SPACE TELESCOPE SCIENCE INSTITUTE

INSTRUMENT SCIENCE REPORT

HSP-006-

TITLE: Polynomial Fitting for HSP RSDP Algorithms

AUTHOR: Richard L. White

DATE: 27 January 1988

ABSTRACT

Most of the calibration parameters used by the HSP RSDP algorithms are allowed to be cubic polynomial functions of the instrument temperature and the epoch of the observation. This paper describes the best method for determining both the polynomial coefficients and the uncertainty in the fit as a function of temperature and time. The uncertainty is required for the propagation of errors through the RSDP algorithms, which may be added to the RSDP in the future.

DISTRIBUTION:

ISB Macchetto, Jenkner, Cox
CSC
SDAS Hanisch
SOGS
IDT Bless, Percival
OPD Pence, Hsu

SPACE TELESCOPE SCIENCE INSTITUTE

INSTRUMENT SCIENCE REPORT

TITLE: Polynomial Fitting for HSP RSDP Algorithms

AUTHOR: Richard L. White

DATE: 27 January 1988

ABSTRACT

DISTRIBUTION:

ISB Macchetto, Jenkner, Cox

OPD Pence, Hsu

CSC

SDAS Hanisch

SOGS

IDT Bless, Percival

SPACE TELESCOPE SCIENCE INSTITUTE

INSTRUMENT SCIENCE REPORT

TITLE: Polynomial Fitting for HSP RSDP Algorithms

AUTHOR: Richard L. White

DATE: 27 January 1988

ABSTRACT

Most of the calibration parameters used by the HSP RSDP algorithms are allowed to be cubic polynomial functions of the instrument temperature and the epoch of the observation. This paper describes the best method for determining both the polynomial coefficients and the uncertainty in the fit as a function of temperature and time. The uncertainty is required for the propagation of errors through the RSDP algorithms, which may be added to the RSDP in the future.

DISTRIBUTION:

ISB Macchetto, Jenkner, Cox
CSC
SDAS Hanisch
SOGS
IDT Bless, Percival
OPD Pence, Hsu

POLYNOMIAL FITTING FOR HSP RSDP ALGORITHMS

Richard L. White, 26 January 1988

Summary

Most of the calibration parameters used by the HSP RSDP algorithms are allowed to be cubic polynomial functions of the instrument temperature and the epoch of the observation. This paper describes the best method for determining both the polynomial coefficients and the uncertainty in the fit as a function of temperature and time. The uncertainty is required for the propagation of errors through the RSDP algorithms, which may be added to the RSDP in the future.

Introduction

In order to permit the best possible accuracy for the HSP "pipeline" data calibration, most of the calibration parameters (which include dark count rates, relative sensitivity factors, etc.) are not assumed to be constant but are allowed to vary with temperature and time. The value of one of these parameters is determined from

$$C = \sum_{i=0}^{3} \sum_{j=0}^{3} c_{ij} (T - T_0)^i (t - t_0)^j \quad , \tag{1}$$

where T is some HSP temperature (from telemetry) at the start of the observation, t is the epoch of the observation, T_0 and t_0 are reference temperatures and times for the calibration quantity, and c_{ij} are the calibration coefficients for calibration parameter C.

I describe here what appears to be the best possible way to derive these coefficients from a set of calibration observations at various temperatures and times. The technique has the following desirable characteristics:

(1) it gives the coefficients for the best (least squares) polynomial in T and t by fitting in both variables simultaneously;
(2) it is very robust;
(3) it gives a measure of the quality of the fit; and
(4) it permits error estimates on the fit as a function of T and t.

This paper describes the method used, gives a Fortran listing of a program which performs the fit, and shows an example fit using the program. Some implications of the use of this technique for deriving RSDP parameters are also discussed.

Method

Chapter 14 of *Numerical Recipes* (Press *et al.* 1986) thoroughly discusses the techniques for solving linear least squares problems in one dimension. I repeat here only enough of their discussion to show how their method of choice is used and how it is easily generalized to fitting in an arbitrary number of dimensions. I follow their notation.

One-Dimensional Fitting

First consider the 1-dimensional problem. Suppose we have a set of N data points (x_i, y_i) to which we wish to fit a linear combination of M basis functions,

$$y(x) = \sum_{k=1}^{M} a_k X_k(x) \tag{2}$$

where $X_1(x), \ldots, X_M(x)$ are arbitrary functions of x. According to Press *et al.*, the best method for solving this problem is singular value decomposition (SVD). SVD successfully solves for the a_k even in the case when the $X_k(x)$ are not well-distinguished by the data.

An extreme example of this occurs if the $X_k(x)$ are not linearly independent. Suppose, for example, that we are trying to fit the sum of 3 functions $X_0 = 1$, $X_1 = x$, and $X_2 = 1 + x$. Obviously there is not a unique solution for (a_0, a_1, a_2) in this case, and most numerical techniques will either fail to find any solution or will find a solution in which *e.g.*, a_0 and a_1 are very large positive numbers and a_2 is a very large negative number, with the resulting large numbers nearly canceling to produce the desired solution. However, SVD returns the solution with the smallest length $\sum a_k^2$, which is in some sense the most reasonable solution to such an underdetermined problem.

Obviously one tries to avoid linearly dependent basis functions; however, a similar circumstance arises when the data insufficiently sample the various basis functions, so that the functions are practically indistinguishable. (A familiar example of this occurs when one tries to fit a Fourier series to evenly sampled data. The even sampling makes it impossible for the data to distinguish between low frequencies and frequencies above the Nyquist limit.) In that case as well, SVD is very well behaved. According to Press *et al.*, SVD *cannot fail* regardless of how ill-conditioned the problem may be.

