TgY Calibration

10/06/10

$$tgY(x_{Fp}, y_{Fp}, \vartheta_{Fp}, \varphi_{Fp}) = \underbrace{\sum_{ijkl} a_{ijkl}\, x_{Fp}^{i}\, y_{Fp}^{j}\, \vartheta_{Fp}^{k}\, \varphi_{Fp}^{l}}_{\text{Polynomial expansion}} =$$

$$= \left| \begin{array}{l} \text{We consider} \\ \text{35 matrix elements} \\ \text{for TgY} \end{array} \right| = \underbrace{a_{0000}\cdot 1 + a_{0001}\cdot\varphi + a_{0003}\cdot\varphi^3 + a_{0010}\cdot y + a_{0011}\cdot\varphi\cdot y + a_{0012}\cdot y\cdot\varphi^2 + \ldots}_{\text{This is linear in } a_{ijkl}}$$

We search for matrix element using following eq.:

event

$$tgY^{(1)}\Big|_{HRSL} = a_{0000}\cdot 1 + a_{0001}\cdot\varphi^{(1)} + a_{0003}\,\varphi^{3(1)} + \ldots$$

$$tgY^{(2)}\Big|_{HRSL} = a_{0000}\cdot 1 + a_{0001}\cdot\varphi^{(2)} + a_{0003}\,\varphi^{3(2)} + \ldots$$

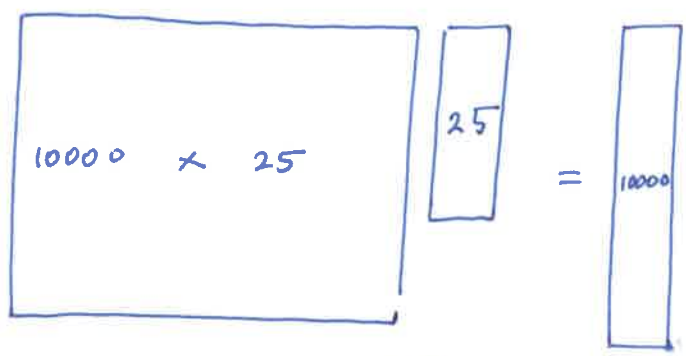
$$\vdots$$

(Number of considered events)

This set of equations can be rewritten in matrix form as:

$$\begin{pmatrix} 1 & \varphi^{(1)} & \varphi^{3(1)} & y^{(1)} & (\varphi y)^{(1)} & \cdots \\ 1 & \varphi^{(2)} & \varphi^{3(2)} & y^{(2)} & (\varphi y)^{(2)} & \cdots \\ & & \vdots & & & \end{pmatrix} \begin{pmatrix} a_{0000} \\ a_{0001} \\ a_{0003} \\ \vdots \end{pmatrix} = \begin{pmatrix} Tg\, y^{(1)} \\ Tg\, y^{(2)} \\ \vdots \end{pmatrix}$$

$$\Downarrow$$



$$\underline{\underline{A}} \cdot \underline{y} = \underline{b}$$

This is an over-determined system of equations. It can be solved using <u>SVD.</u>

2.6 Singular Value Decomposition

There exists a very powerful set of techniques for dealing with sets of equations or matrices that are either singular or else numerically very close to singular. In many cases where Gaussian elimination and *LU* decomposition fail to give satisfactory results, this set of techniques, known as *singular value decomposition*, or *SVD*, will diagnose for you precisely what the problem is. In some cases, SVD will not only diagnose the problem, it will also solve it, in the sense of giving you a useful numerical answer, although, as we shall see, not necessarily "the" answer that you thought you should get.

SVD is also the method of choice for solving most *linear least-squares* problems. We will outline the relevant theory in this section, but defer detailed discussion of the use of SVD in this application to Chapter 15, whose subject is the parametric modeling of data.

SVD methods are based on the following theorem of linear algebra, whose proof is beyond our scope: Any $M \times N$ matrix $\mathbf{A}$ whose number of rows M is greater than or equal to its number of columns N, can be written as the product of an $M \times N$ column-orthogonal matrix $\mathbf{U}$, an $N \times N$ diagonal matrix $\mathbf{W}$ with positive or zero elements (the *singular values*), and the transpose of an $N \times N$ orthogonal matrix $\mathbf{V}$. The various shapes of these matrices will be made clearer by the following tableau:

$$\begin{pmatrix} \\ \\ \mathbf{A} \\ \\ \\ \end{pmatrix} = \begin{pmatrix} \\ \\ \mathbf{U} \\ \\ \\ \end{pmatrix} \cdot \begin{pmatrix} w_1 & & & \\ & w_2 & & \\ & & \cdots & \\ & & \cdots & \\ & & & w_N \end{pmatrix} \cdot \begin{pmatrix} & \mathbf{V}^T & \end{pmatrix} \tag{2.6.1}$$

