2_x0, im2_y0,    $     ;Image 2 and its dimensions.
  xyvis1, xyvis2,         $     ;Coordinates of draw windows (LL corner).
  magn1, magn2,           $     ;Current magnification of images 1 and 2.
  draw1id,draw2id,        $     ;Window IDs for display widgets.
  det1id, det2id,         $     ;Window IDs for detail displays.
  winsz,                  $     ;Size of the draw windows.
  outfile,unitnum,        $     ;Output file flag and unit number.
  mouseflag,xymouse,      $     ;Flag and position of mouse.
```

```
  x1, y1, x1flag,        $     ;Selected points in image 1 and flag.
  x2, y2, x2flag,        $     ;ditto for image 2.
  xr, xpr,               $     ;Returned arrays.
  detsz                        ;Size of detail windows

COMMON widget_ids,       $
  irbase,                $     ;Main base ID.
  draw1, draw2,          $     ;Drawing areas.
  detail1, detail2,      $     ;Detail view areas.
  text1x, text1y,        $     ;Text output areas for image 1.
  text2x, text2y               ;ditto for image 2.

;Constants:
d2 = detsz/4

xpad = 0
ypad = 0

CASE wid of

    detail1: begin
        winid = det1id
        d2 = d2/magn1
        d = 2*d2
        subim = BYTARR(d,d)
        xlo = (xm-d2+1)
        xhi = (xm+d2)
        if xlo lt 0 then xpad = d-xhi-1
        ylo = (ym-d2+1)
        yhi = (ym+d2)
        if ylo lt 0 then ypad = d-yhi-1
        subim(xpad,ypad) = BYTSCL( im1[(xlo>0):(xhi<(im1_x0-1)), (ylo>0):(yhi<(im1_y0-1))] )
    end

    detail2: begin
        winid = det2id
        d2 = d2/magn2
        d = 2*d2
        subim = BYTARR(d,d)
        xlo = (xm-d2+1)
        xhi = (xm+d2)
        if xlo lt 0 then xpad = d-xhi-1
        ylo = (ym-d2+1)
        yhi = (ym+d2)
        if ylo lt 0 then ypad = d-yhi-1
        subim(xpad,ypad) = BYTSCL( im2[(xlo>0):(xhi<(im2_x0-1)), (ylo>0):(yhi<(im2_y0-1))] )
```

```
    end
end

WSET,winid
TVSCL,CONGRID(subim,detsz,detsz)
TVCIRCLE,20,detsz/2,detsz/2,color=200
PLOTS,[0,detsz-1],[detsz/2,detsz/2],color=200,/device
PLOTS,[detsz/2,detsz/2],[0,detsz-1],color=200,/device

RETURN
END

;=============================================================================+
        PRO Image_Register, image1, image2, xret, xpret, $
                            CENTER=xycen, OUTPUT=output

; MAIN PROGRAM: Creates the main window widget and registers it with XManager.

;=============================================================================+

COMMON im_register,       $
  im1, im1_x0, im1_y0,    $     ;Image 1 and its dimensions.
  im2, im2_x0, im2_y0,    $     ;Image 2 and its dimensions.
  xyvis1, xyvis2,         $     ;Coordinates of draw windows (LL corner).
  magn1, magn2,           $     ;Current magnification of images 1 and 2.
  draw1id,draw2id,        $     ;Window IDs for display widgets.
  det1id, det2id,         $     ;Window IDs for detail displays.
  winsz,                  $     ;Size of the draw windows.
  outfile,unitnum,        $     ;Output file flag and unit number.
  mouseflag,xymouse,      $     ;Flag and position of mouse.
  x1, y1, x1flag,         $     ;Selected points in image 1 and flag.
  x2, y2, x2flag,         $     ;ditto for image 2.
  xr, xpr,                $     ;Returned arrays.
  detsz                         ;Size of detail windows

COMMON widget_ids,        $
  irbase,                 $     ;Main base ID.
  draw1, draw2,           $     ;Drawing areas.
  detail1, detail2,       $     ;Detail view areas.
  text1x, text1y,         $     ;Text output areas for image 1.
  text2x, text2y                ;ditto for image 2.


;ON_ERROR,2

;check for previous registration: because of common block, can only have one
```

```
;realization of XYView operating at a time.
if (Xregistered('Image_Register') ne 0) then MESSAGE,'Widget is already registered'

if N_ELEMENTS(image1) eq 0 then im1=DIST(512,512) else im1 = image1
if N_ELEMENTS(image2) eq 0 then im2=DIST(512,512) else im2 = image2
sz1 = SIZE(im1)
sz2 = SIZE(im2)
if sz1[0] ne 2 or sz2[0] ne 2 then MESSAGE,'One or both of the input images is not 2-D: returning

;Initialize output state:
outfile = 0
if KEYWORD_SET(output) then begin
    outfile = 1
    OPENW,unitnum,STRING(output),/GET_LUN
end

;Initialize the selected points array (1-D coordinates):
x1 = [-1.]
x2 = x1
y1 = [-1.]
y2 = y1
x1flag=0
x2flag=1
mouseflag=0