Numerical Recipes lists several subroutines which can be used for 1-dimensional least squares fitting. `SVDFIT` performs the fit and returns the coefficients; it also returns the χ^2 value for the fit. `SVDVAR` calculates the covariance matrix, which describes the uncertainty in the parameters (this is discussed further below). `SVDFIT` calls `SVDCMP` and `SVBKSB`, which are also *Numerical Recipes* subroutines. The Fortran source code for all *Numerical Recipes* subroutines is available on the STScI VAX cluster in `DISK$SDAS:[NUMREC.SOURCE]`.

Multi-Dimensional Fitting

It is very simple to generalize the SVD technique given in *Numerical Recipes* so that it fits a multi-dimensional function to multi-dimensional data. The equations which are solved to find the coefficients a_k depend only on the value of the M basis functions X_k at the N data points x_i. Thus, the solution involves a matrix which looks like

$$\begin{pmatrix} X_1(x_1) & X_2(x_1) & \ldots & X_M(x_1) \\ X_1(x_2) & X_2(x_2) & \ldots & X_M(x_2) \\ \vdots & \vdots & & \vdots \\ X_1(x_N) & X_2(x_N) & \ldots & X_M(x_N) \end{pmatrix} \tag{3}$$

The functions $X_k(x)$ may be arbitrarily complicated. Now consider the general D-dimensional case. The basic matrix involved in the problem is *identical* to that given above: there

are still N data points and M functions. The only change is that the basis functions are $X_k(\mathbf{x})$, where $\mathbf{x}$ is a vector of length D. Since only the value of $X_k(\mathbf{x})$ at a finite number of points are involved in the fit, the fitting routine *does not care* that the basis functions depend on D variables instead of 1; it changes hardly at all from the 1-dimensional case.

I have modified the *Numerical Recipes* routine `SVDFIT` to handle fitting in an arbitrary number of dimensions; the resulting Fortran subroutine is called `SVDFITN` and is listed at the end of this paper. The subroutines `SVDCMP` and `SVBKSB` which it calls do not have to be changed, nor does `SVDVAR`.

I have also written another subroutine, called `POLYFIT2`, which is designed for the specific problem of fitting 2-dimensional cubic polynomials in $x - x_0$ and $y - y_0$ to data for the HSP RSDP algorithms. Both it and `TESTPOLY`, a sample program which reads in data and performs a fit using `POLYFIT2`, are also listed at the end of this paper, and their use is discussed further below.

Using the Results

Most of the parameters used by `POLYFIT2` are described in the comments in that routine and are self-explanatory. The `CHISQ` and `PROB` parameters, which quantify the accuracy of the fit, and the covariance matrix, returned in `COVAR`, may need some explanation.

The χ^2 of the fit is defined as

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y(x_i) - y_i}{\sigma_i} \right)^2 \quad , \tag{4}$$

where $y(x)$ is defined in equation (2) and σ_i is the standard deviation in the measurement of y_i. The fitting routines described in this paper solve for the set of coefficients a_k which minimize χ^2. If the function $y(x)$ is a good fit to the data, then the value of χ^2 will be about equal to the number of degrees of freedom in the fit, $N - M$. `POLYFIT2` returns both the χ^2 and the probability that random errors in the data would lead to a χ^2 value at least as large as that found, assuming that the measurement errors are normally distributed. If `PROB` is small (say, less than 0.05 or so) then the fit to the data is relatively poor, meaning that either the data is too complicated for the cubic polynomials to fit, or that the errors σ_i are underestimated. See *Numerical Recipes* for more discussion of interpreting the χ^2 and probabilities.

The covariance matrix `COVAR` gives the uncertainty and correlations in all the coefficients. It can be used to calculate the uncertainty in the value of the fitted function at any value of (x, y). Let E_{lm} be the covariance matrix for the coefficients a_k. The diagonal terms

$$E_{mm} = \sigma_m^2 \quad , \tag{5}$$

where σ_m^2 is the variance in coefficient a_m. The off-diagonal terms are the covariance of a_l and a_m,

$$E_{lm} = \mathrm{Cov}(a_l, a_m) = r_{lm}\sigma_l\sigma_m \quad , \tag{6}$$

where r_{lm} is the correlation coefficient between a_l and a_m ($-1 \le r_{lm} \le 1$). Note that E_{lm} is symmetrical, *i.e.*, $E_{lm} = E_{ml}$. The variance $v(x)$ in the fitted function $y(x)$ at any point x is given by

$$v(x) = \sigma^2(x) = \sum_{l=1}^{M} \sum_{m=1}^{M} E_{lm} X_l(x) X_m(x) \quad . \tag{7}$$

These expressions apply for 1-dimensional or multi-dimensional fits.

For a 1-dimensional polynomial fit ($X_k(x) = x^{k-1}, k = 1 \ldots M$), the variance $v(x)$ is a polynomial of order $2M$. Similarly, for a 2-dimensional fit, $v(x,y)$ is a 2-dimensional polynomial of order $2M$ in x and y. The uncertainty in the fit behaves as one would intuitively expect: it is small in regions where it is well-constrained by the data points, but outside those regions it increases rapidly.

For the specific case of a polynomial fit, the evaluation of $v(x)$ using equation (7) is very sensitive to round-off errors because the correlation coefficients are generally very close to ± 1. It is possible to evaluate $v(x)$ using double precision to reduce the round-off errors; however, it is faster and more accurate to use the special form of the covariance matrix to advantage. Consider a 1-dimension fit of a polynomial with M terms. Define matrices D and F as

$$D_{lm} = (-1)^{l+m}\sqrt{E_{ll}E_{mm}} = (-1)^{l+m}\sigma_l\sigma_m \quad , \tag{8}$$

and

$$F_{lm} = E_{lm} - D_{lm} \quad . \tag{9}$$

The variance at x is then

$$\begin{aligned} v(x) &= \sum_{l=1}^{M}\sum_{m=1}^{M}(D_{lm} + F_{lm})x^{l+m} \\ &= \sum_{l=1}^{M}\sum_{m=1}^{M}F_{lm}x^{l+m} + \sum_{l=1}^{M}\sigma_l(-x)^l\sum_{m=1}^{M}\sigma_m(-x)^m \\ &= \sum_{l=1}^{M}\sum_{m=1}^{M}F_{lm}x^{l+m} + \left[\sum_{l=1}^{M}\sigma_l(-x)^l\right]^2 \end{aligned} \tag{10}$$

By breaking the sum up this way, the cancellations which lead to round-off errors in the direct summation over E_{lm} are analytically removed and $v(x)$ can be accurately evaluated using single precision arithmetic. The routines `COVARSPLIT` and `VARP2` which use this technique to evaluate the variance are listed at the end of the paper. Their use is demonstrated in `TESTPOLY`.