The matrices $\mathbf{U}$ and $\mathbf{V}$ are each orthogonal in the sense that their columns are orthonormal,

$$\sum_{i=1}^{M} U_{ik} U_{in} = \delta_{kn} \qquad \begin{array}{l} 1 \le k \le N \\ 1 \le n \le N \end{array} \tag{2.6.2}$$

$$\sum_{j=1}^{N} V_{jk} V_{jn} = \delta_{kn} \qquad \begin{array}{l} 1 \le k \le N \\ 1 \le n \le N \end{array} \tag{2.6.3}$$

or as a tableau,

$$
\left(\begin{array}{ccc} & & \\ & \mathbf{U}^T & \\ & & \end{array}\right) \cdot \left(\begin{array}{c} \\ \\ \mathbf{U} \\ \\ \\ \end{array}\right) = \left(\begin{array}{c} \\ \mathbf{V}^T \\ \\ \end{array}\right) \cdot \left(\begin{array}{c} \\ \mathbf{V} \\ \\ \end{array}\right)
$$

$$
= \left(\begin{array}{c} \\ \mathbf{1} \\ \\ \end{array}\right) \tag{2.6.4}
$$

Since $\mathbf{V}$ is square, it is also row-orthonormal, $\mathbf{V} \cdot \mathbf{V}^T = \mathbf{1}$.

The SVD decomposition can also be carried out when $M < N$. In this case the singular values w_j for $j = M+1, \ldots, N$ are all zero, and the corresponding columns of $\mathbf{U}$ are also zero. Equation (2.6.2) then holds only for $k, n \leq M$.

The decomposition (2.6.1) can always be done, no matter how singular the matrix is, and it is "almost" unique. That is to say, it is unique up to (i) making the same permutation of the columns of $\mathbf{U}$, elements of $\mathbf{W}$, and columns of $\mathbf{V}$ (or rows of $\mathbf{V}^T$), or (ii) forming linear combinations of any columns of $\mathbf{U}$ and $\mathbf{V}$ whose corresponding elements of $\mathbf{W}$ happen to be exactly equal. An important consequence of the permutation freedom is that for the case $M < N$, a numerical algorithm for the decomposition need not return zero w_j's for $j = M+1, \ldots, N$; the $N - M$ zero singular values can be scattered among all positions $j = 1, 2, \ldots, N$.

At the end of this section, we give a routine, `svdcmp`, that performs SVD on an arbitrary matrix $\mathbf{A}$, replacing it by $\mathbf{U}$ (they are the same shape) and giving back $\mathbf{W}$ and $\mathbf{V}$ separately. The routine `svdcmp` is based on a routine by Forsythe et al. [1], which is in turn based on the original routine of Golub and Reinsch, found, in various forms, in [2-4] and elsewhere. These references include extensive discussion of the algorithm used. As much as we dislike the use of black-box routines, we are going to ask you to accept this one, since it would take us too far afield to cover its necessary background material here. Suffice it to say that the algorithm is very stable, and that it is very unusual for it ever to misbehave. Most of the concepts that enter the algorithm (Householder reduction to bidiagonal form, diagonalization by QR procedure with shifts) will be discussed further in Chapter 11.

If you are as suspicious of black boxes as we are, you will want to verify yourself that `svdcmp` does what we say it does. That is very easy to do: Generate an arbitrary matrix $\mathbf{A}$, call the routine, and then verify by matrix multiplication that (2.6.1) and (2.6.4) are satisfied. Since these two equations are the only defining requirements for SVD, this procedure is (for the chosen $\mathbf{A}$) a complete end-to-end check.

Now let us find out what SVD is good for.

Solution by Use of Singular Value Decomposition

In some applications, the normal equations are perfectly adequate for linear least-squares problems. However, in many cases the normal equations are very close to singular. A zero pivot element may be encountered during the solution of the linear equations (e.g., in `gaussj`), in which case you get no solution at all. Or a very small pivot may occur, in which case you typically get fitted parameters a_k with very large magnitudes that are delicately (and unstably) balanced to cancel out almost precisely when the fitted function is evaluated.