;Widget creation:
;1. Main base:
irbase = WIDGET_BASE( TITLE='Image Registration', MBAR=bar )
pdstruct = {flags:0,name:''}
desc = REPLICATE(pdstruct,17)
desc.flags=[1,0,0,2, 1,2, 1,1,0,0,2,1,0,2,2, 1,2]
desc.name=[ 'File', $
                'Save', 'Save As', 'Exit',$
            'Edit', $
                'Undo Selection',$
            'Display', $
                'Selections', $
                    'Print','Circle','Highlight',$
                'Color Table', $
                     'Xloadct...', 'XPalette...', '', $
            'Transform',$
               'Affine']

mainmenu = CW_PDMENU( bar, desc, /RETURN_FULL_NAME, UVALUE='menu', /MBAR )
```

```
;2. Display areas: main windows are 480x480 pixels, detail windows are 256x256
minsz = 380
winsz = 480
detsz = 256

im1_x0 = sz1[1]
im1_y0 = sz1[2]
magn1 = 1
im1s = MIN([im1_x0,im1_y0],xymin1)

im2_x0 = sz2[1]
im2_y0 = sz2[2]
magn2 = 1
im2s = MIN([im2_x0,im2_y0],xymin2)
imin = MIN(im1s,im2s)
if imin lt minsz then winsz = minsz else minsz = winsz

drawbase = WIDGET_BASE( irbase, GROUP_LEADER=irbase, /FRAME,/ROW )

desc2 = REPLICATE(pdstruct,11)
desc2.flags = [1,0,0,0,0,0,2, 1,0,0,2]
desc2.name = ['Zoom',$
                '1:4','1:2','1:1','2:1','4:1','8:1',$
            'Move',$
                'Image Center','Mouse...','Enter...']

;Image 1 windows and controls
draw1_base = WIDGET_BASE( drawbase, GROUP_LEADER=drawbase, /COLUMN)
junk = WIDGET_LABEL( draw1_base,VALUE='Reference Image' )
draw1 = WIDGET_DRAW( draw1_base, RETAIN=2, GRAPHICS_LEVEL=1, $
                  XSIZE=im1_x0, YSIZE=im1_y0, FRAME=2,$
                  /BUTTON_EVENTS, /MOTION_EVENTS, UVALUE = 'draw1', $
                  /SCROLL, X_SCROLL_SIZE=winsz, Y_SCROLL_SIZE=winsz)
low1 = WIDGET_BASE( draw1_base, /ALIGN_LEFT, GROUP_LEADER=irbase,/ROW)
con1 = WIDGET_BASE( low1, GROUP_LEADER=irbase,/COL)
menu1 = CW_PDMENU( con1, desc2, /RETURN_FULL_NAME, UVALUE='menu1' )
text1x = WIDGET_TEXT( con1, FRAME=0, SCR_XSIZE=100, UVALUE='text1x' )
text1y = WIDGET_TEXT( con1, FRAME=0, SCR_XSIZE=100, UVALUE='text1y' )
detail1 = WIDGET_DRAW( low1, RETAIN=2, GRAPHICS_LEV=1, $
                    SCR_XSIZE=256, SCR_YSIZE=256, FRAME=2, $
                    /BUTTON_EVENTS, UVALUE='det1')
;Image 2 windows and controls
draw2_base = WIDGET_BASE( drawbase, GROUP_LEADER=drawbase, /COLUMN)
junk = WIDGET_LABEL( draw2_base, VALUE='Test Image' )
draw2 = WIDGET_DRAW( draw2_base, RETAIN=2, GRAPHICS_LEVEL=1, $
                  XSIZE=im2_x0, YSIZE=im2_y0, FRAME=2, $
```

4.3. IDL Destretching Tools

```
;=============================================================================

FUNCTION destretch, im0, im1, gridnest, fwfac, grid, stretch_clip,$
  OVERLAP=overlap

;=============================================================================

;Uses ANA routines GRIDMATCH and STRETCH (available via DLMs
; at http://ana.lmsal.com/anaidl) to destretch IM1 onto IM0. Useful only
; for destretching a single pair of images. Time series require
; special grid handling and smoothing.

;GRIDNEST is an N-dimensional integer array in which
; GRIDNEST(n) = the number of subtiles to be used on the
; nth destretching iteration. The tiles have the same aspect
; ratio as the images. Thus, if the image is not square, the
; user must ensure that the grid size is sufficient in longest
; dimension.

;FWFAC: apodization gaussian FWHM = least tile width * fwfac

;STRETCH_CLIP: array: maximum displacement allowed, in pixels
; for each grid in gridnest.

;OUTPUT: IM1S = destretched IM1
;        GRID = FLTARR(2,MAX(gridnest),MAX(gridnest))
;               Note that this grid is only the final incremental
;               grid applied to the image, not the full dis-
;               placement grid needed to go from im1 to im0!