Other Points

It may be worth mentioning here that the subroutine `SVDFITN` which is used by `POLYFIT2` can, like the *Numerical Recipes* routine `SVDFIT`, be used with any set of basis functions, not just with polynomial terms. For example, the strong correlation between the coefficients in the usual polynomial series $\sum a_k x^k$ can be greatly reduced by fitting instead a Legendre polynomial series $\sum a_k L_k(z)$, where L_k is the Legendre polynomial of order k and $z = (2x - b - a)/(b - a)$ is a linear transformation of x so that $-1 \le z \le 1$ when $a \le x \le b$. Similarly, coefficients for any other set of (multi-dimensional) functions may also be determined using `SVDFITN`.

I have not bothered to use Legendre polynomials in `TESTPOLY` because the trick discussed above for evaluating the variance $v(x)$ in the presence of strong correlations makes it unnecessary in the cases I have tried.

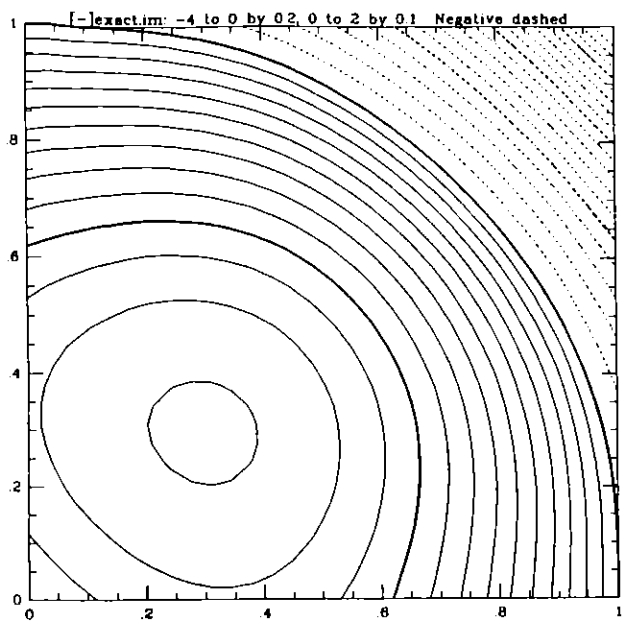

Figure 1. *Exact value of function used in example. Contour levels are* $-4, -3.8, -3.6, \ldots, 0, 0.1, 0.2, \ldots, 2$. *Negative contours are dashed.*

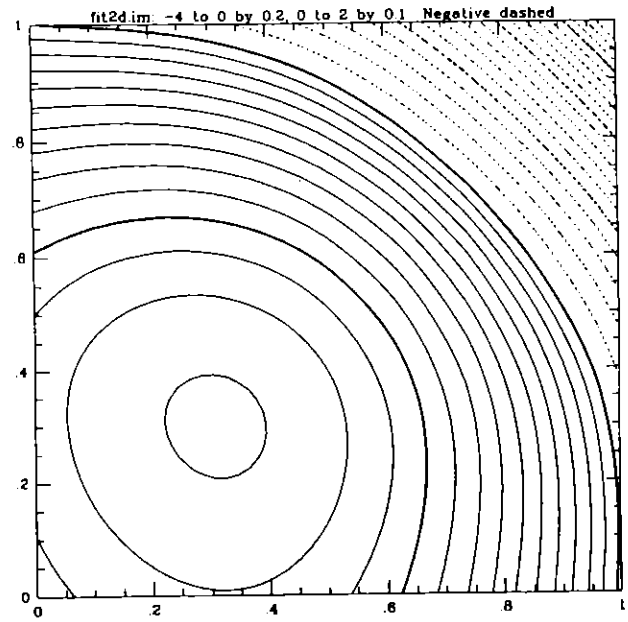

Figure 2. *Fitted function. Contour levels are same as for Figure 1.*

Example

This section describes an example of the use of POLYFIT2. Test data was generated by randomly sampling a function

$$F_e(x,y) = \begin{pmatrix} 1 & x & x^2 & x^3 \end{pmatrix} \begin{pmatrix} 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ -1 & 1 & 1 & -1 \\ -1 & -1 & -1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ y \\ y^2 \\ y^3 \end{pmatrix} \tag{11}$$

at 30 points in the interval $0 \le x \le 1$ and $0 \le y \le 1$. Gaussian noise with $\sigma = 0.01$ was added to the value of the exact function at each point.

Figure 1 is a contour plot of the exact function $F_e(x,y)$; Figure 2 shows a contour plot of the fitted function $F_f(x,y)$. The fitted function is obviously very similar to the exact function, though some small differences can be seen. Note, for example, the difference in the contours between the two functions near $x = 1$, $y = 0$.

Figure 3 shows a contour plot of the uncertainty in the fitted function, $\sigma(x,y)$. The (random) positions of the data points used in the fitting are also shown. The uncertainty in the fitted function is small near the data points, but is large near the edges of the fitted region where the data do not constrain the fit very well.

Figure 4 shows the difference between the fitted function and the exact function in standard deviations, *i.e.*, $[F_f(x,y) - F_e(x,y)]/\sigma(x,y)$. This difference is of order 1 or less over most of the fitted region, as it should be if $\sigma(x,y)$ is a good measure of the uncertainty.