Why does this commonly occur? The reason is that, more often than experimenters would like to admit, data do not clearly distinguish between two or more of the basis functions provided. If two such functions, or two different combinations of functions, happen to fit the data about equally well — or equally badly — then the matrix $[\alpha]$, unable to distinguish between them, neatly folds up its tent and becomes singular. There is a certain mathematical irony in the fact that least-squares problems are *both* overdetermined (number of data points greater than number of parameters) *and* underdetermined (ambiguous combinations of parameters exist); but that is how it frequently is. The ambiguities can be extremely hard to notice *a priori* in complicated problems.

Enter singular value decomposition (SVD). This would be a good time for you to review the material in §2.6, which we will not repeat here. In the case of an overdetermined system, SVD produces a solution that is the best approximation in the least-squares sense, cf. equation (2.6.10). That is exactly what we want. In the case of an underdetermined system, SVD produces a solution whose values (for us, the a_k's) are smallest in the least-squares sense, cf. equation (2.6.8). That is also what we want: When some combination of basis functions is irrelevant to the fit, that combination will be driven down to a small, innocuous, value, rather than pushed up to delicately canceling infinities.

In terms of the design matrix $\mathbf{A}$ (equation 15.4.4) and the vector $\mathbf{b}$ (equation 15.4.5), minimization of χ^2 in (15.4.3) can be written as

$$\text{find} \quad \mathbf{a} \quad \text{that minimizes} \quad \boxed{\chi^2 = |\mathbf{A} \cdot \mathbf{a} - \mathbf{b}|^2} \tag{15.4.16}$$

Comparing to equation (2.6.9), we see that this is precisely the problem that routines `svdcmp` and `svbksb` are designed to solve. The solution, which is given by equation (2.6.12), can be rewritten as follows: If $\mathbf{U}$ and $\mathbf{V}$ enter the SVD decomposition of $\mathbf{A}$ according to equation (2.6.1), as computed by `svdcmp`, then let the vectors $\mathbf{U}_{(i)}$ $i = 1, \ldots, M$ denote the *columns* of $\mathbf{U}$ (each one a vector of length N); and let the vectors $\mathbf{V}_{(i)}; i = 1, \ldots, M$ denote the *columns* of $\mathbf{V}$ (each one a vector of length M). Then the solution (2.6.12) of the least-squares problem (15.4.16) can be written as

$$\mathbf{a} = \sum_{i=1}^{M} \left(\frac{\mathbf{U}_{(i)} \cdot \mathbf{b}}{w_i} \right) \mathbf{V}_{(i)} \tag{15.4.17}$$

where the w_i are, as in §2.6, the singular values calculated by `svdcmp`.

Equation (15.4.17) says that the fitted parameters $\mathbf{a}$ are linear combinations of the columns of $\mathbf{V}$, with coefficients obtained by forming dot products of the columns of U with the weighted data vector (15.4.5). Though it is beyond our scope to prove here, it turns out that the standard (loosely, "probable") errors in the fitted parameters are also linear combinations of the columns of V. In fact, equation (15.4.17) can be written in a form displaying these errors as

$$\boxed{\mathbf{a} = \left[\sum_{i=1}^{M} \left(\frac{\mathbf{U}_{(i)} \cdot \mathbf{b}}{w_i}\right) \mathbf{V}_{(i)}\right] \pm \frac{1}{w_1}\mathbf{V}_{(1)} \pm \cdots \pm \frac{1}{w_M}\mathbf{V}_{(M)}} \tag{15.4.18}$$

Here each $\pm$ is followed by a standard deviation. The amazing fact is that, decomposed in this fashion, the standard deviations are all mutually independent (uncorrelated). Therefore they can be added together in root-mean-square fashion. What is going on is that the vectors $\mathbf{V}_{(i)}$ are the principal axes of the error ellipsoid of the fitted parameters $\mathbf{a}$ (see §15.6).

It follows that the variance in the estimate of a parameter a_j is given by

$$\sigma^2(a_j) = \sum_{i=1}^{M} \frac{1}{w_i^2}[\mathbf{V}_{(i)}]_j^2 = \sum_{i=1}^{M} \left(\frac{V_{ji}}{w_i}\right)^2 \tag{15.4.19}$$

whose result should be identical with (15.4.14). As before, you should not be surprised at the formula for the covariances, here given without proof,

$$\mathrm{Cov}(a_j, a_k) = \sum_{i=1}^{M} \left(\frac{V_{ji}V_{ki}}{w_i^2}\right) \tag{15.4.20}$$

We introduced this subsection by noting that the normal equations can fail by encountering a zero pivot. We have not yet, however, mentioned how SVD overcomes this problem. The answer is: If any singular value w_i is zero, its reciprocal in equation (15.4.18) should be set to zero, not infinity. (Compare the discussion preceding equation 2.6.7.) This corresponds to adding to the fitted parameters $\mathbf{a}$ a *zero* multiple, rather than some random large multiple, of any linear combination of basis functions that are degenerate in the fit. It is a good thing to do!