ON_ERROR,2

sz0 = SIZE(im0)
sz1 = SIZE(im1)

if sz0(0) ne 2 or sz1(0) ne 2 then begin
    MESSAGE,'Images must be 2-D'
    RETURN
end
if sz0(1) ne sz1(1) or sz0(2) ne sz1(2) then begin
    MESSAGE,'Images must be same size'
    RETURN
end
```

```
  if N_ELEMENTS(stretch_clip) eq 0 then str=REPLICATE(20,N_ELEMENTS(gridnest)) else $
    str=stretch_clip

  ;default tile overlapping = 10%
  if KEYWORD_SET(overlap) then ovr = overlap else ovr = 0.10

  im1s = im1
  for i=0,N_ELEMENTS(gridnest)-1 do begin

      ni = gridnest(i)
      ngx = sz0(1)/ni
      ngy = sz0(2)/ni

      grid_x = (INDGEN(ni)#ngx + ngx/2)#(INTARR(ni)+1)
      grid_y = (INTARR(ni)+1)#(INDGEN(ni)#ngy + ngy/2)

      wind_x = FIX(ngx+ovr#ngx)
      if wind_x mod 2 eq 0 then wind_x = wind_x + 1
      wind_y = FIX(ngy+ovr#ngy)
      if wind_y mod 2 eq 0 then wind_y = wind_y + 1

      fwhm = FIX(MIN([ngx#fwfac,ngy#fwfac]))

      grid = GRIDMATCH(im0,im1s,grid_x,grid_y,wind_x,wind_y,fwhm,str(i))
      im1s = STRETCH(im1s,grid)

  end

  RETURN,im1s
  END



  ;=====================================================================

  PRO ds_series, intmpl, i0, i1, outmpl, grids, clips, sfac, delta,$
                 MAXDISP=maxdisp, LOGFILE=logfile, SILENT=silent

  ;=====================================================================


  szg = SIZE(grids)
  ngrid = szg(1)
  maxg = grids(ngrid-1)

  ;Default: no unsharp masking
```

```
if N_ELEMENTS(sfac) eq 0 then sfac = i1-i0

;The maximum allowed displacement value in pixels:
if KEYWORD_SET(maxdisp) then maxstr = maxdisp else maxstr=20
;Shutup if asked
if KEYWORD_SET(silent) then loud = 0 else loud = 1
;Redirect output to a logfile if asked
;if KEYWORD_SET(logfile) then SPAWN,'....redirect stdout to file...'


t0 = SYSTIME(1)
if loud then PRINT,'Calculating displacement grid...'
FZREAD,refim,fns(intmpl,i0),h    ;Note the use of the RIG2 files (rigid aligned)
FCWRITE,refim,fns(outmpl,i0),h,5
delta = FLTARR(i1-i0+1,2,maxg,maxg)
for i=i0+1,i1 do begin

    t0i = SYSTIME(1)
    if loud then PRINT,'Working on image ',i
    FZREAD,im,fns(intmpl,i),h
    dq = dsgridnest(refim,im,grids,clips)
    badg = WHERE(dq gt maxstr, num)
    if num gt 0 then begin
        if loud then PRINT,'    Accumulated offsets > maxdisp: ',num
        dq(badg) = 0.0
    end
    delta(i-i0,*,*,*) = dq
    if loud then PRINT,'Mean dx, dy = ',STRTRIM(AVG(ABS(dq(0,*,*))),2),' ',$
      STRTRIM(AVG(ABS(dq(1,*,*))),2)
    if loud then PRINT,'Max displ = ',STRTRIM(MAX(ABS(dq)),2)

    refim = im
    im = 0
    if loud then begin
        PRINT,'Time for image ',STRTRIM(i,2),' = ',STRTRIM(SYSTIME(1)-t0i,2),' seconds'
        PRINT,' '
    end

end
if loud then PRINT,'Time for displacement grid calculation: ',SYSTIME(1)-t0,' seconds'
;Zero out the IEEE NaN values from GRIDMATCH:
delta(WHERE(FINITE(delta) eq 0)) = 0.0

;Linear detrend and unsharp the displacements:
delx = REFORM(delta(*,0,*,*))
dely = REFORM(delta(*,1,*,*))
```

```
dsz = SIZE(delx)
nt = dsz(1)
xvec = INDGEN(nt)
ngx = dsz(2)
ngy = dsz(3)
delta = 0

if loud then PRINT,'Detrending and unsharp masking the displacements...'
t0 = SYSTIME(1)
for j=0,ngy-1 do for i=0,ngx-1 do begin

    xq = TOTAL(delx(*,i,j),/cumu)
    cf = POLY_FIT(xvec,xq,1,yfit)
    xq = xq - yfit
    xq = xq - SMOOTH(xq,sfac,/EDGE)
    delx(0,i,j) = xq - xq(0)  ;insure good starting point.

    xq = TOTAL(dely(*,i,j),/cumu)
    cf = POLY_FIT(xvec,xq,1,yfit)
    xq = xq - yfit
    xq = xq - SMOOTH(xq,sfac,/EDGE)
    dely(0,i,j) = xq - xq(0)

end
if loud then PRINT,'Detrend and smooth time = ',SYSTIME(1)-t0,' seconds'

delta = FLTARR(nt,2,ngx,ngy)
delta(*,0,*,*) = delx
delta(*,1,*,*) = dely
delx = 0
dely = 0

;Apply the grid:
t0 = SYSTIME(1)
if loud then PRINT,'Applying processed grids to images...'
for i=i0+1,i1 do begin