It is a good idea to try fitting the data with a polynomial of lower order to see whether the higher order terms significantly improve the fit. This is done in TESTPOLY, which fits

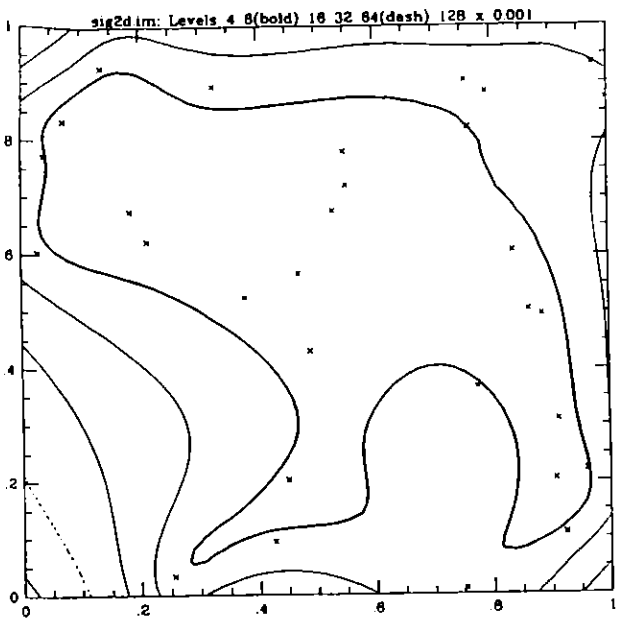

Figure 3. *Uncertainty in fitted function,* $\sigma(x, y)$. *Contour levels are* $8, 16, 32, 64, 128 \times 10^{-3}$. *Positions of the 30 data points used in the fitting are also shown.*

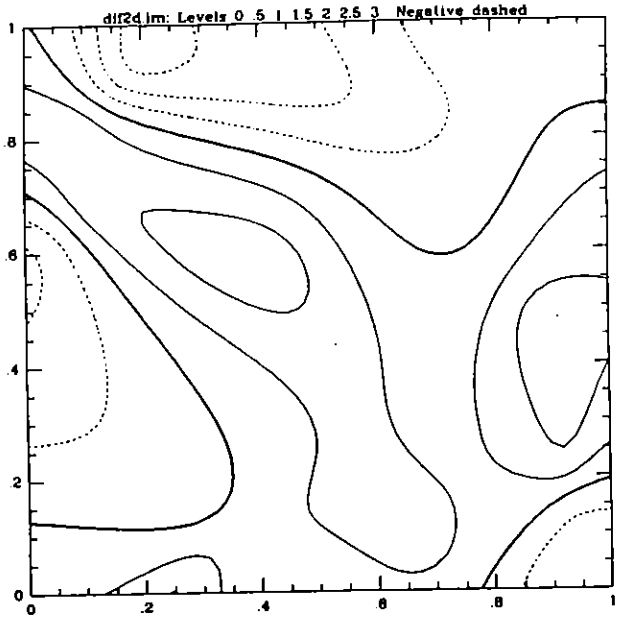

Figure 4. *Difference between fitted and exact functions in standard deviations,* $[F_f(x, y) - F_e(x, y)]/\sigma(x, y)$. *Contour levels are* $-1.5, -1, -0.5, 0, 0.5, 1$. *Negative contours are dashed.*

polynomials of order (N_x, N_y), with N_x and $N_y = 0 \ldots 3$. If a lower order fit is adequate, it is preferred because the uncertainty in the fit is substantially reduced as the order decreases, especially at the edges of the fitting region.

For the example, the minimum χ^2 as a function of the order of the polynomial being fit is

	N_x			
N_y	0	1	2	3
0	3.1×10^5	2.1×10^5	1.6×10^5	1.6×10^5
1	2.3×10^5	6.9×10^4	2.3×10^4	2.0×10^4
2	1.9×10^5	3.1×10^4	1.1×10^3	2.8×10^2
3	1.5×10^5	2.0×10^4	4.5×10^2	1.2×10^1

The probability of a χ^2 as bad as those given is essentially zero except for the case $N_x = N_y = 3$; then PROB is 0.58, indicating that cubic terms are necessary, and that when they are included, the fit is very good.

Discussion of Applications to HSP RSDP Algorithms

Currently the HSP RSDP algorithms do not include the capability of calculating the error introduced through uncertainties in the calibration parameters. However, it would be straightforward to include in the RSDP database the polynomial coefficients required to describe the uncertainty $\sigma(T, t)$ as well as the fitted function $C(T, t)$. There are 16

coefficients in the fitting function of equation (1), so the full error matrix has $16 \times 16 = 256$ terms; however, as pointed out above, the variance $v(T,t)$ is a 6th degree polynomial, so only $7 \times 7 = 49$ coefficients are need to fully characterize the uncertainty as a function of (T,t).

Thus, the error coefficients require about a factor of 3 more storage than the coefficients required to describe the parameter itself (49 versus 16). I have experimented with fitting a lower order polynomial to $\sigma(T,t)$ in the hopes that it would be possible to reduce these storage requirements, but to date I have not been successful. However, the scientific benefits of providing the observer with error estimates are sufficiently great that even a factor of 3 increase in storage requirements is probably justified.

The use of these error estimates and their propagation through the HSP RSDP algorithms will be discussed in a future paper.

Regardless of whether the error estimates are used, the method described here appears to be the best way to derive polynomial coefficients for the HSP RSDP algorithms. It should be incorporated into the CDBS algorithms as soon as possible.

Reference

Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. 1986, *Numerical Recipes*, (Cambridge: Cambridge University Press).