Moreover, if a singular value w_i is nonzero but very small, you should also define *its* reciprocal to be zero, since its apparent value is probably an artifact of roundoff error, not a meaningful number. A plausible answer to the question "how small is small?" is to edit in this fashion all singular values whose ratio to the largest singular value is less than N times the machine precision ϵ. (You might argue for $\sqrt{N}$, or a constant, instead of N as the multiple; that starts getting into hardware-dependent questions.)

There is another reason for editing even *additional* singular values, ones large enough that roundoff error is not a question. Singular value decomposition allows you to identify linear combinations of variables that just happen not to contribute much to reducing the χ^2 of your data set. Editing these can sometimes reduce the probable error on your coefficients quite significantly, while increasing the minimum χ^2 only negligibly. We will learn more about identifying and treating such cases in §15.6. In the following routine, the point at which this kind of editing would occur is indicated.

Generally speaking, we recommend that you always use SVD techniques instead of using the normal equations. SVD's only significant disadvantage is that it requires

(6)

an extra array of size $N \times M$ to store the whole design matrix. This storage is overwritten by the matrix **U**. Storage is also required for the $M \times M$ matrix **V**, but this is instead of the same-sized matrix for the coefficients of the normal equations. SVD can be significantly slower than solving the normal equations; however, its great advantage, that it (theoretically) *cannot fail*, more than makes up for the speed disadvantage.

In the routine that follows, the matrices u,v and the vector w are input as working space. The logical dimensions of the problem are ndata data points by ma basis functions (and fitted parameters). If you care only about the values a of the fitted parameters, then u,v,w contain no useful information on output. If you want probable errors for the fitted parameters, read on.

```
#include "nrutil.h"
#define TOL 1.0e-5

void svdfit(float x[], float y[], float sig[], int ndata, float a[], int ma,
	float **u, float **v, float w[], float *chisq,
	void (*funcs)(float, float [], int))
```

Given a set of data points x[1..ndata],y[1..ndata] with individual standard deviations sig[1..ndata], use χ^2 minimization to determine the coefficients a[1..ma] of the fitting function $y = \sum_i a_i \times \text{afunc}_i(x)$. Here we solve the fitting equations using singular value decomposition of the ndata by ma matrix, as in §2.6. Arrays u[1..ndata][1..ma], v[1..ma][1..ma], and w[1..ma] provide workspace on input; on output they define the singular value decomposition, and can be used to obtain the covariance matrix. The program returns values for the ma fit parameters a, and χ^2, chisq. The user supplies a routine funcs(x,afunc,ma) that returns the ma basis functions evaluated at $x =$ x in the array afunc[1..ma].

```
{
	void svbksb(float **u, float w[], float **v, int m, int n, float b[],
		float x[]);
	void svdcmp(float **a, int m, int n, float w[], float **v);
	int j,i;
	float wmax,tmp,thresh,sum,*b,*afunc;

	b=vector(1,ndata);
	afunc=vector(1,ma);
	for (i=1;i<=ndata;i++) {                Accumulate coefficients of the fitting ma-
		(*funcs)(x[i],afunc,ma);                trix.
		tmp=1.0/sig[i];
		for (j=1;j<=ma;j++) u[i][j]=afunc[j]*tmp;
		b[i]=y[i]*tmp;
	}
	svdcmp(u,ndata,ma,w,v);                 Singular value decomposition.
	wmax=0.0;                               Edit the singular values, given TOL from the
	for (j=1;j<=ma;j++)                         #define statement, between here ...
		if (w[j] > wmax) wmax=w[j];
	thresh=TOL*wmax;
	for (j=1;j<=ma;j++)
		if (w[j] < thresh) w[j]=0.0;        ...and here.
	svbksb(u,w,v,ndata,ma,b,a);
	*chisq=0.0;                             Evaluate chi-square.
	for (i=1;i<=ndata;i++) {
		(*funcs)(x[i],afunc,ma);
		for (sum=0.0,j=1;j<=ma;j++) sum += a[j]*afunc[j];
		*chisq += (tmp=(y[i]-sum)/sig[i],tmp*tmp);
	}
	free_vector(afunc,1,ma);
	free_vector(b,1,ndata);
}
```