    FZREAD,im,fns(intmpl,i),h
    gridi = REFORM(delta(i-i0,*,*,*))
    gstd = STDEV(gridi,gavg)
    bads = WHERE(ABS(gridi) gt gavg + 3*gstd,nbad)
    if nbad gt 0 then gridi(bads) = 0
    im = STRETCH(im,gridi)

    fileout = fns(outmpl,i)
    if loud then PRINT,'Writing ',fileout
```

```
    FCWRITE,im,fileout,h,5

end
if loud then PRINT,'Time for grid application: ',SYSTIME(1)-t0,' seconds'
if loud then begin
    PRINT,' '
    PRINT,'Done destretching the time series.'
end


RETURN
END
```

4.4. IDL FFT and P-mode Filtration Routines

```
#include <stdio.h>
#include <math.h>
#include "export.h"
#include "nr.h"
#include "nrutil.h"

#define ARGS        int, IDL_VPTR[]
#define ARRLEN(arr) (sizeof(arr)/sizeof(arr[0]))
#define SWAP(a,b)   ftemp=(a);(a)=(b);(b)=ftemp
#define NRANSI

static void IDL_fourfs( int argc, IDL_VPTR *argv);
void fourew(FILE *file[5], int *na, int *nb, int *nc, int *nd);
void print_vptr( IDL_VPTR, char * );

int IDL_Load(void)
{
  static IDL_SYSFUN_DEF2 new_pro[] = {
    { (IDL_FUN_RET) IDL_fourfs, "FOURFS", 3, 3, 0, 0}
  };

  return IDL_SysRtnAdd(new_pro, FALSE, ARRLEN(new_pro));
}

/*****************************************************************************/
static void IDL_fourfs( int argc, IDL_VPTR *argv)

/*      Singleton's algorithm for FFT of a dataset stored on disk.
```

```
        Modified slightly from Numerical Recipes in C, 2nd Edition,
        Section 12.6.

        NOTE!! Data must be COMPLEX valued. For each read, the
        routine assumes it's reading a (real,imag) value pair
        stored in native floating point in the disk files. Thus
        for a (nx=128,ny=128,nt=256) image cube, the resulting
        disk file A containing 1/2 of the data would be
        128L*128*256*sizeof(float)*2 bytes in size. That extra
        little factor of 2 caused me no end of pain when first
        trying to use this routine on real valued data.

        Inputs:   fnames[] string array, contains the filenames
                           of the 4 storage arrays on disk.

                  nn[]     int array, contains the length of each
                           dimension of the data array.

                  isign    int, forward or reverse transform flag.
                            1 = forward transform
                           -1 = inverse transform

        Outputs:  none.

*************************************************************************/
{
  char warn;
  int cc,na,nb,nc,nd;
  float tempr,tempi,*afa,*afb,*afc;
  double wr,wi,wpr,wpi,wtemp,theta;
  static int mate[5] = {0,2,1,4,3};
  FILE *file[5];

  IDL_FILE_STAT fstat;
  IDL_ULONG i,j,j12,jk,k,kk,n=1,mm,kc=0,kd,ks,kr,nr,ns,nv;
  IDL_VPTR fnames,nnv,dum[2];
  IDL_LONG luns[4],*nn;
  IDL_VARIABLE unit, name;
  IDL_STRING *fn;
  int ndim, isign;

#define KBF 1048576
  /*Setting KBF to 128 or 512 gave totally screwed up transforms
    on the test case run on my SGI O2.  KBF=1024 worked, but no faster than 256.
    Leaving it at 256... */
  /*Bigger test cases take 25 hours to transform! Need to set KBF MUCH larger
```

```
    in order to make the routine reasonable - or look into virtual memory tricks...*/

  /* printf("argc = %d\n",argc); */

  /*get file pointers set up based on argv[0] array*/
  fnames= argv[0];
  /*print_vptr(luns,"luns"); */
  IDL_ENSURE_STRING(fnames);
  IDL_ENSURE_ARRAY(fnames);
  if (fnames->value.arr->n_dim != 1)
    IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
                "Logical unit argument must be a 1-D array.");
  if (fnames->value.arr->n_elts != 4)
    IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
                "Wrong number of filenames passed.");
/*   printf("\nfnames type: %d\n",fnames->type); */
/*   printf("fnames nelt = %d\n",fnames->value.arr->n_elts); */

  /*get filenames */
  printf("\nStorage Files:\n");
  fn = &fnames->value.arr->data[0];
  for(i=0;i<4;i++) {
/*     printf("filename %d length = %d\n",i,fn->slen);  */
/*     printf("filename %d type = %d\n",i,fn->stype); */
    printf("File %d = %s\n",i+1,fn->s);
    fn++;
  }

  /* Open files */
  printf("\nOpening files...\n");
  fn = &fnames->value.arr->data[0];
  for(i=0;i<4;i++) {
    /* get file unit */
    dum[0] = &unit;
    unit.type = IDL_TYP_LONG;
    unit.flags = 0;
    IDL_FileGetUnit(1,dum);
    luns[i] = dum[0]->value.l;

    /* set up filename */
/*     printf("Opening LUN %d\n",(int) luns[i]); */
    name.type = IDL_TYP_STRING;
    name.flags = IDL_V_CONST;
    name.value.str.s = fn->s;
    name.value.str.slen = sizeof(fn->s)-1;
    name.value.str.stype = 0;
```

```
    dum[1] = &name;

    /* NOTE: Extern.Dev.Guide function prototype incorrect: needs one more arg */
    IDL_FileOpen(2, dum, (char *) 0, IDL_OPEN_R | IDL_OPEN_W,
                 IDL_F_UNIX_F77, 1, 0);
    fn++;
  }
  printf("Files successfully opened.\n");