Fortran Listings

Subroutine SVDFITN

This is the primary subroutine used to perform fitting in multiple dimensions. It is a slightly modified version of the *Numerical Recipes* subroutine SVDFIT, which performs fits in one dimension. It calls *Numerical Recipes* subroutines SVDCMP and SVDKSB.

```
      subroutine SVDFITN (x,y,sig,ndata,nx,nxp,a,ma,
     &   u,v,w,mp,np,chisq,funcs)
c
c Linear least squares fitting of MA functions of NX variables using
c singular value decomposition.  Adapted from Numerical Recipes routine
c SVDFIT for linear least squares fitting using singular value
c decomposition.
c
c Programmer: R. White     Date: 7 January 1988
c
c Parameter    Description
c ---------    -----------
c X            (NX,NDATA) independent variable
c Y            (NDATA) measured values at X points
c SIG          (NDATA) standard deviations in Y
c NDATA        number of data points
c NX           number of independent variables
c NXP          physical dimension of X array (>=NX)
c A            (MA) coefficients of fitting functions (returned)
c MA           number of fitting functions
c U            (MP,NP) singular value decomposition array 1 (returned)
c V            (NP,NP) singular value decomposition array 2 (returned)
c W            (NP)    singular value decomposition array 3 (returned)
c                zero values of W indicate that the corresponding function
c                were excluded from the fit by the SVD.
c MP           Physical dimension of array U (>=NDATA)
c NP           Physical dimension of arrays U,V,W (>=MA)
c CHISQ        Chi-square for fit (returned)
c FUNCS        External subroutine FUNCS(X,NX,AFUNC,MA) which returns
c                the MA function values at X(NX) in array AFUNC(MA)
c
c General description:
c
c Given a set of NDATA points X(NX,I),Y(I) with individual standard
c deviations SIG(I), use chi-square minimization to determine the MA
c coefficients A of the fitting function:
c
c                MA
c        y(x) =Sum A   AFUNC (x).
c              i=1  i       i
c
c Here we solve the fitting equations using singular value decomposition
c of the NDATA by MA matrix, as in section 2.9.  Arrays U,V,W provide
c workspace on input, on output they define the singular value
c decomposition, and can be used to obtain the covariance matrix (using
c routine SVDVAR).  MP,NP are the physical dimensions of the matrices
c U,V,W, as indicated below.  It is necessary that MP>=NDATA and
c NP>=MA.  The program returns values for the MA fit parameters A, and
c chi-square, CHISQ.  The user supplies a subroutine FUNCS(X,NX,AFUNC,MA)
c that returns the MA basis functions evaluated at X in the array
c AFUNC.
c
c NMAX is maximum expected number of data points NDATA.
c MMAX is maximum expected number of fitting functions MA.
c
```



```
      parameter(NMAX=1000,MMAX=50,TOL=1.e-5)
      dimension x(nxp,ndata),y(ndata),sig(ndata),a(ma),v(np,np),
     *          u(mp,np),w(np),b(NMAX),afunc(MMAX)
c
c Accumulate coefficients of the fitting matrix
c
      do i=1,ndata
         call funcs(x(1,i),nx,afunc,ma)
         tmp=1./sig(i)
         do j=1,ma
            u(i,j)=afunc(j)*tmp
         enddo
         b(i)=y(i)*tmp
      enddo
c
c singular value decomposition
c
      call svdcmp(u,ndata,ma,mp,np,w,v)
c
c Edit the singular values, given TOL from the parameter statement.
c
      wmax=0.
      do j=1,ma
         if(w(j).gt.wmax) wmax=w(j)
      enddo
      thresh=TOL*wmax
      do j=1,ma
         if(w(j).lt.thresh) w(j)=0.
      enddo
c
c Solve U.a=b, where U is the matrix before SVD.
c
      call svbksb(u,w,v,ndata,ma,mp,np,b,a)
c
c Evaluate chi-square
c
      chisq=0.
      do i=1,ndata
         call funcs(x(1,i),nx,afunc,ma)
         sum=0.
         do j=1,ma
            sum=sum+a(j)*afunc(j)
         enddo
         chisq=chisq+((y(i)-sum)/sig(i))**2
      enddo
      return
      end
```

Fortran Listings (Continued)