  /*Check file status */
  printf("\nEnsuring file status...\n");
  for(i=0;i<4;i++){
    if(IDL_FileEnsureStatus(IDL_MSG_LONGJMP, (int) luns[i],
                            IDL_EFS_OPEN | IDL_EFS_READ | IDL_EFS_WRITE))
      printf("File %d is open for reading and writing.\n",
             (int) luns[i]);
  }
  printf("Status ensured.\n");

  printf("\nGetting FILE pointers...\n");
  for (i=0;i<4;){
    IDL_FileStat((int) luns[i], &fstat);
/*     printf("Filename from FileStat(): %s\n",fstat.name); */
    file[++i] = fstat.fptr;  /*Note unity offset required by NR!! */
  }
  printf("FileStats complete.\n");

  /* Establish dimensions of data array */
  nnv = argv[1];
  /*print_vptr(nnv,"nnv"); */
  IDL_ENSURE_SIMPLE(nnv);
  IDL_ENSURE_ARRAY(nnv);
  nn = (IDL_LONG *) nnv->value.arr->data;
  if (nnv->value.arr->n_dim != 1)
    IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
                "Array dimensions argument must be a 1-D array.");
  ndim = nnv->value.arr->n_elts; /*confusing name, but keep NR names intact... */

  /* get isign */
  isign = argv[2]->value.i;
  /* printf("isign=%d\n",isign);  */
  if (isign == 1) {
    for (i=0;i<ndim;i++) {
      if (nn[i] <  KBF) warn=1;
    }
    if (warn) IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_INFO,
```

```
                    "At least one array dimension < KBF record length: Transform will be wrapp
  }

  nn -= 1;  /* unity offset for all NR routines!! */
/*   for (i=1;i<=ndim;i++) printf("nn(%d) = %d      ",i,nn[i]);  */

  /* Here begins the NR routine (almost) unmodified from the original:    */
/*   printf("\n\nsizeof(float) = %d\n",sizeof(float)); */
  afa=vector(1,KBF);
  afb=vector(1,KBF);
  afc=vector(1,KBF);
  for (j=1;j<=ndim;j++) {
    n *= nn[j];
    if (nn[j] <= 1) IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
                                "Invalid float or wrong ndim in fourfs");
  }
/*   printf("\nn = %d\n",n); */
  nv=1;
  jk=nn[nv];
  mm=n;
  ns=n/KBF;
/*   printf("ns = %d\n",ns); */
  nr=ns >> 1;
/*   printf("nr = %d\n",nr); */
  kd=KBF >> 1;
  ks=n;
  fourew(file,&na,&nb,&nc,&nd);

/* First phase of the transform starts here */
/*   printf("\nStarting first phase...\n"); */
/*   printf("mm = %d\n",mm); */
/*   cnt=0; */
  for (;;) {        /*Start of the computing pass. */
    /* Done M-K-1 times, where n = 2^M and KBF=2^K */
/*     printf("count = %d\n",++cnt); */
    theta=isign*3.141592653589793/(n/mm);
    wtemp=sin(0.5*theta);
    wpr = -2.0*wtemp*wtemp;
    wpi=sin(theta);
    wr=1.0;
    wi=0.0;
    mm >>= 1;
/*     printf("    mm = %d\n",mm); */
    for (j12=1;j12<=2;j12++) {
/*       printf("    j12 = %d\n",j12); */
      kr=0;
```

```
/*      printf("ftell(file[na]): %d\n",ftell(file[na])); */
/*      printf("ftell(file[nb]): %d\n",ftell(file[nb])); */

     do {
/*     printf("kr = %d\n",kr); */
       cc=fread(&afa[1],sizeof(float),KBF,file[na]);
       if (cc != KBF) {
         printf("KBF = %d, cc = %d\n",KBF,cc);
         IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
                                  "read error in fourfs 1");
       }
       cc=fread(&afb[1],sizeof(float),KBF,file[nb]);
       if (cc != KBF) {
         printf("KBF = %d, cc = %d\n",KBF,cc);
         IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
                                  "read error in fourfs 2");
       }

       for (j=1;j<=KBF;j+=2) {
         tempr=((float)wr)*afb[j]-((float)wi)*afb[j+1];
         tempi=((float)wi)*afb[j]+((float)wr)*afb[j+1];
         afb[j]=afa[j]-tempr;
         afa[j] += tempr;
         afb[j+1]=afa[j+1]-tempi;
         afa[j+1] += tempi;
       }
       kc += kd;
       if (kc == mm) {
         kc=0;
         wr=(wtemp=wr)*wpr-wi*wpi+wr;
         wi=wi*wpr+wtemp*wpi+wi;
       }

       cc=fwrite(&afa[1],sizeof(float),KBF,file[nc]);
       if (cc != KBF) {
         printf("KBF = %d, cc = %d\n",KBF,cc);
         IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
                    "write error in fourfs 1");
       }

       cc=fwrite(&afb[1],sizeof(float),KBF,file[nd]);
       if (cc != KBF) {
         printf("KBF = %d, cc = %d\n",KBF,cc);
         IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
                                  "write error in fourfs 2");
       }
```

```
      } while (++kr < nr); /*end of the big DO loop */

      if (j12 == 1 && ks != n && ks == KBF) {
        na=mate[na];
        nb=na;
      }
      if (nr == 0) break;
    }
    fourew(file,&na,&nb,&nc,&nd);    /*Start of the permutation pass */
    jk >>= 1;
    while (jk == 1) {
      mm=n;
      jk=nn[++nv];
    }
    ks >>= 1;
    if (ks > KBF) {
      for (j12=1;j12<=2;j12++) {
        for (kr=1;kr<=ns;kr+=ks/KBF) {
          for (k=1;k<=ks;k+=KBF) {
            cc=fread(&afa[1],sizeof(float),KBF,file[na]);
            if (cc != KBF) IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
                                       "read error in fourfs 3");
            cc=fwrite(&afa[1],sizeof(float),KBF,file[nc]);
            if (cc != KBF) IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
                                       "read error in fourfs 4");
          }
          nc=mate[nc];
        }
        na=mate[na];
      }
      fourew(file,&na,&nb,&nc,&nd);
    } else if (ks == KBF) nb=na;
    else break;
  }

/*   printf("Starting second phase...\n"); */
  /*Second phase starts here. Done K times where KBF = 2^K */
  j=1;
  for (;;) {
    theta=isign*3.141592653589793/(n/mm);
    wtemp=sin(0.5*theta);
    wpr = -2.0*wtemp*wtemp;
    wpi=sin(theta);
    wr=1.0;
    wi=0.0;
```

```
    mm >>= 1;
/*     printf("    mm = %d\n",mm); */
    ks=kd;
    kd >>= 1;
    for (j12=1;j12<=2;j12++) {
      for (kr=1;kr<=ns;kr++) {
        cc=fread(&afc[1],sizeof(float),KBF,file[na]);
        if (cc != KBF) IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
                                   "read error in fourfs 5");
        kk=1;
        k=ks+1;
        for (;;) {
          tempr=((float)wr)*afc[kk+ks]-((float)wi)*afc[kk+ks+1];
          tempi=((float)wi)*afc[kk+ks]+((float)wr)*afc[kk+ks+1];
          afa[j]=afc[kk]+tempr;
          afb[j]=afc[kk]-tempr;
          afa[++j]=afc[++kk]+tempi;
          afb[j++]=afc[kk++]-tempi;
          if (kk < k) continue;
          kc += kd;
          if (kc == mm) {
            kc=0;
            wr=(wtemp=wr)*wpr-wi*wpi+wr;
            wi=wi*wpr+wtemp*wpi+wi;
          }
          kk += ks;
          if (kk > KBF) break;
          else k=kk+ks;
        }
        if (j > KBF) {
          cc=fwrite(&afa[1],sizeof(float),KBF,file[nc]);
          if (cc != KBF) IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
                                     "write error in fourfs 3");
          cc=fwrite(&afb[1],sizeof(float),KBF,file[nd]);
          if (cc != KBF) IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
                                     "write error in fourfs 4");
          j=1;
        }
      } /* end of kr loop */
      na=mate[na];
    } /* end of j12 loop */

    fourew(file,&na,&nb,&nc,&nd);
    jk >>= 1;
/*     printf("jk = %d\n",jk); */
    if (jk > 1) continue;
```

```
    mm=n;
    do {
      if (nv < ndim) {
        jk=nn[++nv];
/*      printf("remainders: jk = %d\n",jk); */
      }
      else {
        free_vector(afc,1,KBF);
        free_vector(afb,1,KBF);
        free_vector(afa,1,KBF);
        /*Free the file units*/
        printf("\nFreeing the file units...\n");
        for (i=0;i<4;i++) {
/*        printf("Freeing LUN %d\n",(int) luns[i]); */
          /* setup IDL_VPTR */
          unit.type = IDL_TYP_LONG;
          unit.flags = 0;
          unit.value.l = luns[i];
          dum[0] = &unit;
          IDL_FileFreeUnit(1,dum);
        }
        return;
      }
    } while (jk == 1);
  }
}
#undef KBF
#undef NRANSI

/*************************************************************/

void fourew(FILE *file[5], int *na, int *nb, int *nc, int *nd)

/*************************************************************/
{
        int i;
        FILE *ftemp;

        for (i=1;i<=4;i++) rewind(file[i]);
        SWAP(file[2],file[4]);
        SWAP(file[1],file[3]);
        *na=3;
        *nb=4;
        *nc=1;
        *nd=2;
}
```

```
#undef SWAP


/* Applies sub-v or sub-fundamental space-time filters in
Fourier domain using the temporary file structure given by
fourfs.c. Useful for p-mode filtering large datasets that
don't fit in RAM. */

#include <stdio.h>
#include <math.h>
#include "export.h"
#include "nr.h"
#include "nrutil.h"

#define ARRLEN(arr) (sizeof(arr)/sizeof(arr[0]))
#define SWAP(a,b)   ftemp=(a);(a)=(b);(b)=ftemp
#define NRANSI