Subroutine POLYFIT2 *and Function* FUNCPN

This subroutine uses the above routine SVDFITN to fit a 2-dimensional polynomial of degree up to 3 in each dimension to a set of data. Function FUNCPN calculates the polynomial basis functions used for the fitting; if different basis functions are required, it can be modified.

```
      subroutine POLYFIT2 (x,y,err,n,xoff,iorder,ma,a,covar,chisq,prob)
c
c POLYFIT2.FOR
c   Least-squares fit of a 2-dimensional polynomial (cubic or less in
c   each coordinate) to a 2-d set of data points using the numerical
c   recipes singular values decomposition method (subroutine SVDFITN).
c   Based on routines adapted from Numerical Recipes.
c
c Programmer: R. White       Date: 12 January 1988
c
c Input parameters:
c
c Parameter       Description
c ---------       -----------
c X               (2,N) array containing values of coordinates at
c                    each of N data points
c Y               (N) array containing data value at each point
c ERR             (N) array containing uncertainty (1 sigma) at each point
c N               number of data points (if N>1000 then NMAX will have
c                    to be modified here and in SVDFITN)
c XOFF            (2) array with offsets for each coordinate in
c                    polynomial (fit polynomial in X-XOFF)
c IORDER          (2) array with order of polynomial to fit in each
c                    coordinate direction (0=constant, 1=linear, etc.)
c                    (if either IORDER > 3 then NPMAX will have to be
c                    modified)
c MA              =(IORDER(1)+1)*(IORDER(2)+1), total number of
c                    terms in fit
c
c Output parameters:
c
c Parameter       Description
c ---------       -----------
c A               (0:IORDER(1),0:IORDER(2)) array of coefficients for
c                    polynomial terms
c COVAR           (0:IORDER(1),0:IORDER(2),0:IORDER(1),0:IORDER(2))
c                    covariance matrix for polynomial coefficients
c CHISQ           chi-square value for best fit
c PROB            probability of chi-square value this bad or worse
c
      parameter (NMAX=1000, NPMAX=3, NDIM=2)
      parameter (NVMAX=(NPMAX+1)**NDIM)
c
      external funcpn
c
      dimension x(NDIM,n),y(n),err(n),
     &   xoff(NDIM),iorder(NDIM),
     &   a(ma),covar(ma,ma),
     &   u(NMAX,NVMAX),v(NVMAX,NVMAX),w(NVMAX)
c
      common/pft2cm/ x0(NDIM),ipower(NDIM)
c
c check for errors in parameters
c
      if(n.gt.NMAX) then
         print *, 'Error in POLYFIT2: N=',n,' > NMAX=',NMAX
         print *, 'Modify NMAX in POLYFIT2 and SVDFIT for larger N'
```



```
         call exit
      endif
c
      do idim=1,NDIM
         if(iorder(idim).gt.NPMAX) then
            print *, 'Error in POLYFIT2: IORDER(',idim,
     &           ') > NPMAX=',NPMAX
            print *, 'Modify NPMAX in POLYFIT2 for higher orders'
            call exit
         endif
      end do
c
c set variables in common block (used by FUNCPN) and check value of MA
c
      icheck=1
      do idim=1,NDIM
         icheck=icheck*(iorder(idim)+1)
         ipower(idim)=iorder(idim)
         x0(idim)=xoff(idim)
      end do
c
      if(ma.ne.icheck) then
         print *, 'Error in POLYFIT2: MA, IORDER inconsistent'
         print *, 'MA=',ma,' IORDER=',iorder
         print *, 'MA should be product of (IORDER(i)+1)'
         call exit
      endif
c
c perform the polynomial fit
c
      call svdfitn(x,y,err,n,NDIM,NDIM,a,ma,
     &   u,v,w,n,ma,chisq,funcpn)
c
c probability of chi-square as bad as that found
c
      if(n-ma.gt.0) then
         prob=gammq(0.5*(n-ma),0.5*chisq)
      else
         prob=1.
      endif
c
c calculate covariance matrix from v,w scratch arrays used in SVDFIT
c
      call svdvar(v,ma,ma,w,covar,ma)
      return
      end
```

```
      subroutine FUNCPN (x,nx,afunc,ma)
c
c basis functions for polynomial in (x-x0), where x is a vector of
c   length nx
c
c Maximum power in dimension idim is ipower(idim), e.g. if
c ipower(1)=2, ipower(2)=1 then afunc is
c     1,x1,x1**2,x2,x2*x1,x2*x1**2
c
      dimension afunc(ma),x(nx)
      parameter (NDIM=2)
      common/pft2cm/ x0(NDIM),ipower(NDIM)
c
c check to make sure ma=product of ipower
c
      iprod=ipower(1)+1
      do idim=2,nx
         iprod=iprod*(ipower(idim)+1)
      end do
      if(iprod.ne.ma) then
         print *, 'MA inconsistent with IPOWER in FUNCPN'
         call exit
      endif
c
      afunc(1)=1.0
      do i=2,ipower(1)+1
         afunc(i)=(x(1)-x0(1))*afunc(i-1)
      end do
c
      i=ipower(1)+1
      do idim=2,nx
         ifirst=1
         inumber=i
         dx=x(idim)-x0(idim)
         do ip=2,ipower(idim)+1
            do j=ifirst,ifirst+inumber-1
               i=i+1
               afunc(i)=afunc(j)*dx
            end do
            ifirst=ifirst+inumber
         end do
      end do
      return
      end
```

Fortran Listings (Continued)

Subroutine COVARSPLIT *and Functions* VARP2 *and* VARX

These routines are used to calculate the variance as a function of (x, y) for a 2-dimensional polynomial fit. COVARSPLIT splits off the perfectly correlated part of the covariance matrix as described in the paper. VARP2 uses the split matrix to evaluate the variance at a given location. VARX calculates the variance directly using the sum in equation (7). VARX is called by VARP2 to calculate the off-diagonal part of the sum in equation (10); it can also be used to calculate the complete variance for non-polynomial basis functions if round-off error is not a problem.

```
      subroutine COVARSPLIT (covar,ma,nx,ny,diag,s2covar)
c
c Subroutine which splits covariance matrix from a 2-dimensional
c polynomial fit into a diagonal part (defined by diagonal values and
c +/-1 for off-axis correlations) and off-diagonal part (the rest) to
c reduce round-off errors.
c
c Programmer: R. White       Date: 19 January 1988
c
      dimension covar(ma,ma),diag(0:nx,0:ny),s2covar(ma,ma)
c
c on-diagonal terms
c
      do i=1,ma
         s2covar(i,i)=0.0
c
         ip1=mod(i-1,nx+1)
         ip2=(i-1)/(nx+1)
         diag(ip1,ip2)=sqrt(covar(i,i))
      end do
c
c off diagonal terms in lower triangular half
c
      do i=2,ma
         ip1=mod(i-1,nx+1)
         ip2=(i-1)/(nx+1)
         do j=1,i-1
            jp1=mod(j-1,nx+1)
            jp2=(j-1)/(nx+1)
            s1covar=sqrt(covar(i,i)*covar(j,j))
            if(mod(ip1+ip2+jp1+jp2,2).ne.0) s1covar=-s1covar
            s2covar(i,j)=covar(i,j)-s1covar
         end do
      end do
c
c now fill upper triangular half of matrix
c
      do i=1,ma-1
         do j=i+1,ma
            s2covar(i,j)=s2covar(j,i)
         end do
      end do
      return
      end
```



```
      function VARP2 (x,diag,s2covar,nx,ny)
c
c Returns variance in fitted 2-d polynomial at x using a better
c evaluation algorithm which relies on splitting off the diagonal
c values in the covariance matrix from the rest of the array.
c
c Parameters      Description
c ----------      -----------
c X               (2) array of coordinates
c DIAG            (0:NX,0:NY) diagonal sigmas
c S2COVAR         (MA,MA) where MA=(NX+1)*(NY+1)  covariance matrix
c                    after diagonal terms are removed
c NX,NY           Order of polynomial in each direction
c
      dimension x(2),diag(0:nx,0:ny),s2covar(0:nx,0:ny,0:nx,0:ny)
c
c first evaluate using diagonal terms
c
      vardiag=0.0
      do i=nx,0,-1
         term=0.0
         do j=ny,0,-1
            term=diag(i,j)-term*x(2)
         end do
         vardiag=term-vardiag*x(1)
      end do
c
      vardiag=vardiag**2
c
c now add off-diagonal contribution
c
      varp2=vardiag+varx(x,s2covar,(nx+1)*(ny+1))
      return
      end
```

```
      function VARX (x,covar,ma)
c
c Returns variance in fitted function at x.
c This works for 2-dimension data (i.e. x is an array of
c length 2).  It assumes that the common block /pft2cm/ used
c in FUNCPN has been set up properly (e.g. by a call to POLYFIT2
c to calculate the coefficients and covariance matrix).
c
c Parameters      Description
c ----------      -----------
c X               (2) array of coordinates
c COVAR           (MA,MA) covariance matrix
c MA              Size of covariance matrix
c
      parameter (NPMAX=3, NDIM=2, NVMAX=(NPMAX+1)**NDIM)
      dimension x(NDIM),covar(ma,ma)
      dimension funcs(NVMAX)
c
c get basis functions at x
c
      call funcpn(x,NDIM,funcs,ma)
c
c evaluate variance at x
c
      varx=0.
      do l=1,ma
         sum=0.
         do m=l+1,ma
            sum=sum+covar(l,m)*funcs(m)
         end do
         varx=varx + funcs(l)*(2.*sum+covar(l,l)*funcs(l))
      end do
      return
      end
```