IDL_VPTR IDL_pmodefilter( int argc, IDL_VPTR *argv );
void filesetup( IDL_VPTR );

int IDL_Load(void)
{
  static IDL_SYSFUN_DEF2 new_pro[] = {
    { (IDL_FUN_RET) IDL_pmodefilter, "PMODEFILTER", 5, 5, 0, 0}
  };

  return IDL_SysRtnAdd(new_pro, TRUE, ARRLEN(new_pro));
}


#define NF 2     /*current algorithm stores transformed data in 2 storage arrays */
FILE *file[NF];
IDL_LONG luns[NF];

/*************************************************************************/
IDL_VPTR  IDL_pmodefilter( int argc, IDL_VPTR *argv )


    /*      Inputs:   fnames[2] string array: filenames of temporary files.
                      [nx,ny,nt]: dimensions of ORIGINAL (ie. not transposed) data cube.
                      dxy: spatial scale units in km/pixel.
                      dt: temporal scale units in seconds.
                      vtop: cutoff velocity.
```

```
            Outputs:  3/7/01. last image read from data cube.

see pmf_test.pro for testing routine.
*************************************************************************/
{
  char *data0,*data1;
  long *xyt,i,j,k,jk,jkc,ll;
  unsigned int ndim,fsk,sf=sizeof(float);
  float dk,dkx,dky,dt,dw,dxy,kx,ky,kc,v,vtop,w,wc;
  size_t cc;

  IDL_VARIABLE unit;
  IDL_VPTR fn,nnv,dum[2],tmp0,tmp1,dxyv,dtv,vtopv;
  IDL_LONG nc,nf,nt,nt2,nx,nx2,ny,ny2,ntny[2];


  /*open files and assign logical units */
  fn = argv[0];
  filesetup( fn );


  /*Get dimensions of original data cube */
  nnv = argv[1];
  IDL_ENSURE_SIMPLE(nnv);
  IDL_ENSURE_ARRAY(nnv);
  if (nnv->value.arr->n_elts != 3)
    IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
        "Data array must be 3-D.");
  xyt = nnv->value.arr->data;

  nx = xyt[0];
  nx2 = nx/2;
  ny = xyt[1];
  ny2 = ny/2;
  nt = xyt[2];
  nt2 = nt/2;
  nc = nt*ny;       /* number of complex elements in each w-ky image*/
  nf = 2*nc;        /* number of float elements in each w-ky image (real,imag) */
  printf("Dimension of images in series from nn array: \n");
  printf("\tnx = %d\n",nx);
  printf("\tny = %d\n",ny);
  printf("Number of images in series from nn array: \n");
  printf("\tnt = %d\n",nt);
  printf("Number of floating point variables in a w-ky image: \n");
  printf("\tnf = %d\n",nf);
```

```
/*Units and cutoff velocity - note assumptions on data types!*/
dxyv = argv[2];
if (dxyv->type != 4)
  IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
              "DXY argument must be floating point.");
dxy = dxyv->value.f;

dtv = argv[3];
if (dtv->type != 4)
  IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
              "DT argument must be floating point.");
dt = dtv->value.f;

vtopv = argv[4];
if (vtopv->type != 4)
  IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
              "VTOP argument must be floating point.");
vtop = vtopv->value.f;

/*Fourier domain units */
dw = 1./nt/dt;
wc = 1./2/dt;
dkx = 1./nx/dxy;
dky = 1./ny/dxy;
kc = 1./2/dxy;
printf("dw = %f\n",dw);
printf("dkx = %f\n",dkx);
printf("dky = %f\n",dky);

/* Get temporary array for w-ky image planes */
ntny[0]=nt;
ntny[1]=ny;
data0 = IDL_MakeTempArray(IDL_TYP_COMPLEX,2,ntny,IDL_ARR_INI_NOP,&tmp0);
data1 = IDL_MakeTempArray(IDL_TYP_COMPLEX,2,ntny,IDL_ARR_INI_NOP,&tmp1);

/* Loop through storage arrays. Dimensions of each storage file = (nt,ny,nx/2).
   Note symmetry: file1 is mirror of file0 in time axis for real input*/
/* 3/7/01: This is now successfully reading the w-ky TRANSPOSE image
   from the data cube. NEXT: figure out how to read,
   filter it, and transpose it back so it's ready for the IFFT */
for (i=0;i<nx2;i++) {
  /*read NxM complex image  */

  /* read a w-ky slice image from 1st storage file */
  fsk = fseek(file[0],i*nf*sizeof(float),SEEK_SET);
  cc = fread(&data0[0],sizeof(float),nf,file[0]);
```

```
    if (cc != nf) IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
                              "Read failure on file[0].");

    /* read a w-ky slice image from 2nd storage file which is in reverse kx order*/
    fsk = fseek(file[1],(nx2-i-1)*nf*sizeof(float),SEEK_SET);
    cc = fread(&data1[0],sizeof(float),nf,file[1]);
    if (cc != nf) IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
                              "Read failure on file[1].");

    /*condition the transform images */
    kx = i*dkx;
    for (jk=0;jk<=(sf*nf);jk++){   /*loop over all bytes in the image array */

      jkc = jk/(2*sf); /*pixel index */
      j = jkc % nt;  /*w dimension */
      if (j > nt2) j=nt-j;
      w = j*dw;

      k = jkc/nt;    /*ky dimension */
      if (k > ny2) k=ny-k;
      ky = k*dky;
      dk = sqrt( kx*kx + ky*ky );

      if (dk > 0) v = w/dk;
      if (v > vtop || dk == 0) {
        data0[jk] = 0;
        data1[jk] = 0;
      }
/*       if (k < 64) data0[jk] = 0; */
    }