Fortran Listings (Continued)

Program TESTPOLY *and Subroutines*

TESTPOLY is a program which uses the above routines to perform 2-dimensional polynomial fitting. It was used for the example described in the paper. It uses subroutines PRINT, WRITEFIT, EVALP2, and SIGP2, which are also listed. TESTPOLY writes binary output files from which contour plots like those shown in the paper can be produced. The *.IM files can be read by the MONGO plotting program using the IMAGE command.

```
      program TESTPOLY
c
c Test subroutines which fit a 2-dimensional polynomial to a 2-d set of
c data points using the numerical recipes singular values decomposition
c method (subroutine SVDFITN).
c
c Programmer: R. White       Date: 19 January 1988
c
      character*(*) FITFILE,SIGFILE,DIFFILE
      parameter (FITFILE='fit2d.im',SIGFILE='sig2d.im',
     &   DIFFILE='dif2d.im')
c
      parameter (NMAX=1000, NPMAX=3, NDIM=2)
      parameter (NVMAX=(NPMAX+1)**NDIM)
c
      dimension x0(NDIM),ipower(NDIM),
     &   diag(NVMAX),s2covar(NVMAX,NVMAX)
c
      dimension x(NDIM,NMAX),y(NMAX),err(NMAX),
     &   covar(NVMAX,NVMAX),iorder(NDIM),a(NVMAX),
     &   xmean(NDIM),xmin(NDIM),xmax(NDIM),
     &   chisq(0:NPMAX,0:NPMAX),prob(0:NPMAX,0:NPMAX)
      character*40 filename
c
      print *, 'Name of',NDIM,'-D data file'
      accept 1, nf,filename
1     format(q,a)
      iun=10
      open(unit=iun,file=filename(1:nf),status='old')
c
c read data from file
c
      n=0
      do j=1,NDIM
         xmean(j)=0.
         xmin(j)=1.e37
         xmax(j)=-1.e37
      end do
c
      do while(n.lt.NMAX)
         read(iun,*,end=99) (x(j,n+1),j=1,NDIM),y(n+1),err(n+1)
         n=n+1
         do j=1,NDIM
            xmean(j)=xmean(j)+x(j,n)
            xmin(j)=min(x(j,n),xmin(j))
            xmax(j)=max(x(j,n),xmax(j))
         end do
      end do
      print *, 'Too many data points.  Some were ignored.'

99    close(unit=iun)

      print *, n,' data points read from file ',filename(1:nf)
      do j=1,NDIM
         xmean(j)=xmean(j)/n
      end do
c
      print 2, (j,xmin(j),xmax(j),xmean(j),j=1,NDIM)
```



```
2     format(' x',i1,' range: min',f10.5,' max',f10.5,' mean',f10.5)
      print *, 'Enter', ( ' x0(', j, ')', j=1,NDIM )
      accept *, x0
      print *, 'Enter orders of polynomials 1 -',NDIM,
     &   ' to fit (<',NPMAX,')'
      accept *, (iorder(j),j=1,NDIM)
c
c perform the polynomial fit with polynomials of order 0,1,2,...,iorder
c print arrays with chi-square and probability for each order,
c print coefficients for final case only
c
      do i=0,iorder(1)
         ipower(1)=i
         do j=0,iorder(2)
            ipower(2)=j
            ma=(ipower(1)+1)*(ipower(2)+1)
c
            call polyfit2(x,y,err,n,x0,ipower,ma,
     &         a,covar,chisq(i,j),prob(i,j))
         end do
      end do
c
c print quality of fit as function of degree of polynomial
c
      print 4, 'Chi-square',(i,i=0,iorder(1))
4     format(1x,a,'  x1 across, x2 down'/3x,<iorder(1)+1>i11)
      do j=0,iorder(2)
         print 3, j,(chisq(i,j),i=0,iorder(1))
3        format(i3,1p<iorder(1)+1>e11.3)
      end do
      print 4, 'Probability',(i,i=0,iorder(1))
      do j=0,iorder(2)
         print 3, j,(prob(i,j),i=0,iorder(1))
      end do
c
c print covariance matrix (note that last call returned fit we use)
c
      call print(a,covar,ma,ipower,NDIM)
c
c split covariance into diagonal, off-diagonal parts so that it
c can be evaluated without huge roundoff errors
c
      call covarsplit(covar,ma,ipower(1),ipower(2),
     &   diag,s2covar)
c
c write fit, errors out to file
c
      print *, 'Current limits: min',xmin
      print *, '                max',xmax
      do i=1,NDIM
         print *, 'Enter new min,max for dimension',i
         accept *, xmin(i),xmax(i)
      end do
      call writefit(FITFILE,SIGFILE,DIFFILE,
     &   a,diag,s2covar,ipower(1),ipower(2),ma,x0,NDIM,xmin,xmax)
      print *, 'Fit MONGO image in file ',FITFILE
      print *, 'Sigma MONGO image in file ',SIGFILE
```

```
      print *, 'Ratio of error:sigma MONGO image in file ',DIFFILE
      call exit
      end
```



```