    /*write the image back in file0...  */
    fsk = fseek(file[0],i*nf*sizeof(float),SEEK_SET);
    cc = fwrite(&data0[0],sizeof(float),nf,file[0]);
    if (cc != nf) IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
                              "Write failure to file[0].");


    /* ...and file1 */
    fsk = fseek(file[1],(nx2-i-1)*nf*sizeof(float),SEEK_SET);
    cc = fwrite(&data1[0],sizeof(float),nf,file[1]);
    if (cc != nf) IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
                              "Write failure to file[1].");

  }
```

```
  /* Close the files */
  for (i=0;i<NF;i++) {
    dum[0] = &unit;
    unit.type = IDL_TYP_LONG;
    unit.flags = 0;
    unit.value.l = luns[i];
    IDL_FileClose(1,dum,NULL);
  }


  return tmp0;
}


#undef SWAP
#undef NRANSI

/*************************************************************/

void filesetup( IDL_VPTR fnames )

/*************************************************************/
{
  int i,nfiles;

  IDL_FILE_STAT fstat;
  IDL_STRING *fn;
  IDL_VPTR dum[2];
  IDL_VARIABLE unit, name;

  /* Check properties */
  IDL_ENSURE_STRING(fnames);
  IDL_ENSURE_ARRAY(fnames);
  if (fnames->value.arr->n_dim != 1)
    IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
                "Temporary files array must be 1-D.");
  nfiles = fnames->value.arr->n_elts;
  if (nfiles != NF)
    IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
                "Wrong number of filenames passed.");
/*   printf("\nfnames type: %d\n",fnames->type); */
/*   printf("fnames nelt = %d\n",fnames->value.arr->n_elts); */

  printf("\nStorage Files:\n");
  fn = &fnames->value.arr->data[0];
```

```
  for(i=0;i<nfiles;i++) {
/*     printf("filename %d length = %d\n",i,fn->slen);  */
/*     printf("filename %d type = %d\n",i,fn->stype); */
    printf("File %d = %s\n",i,fn->s);
    fn++;
  }

  /* Open files */
  printf("\nOpening files...\n");
  fn = &fnames->value.arr->data[0];
  for(i=0;i<nfiles;i++) {
    /* get file unit */
    dum[0] = &unit;
    unit.type = IDL_TYP_LONG;
    unit.flags = 0;
    IDL_FileGetUnit(1,dum);
    luns[i] = dum[0]->value.l;

    /* set up filename */
/*     printf("Opening LUN %d\n",(int) luns[i]); */
    name.type = IDL_TYP_STRING;
    name.flags = IDL_V_CONST;
    name.value.str.s = fn->s;
    name.value.str.slen = sizeof(fn->s)-1;
    name.value.str.stype = 0;
    dum[1] = &name;

    /* NOTE: Extern.Dev.Guide function prototype incorrect: needs one more arg */
    IDL_FileOpen(2, dum, (char *) 0, IDL_OPEN_R | IDL_OPEN_W,
                 IDL_F_UNIX_F77, 1, 0);
    fn++;
  }
  printf("Files successfully opened.\n");

  /*Check file status */
  printf("\nEnsuring file status...\n");
  for(i=0;i<nfiles;i++){
    if(IDL_FileEnsureStatus(IDL_MSG_LONGJMP, (int) luns[i],
                            IDL_EFS_OPEN | IDL_EFS_READ | IDL_EFS_WRITE))
      printf("File %d is open for reading and writing.\n",
             (int) luns[i]);
  }
  printf("Status ensured.\n");

  printf("\nGetting FILE pointers...\n");
  for (i=0;i<nfiles;i++){
```

```
    IDL_FileStat((int) luns[i], &fstat);
/*      printf("Filename from FileStat(): %s\n",fstat.name); */
    file[i] = fstat.fptr;
  }
  printf("FileStats complete.\n");

  return;
}
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	23-March-2001	Final Report

4. TITLE AND SUBTITLE
High Resolution and Modeling of Solar Structures Status Report

5. FUNDING NUMBERS
Contract NASW-98008

6. AUTHORS
Thomas E. Berger

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Lockheed Martin Missiles & Space
Advanced Technology Center
3251 Hanover Street, H1-12/252
Palo Alto, CA 94304-1191

8. PERFORMING ORGANIZATION REPORT NUMBER
HRMSS-99-01

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Dr. William J. Wagner
SR,NASA Headquarters
Washington, DC 20546

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

High Resolution and Modeling of Solar Structures is a research project managed by the Lockheed Martin advanced Technology Center for the purpose of studying the smallest visible magnetic structures on the Sun. Specifically, the project seeks to understand (a) the mechanisms by which 100-200 km diameter magnetic elements in the photosphere radiate in the molecular band near 4300 Angstroms in the solar spectrum, and (b) the formation and interaction dynamics of these elements in relation to larger magnetic structures such as sunspots. This report summarizes the results of the three-year contract.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Solar Physics, Sunspots	56
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	

NSN 7540-01-280-5500 Computer Generated STANDARD FORM 298 (Rev 2-89)
Prescribed by ANSI Std 239-18298102