      subroutine PRINT (a,covar,ma,ipower,ndim)
c this routine only works for 2-dimensional data
      dimension a(ma),covar(ma,ma)
      dimension ipower(ndim)
c
      if(ndim.ne.2) then
         pause 'PRINT only works for 2 dimensions'
      endif
c
      print *, 'Coefficients, error matrix ((x-x0)**n basis,',
     &   ' var(l,m) is ratio to sqrt(var(l,l)*var(m,m)):'
      ii=0
      do j=0,ipower(2)
         do i=0,ipower(1)
            ii=ii+1
            if(ii.eq.1) then
               print 32,i,j,a(1),covar(1,1)
32             format(2i2,1pe13.5,1x,1pe9.2)
            else
               print 31,i,j,a(ii),
     &          (covar(ii,k)/sqrt(covar(ii,ii)*covar(k,k)),k=1,ii-1),
     &          covar(ii,ii)
31             format(2i2,1pe13.5,1x,0p<ii-1>f6.3,1pe9.2)
            endif
         end do
      end do
      return
      end
```

```
      subroutine WRITEFIT (fitfile,sigfile,diffile,
     &   a,diag,s2covar,nx,ny,ma,x0,ndim,xmin,xmax)
c
c write fit out to files for plotting with mongo
c this routine only works for 2-dimensional data
c
      character*(*) fitfile,sigfile,diffile
      dimension a(ma),diag(0:nx,0:ny),s2covar(ma,ma)
      dimension x0(ndim),xmax(ndim),xmin(ndim),x(2)
      parameter (NPMAX=3, NVMAX=(NPMAX+1)**2, NPRINT=60)
      dimension funcs(NVMAX),sig(NPRINT),fit(NPRINT),dif(NPRINT),
     &   aex(0:NPMAX,0:NPMAX)
c
      if(ndim.ne.2) then
         pause 'WRITEFIT only works for 2 dimensions'
      endif
c
c get exact values for coefficients
c
      print *, 'Exact polynomial coefficients'
      do j=0,NPMAX
         print 11, (i,j,i=0,NPMAX)
11       format(4(' A',2i1))
         accept *, (aex(i,j),i=0,NPMAX)
      end do
c
      iunfit=10
      iunsig=11
      iundif=12
      open (unit=iunfit,file=fitfile,status='new',form='unformatted',
     &   recordtype='fixed',recl=NPRINT)
      open (unit=iunsig,file=sigfile,status='new',form='unformatted',
     &   recordtype='fixed',recl=NPRINT)
      open (unit=iundif,file=diffile,status='new',form='unformatted',
     &   recordtype='fixed',recl=NPRINT)
c
      do j=1,NPRINT
         x(2)=xmin(2)+(j-1)*(xmax(2)-xmin(2))/(NPRINT-1)
         do i=1,NPRINT
            x(1)=xmin(1)+(i-1)*(xmax(1)-xmin(1))/(NPRINT-1)
c
c evaluate fitted function, uncertainty at x
c
            fit(i)=evalp2(x,a,diag,s2covar,nx,ny,ma,sig(i))
c
c evaluate exact function
c
            dx1=x(1)-x0(1)
            dx2=x(2)-x0(2)
            pex=0.0
            do m=3,0,-1
               pex=dx2*pex +
     &            aex(0,m)+dx1*(aex(1,m)+dx1*(aex(2,m)+dx1*aex(3,m)))
            end do
            dif(i)=(fit(i)-pex)/sig(i)
```



```
         end do
         write(iunfit) (fit(i),i=1,NPRINT)
         write(iunsig) (sig(i),i=1,NPRINT)
         write(iundif) (dif(i),i=1,NPRINT)
      end do
c
      close(unit=iunfit)
      close(unit=iunsig)
      close(unit=iundif)
      return
      end
```

```
      function EVALP2 (x,a,diag,s2covar,nx,ny,ma,sigx)
c
c Evaluate fitting function at x and return error estimate in sigx.
c This works for a 2-d polynomial fit.
c
      dimension x(2),a(0:nx,0:ny),diag(0:nx,0:ny),s2covar(ma,ma)
c
c evaluate function at x
c
      evalp2=0.0
      do i=nx,0,-1
         term=0.0
         do j=ny,0,-1
            term=a(i,j)+term*x(2)
         end do
         evalp2=term+evalp2*x(1)
      end do
c
c evaluate sigma at x
c
      sigx=sigp2(x,diag,s2covar,nx,ny)
      return
      end
```



```
      function SIGP2 (x,diag,s2covar,nx,ny)
c
c Returns standard deviation in fitted 2-d polynomial at x using a bette
c evaluation algorithm which relies on splitting off the diagonal values
c in the covariance matrix from the rest of the array.
c
c Parameters      Description
c ----------      -----------
c X               (2) array of coordinates
c DIAG            (0:NX,0:NY) diagonal sigmas
c S2COVAR         (MA,MA) where MA=(NX+1)*(NY+1)  covariance matrix
c                    after diagonal terms are removed
c NX,NY           Order of polynomial in each direction
c
      dimension x(2),diag(0:nx,0:ny),s2covar(0:nx,0:ny,0:nx,0:ny)
c
      sigp2=sqrt(varp2(x,diag,s2covar,nx,ny))
      return
      end
```