Lecture 33
Modeling of data

- General linear least squares.
- Nonlinear models.

General linear least squares

Linear combinations of functions (e.g. powers, sines and cosines) $$y(x) = \sum_{k=1}^{M} a_k X_k(x)$$

We generalize the merit function we discussed last class, with σ_i the error (standard deviation) of the ith data point. If the errors are not known, we set the sigmas to 1.

$$\chi^2 = \sum_{i=1}^{N} \left[\frac{y_i - \sum_{k=1}^{M} a_k X_k(x_i)}{\sigma_i} \right]^2$$

We define the *design matrix* of the fitting problem, which in general has more rows than columns and a vector **b** of length N, and denote the quantities to be fitted by **a**,

$$A_{ij} = \frac{X_j(x_i)}{\sigma_i}, \qquad b_i = \frac{y_i}{\sigma_i}$$

The minimum of the merit functions occurs when the all the derivatives with respect to the parameters vanish, leading to the equations

$$0 = \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \left[y_i - \sum_{j=1}^{M} a_j X_j(x_i) \right] X_k(x_i) \qquad k = 1, \ldots, M$$

Inverting the order of the sums leads to, $\sum_{j=1}^{M} \alpha_{kj} a_j = \beta_k$

where

$$\alpha_{kj} = \sum_{i=1}^{N} \frac{X_j(x_i) X_k(x_i)}{\sigma_i^2} \qquad \text{or equivalently} \qquad [\alpha] = \mathbf{A}^T \cdot \mathbf{A}$$

an $M \times M$ matrix, and

$$\beta_k = \sum_{i=1}^{N} \frac{y_i X_k(x_i)}{\sigma_i^2} \qquad \text{or equivalently} \qquad [\beta] = \mathbf{A}^T \cdot \mathbf{b}$$

So in matrix form they can be written as $\left(A^T \bullet A\right) \bullet a = A^T \bullet b$

And can be solved by LU decomposition and backsubstitution, Cholesky decomposition or Gauss-Jordan elimination.

To estimate the uncertainties in the parameters, as before we consider the variance of a function,

$$\sigma^2(a_j) = \sum_{i=1}^{N} \sigma_i^2 \left(\frac{\partial a_j}{\partial y_i} \right)^2$$

Note that α_{jk} is independent of y_i, so that

$$\frac{\partial a_j}{\partial y_i} = \sum_{k=1}^{M} C_{jk} X_k(x_i)/\sigma_i^2$$

Consequently, we find that

Inverse of C

$$\sigma^2(a_j) = \sum_{k=1}^{M} \sum_{l=1}^{M} C_{jk} C_{jl} \left[\sum_{i=1}^{N} \frac{X_k(x_i) X_l(x_i)}{\sigma_i^2} \right]$$

Therefore $\sigma^2(a_j) = C_{jj}$

So the diagonal elements of the matrix C are a measure of the error in the determined coefficients.

For some applications, solving the normal equations as we just discussed works well enough. However, in many cases the equations turn out to be close to singular. The reason is that more often than not the data is not good enough to distinguish two or more of the functions of the basis chosen. If two of the functions fit the data well, the method folds up and the matrix becomes singular. We are in the peculiar situation of having a linear problem that is on the one hand overdetermined (more equations than variables to determine) and underdetermined.

When problems of this sort arise, the method of choice is the Singular Value Decomposition that we discussed a few classes back.

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

The solution to the normal problem can then be written as, with $U_{(i)}$ and $V_{(i)}$ the columns of U and V respectively.

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

The variance of the estimate of the parameters is proportional to the V's,

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

So how does this help if the system is singular? If one of the eigenvalues wi turns out to be zero, its inverse in the top equation should be set to zero rather than infinity. That is, we exclude the corresponding function of the basis from the fitting. One does the same if a given eigenvalue is small. How small? A rule of thumb is that it is smaller than the largest eigenvalue by N times the machine precision. There is even the case for eliminating functions of the basis whose corresponding eigenvalues are considerably larger than this, as it usually improves the fitting of the remaining coefficients.

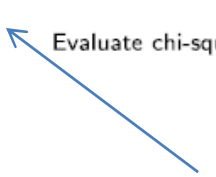
```
      SUBROUTINE svdfit(x,y,sig,ndata,a,ma,u,v,w,mp,np,
     *     chisq,funcs)
      INTEGER ma,mp,ndata,np,NMAX,MMAX
      REAL chisq,a(ma),sig(ndata),u(mp,np),v(np,np),w(np),
     *     x(ndata),y(ndata),TOL
      EXTERNAL funcs
      PARAMETER (NMAX=1000,MMAX=50,TOL=1.e-5)
```

NMAX is the maximum expected value of ndata; MMAX the maximum expected for ma; the default TOL value is appropriate for single precision and variables scaled to be of order unity.

```
C     USES svbksb,svdcmp
```

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 ma coefficients a of the fitting function $y = \sum_i \mathtt{a}_i \times \mathtt{afunc}_i(x)$. Here we solve the fitting equations using singular value decomposition of the ndata by ma matrix, as in §2.6. Arrays u(1:mp,1:np),v(1:np,1:np), w(1:np) provide workspace on input; on output they define the singular value decomposition, and can be used to obtain the covariance matrix. mp,np are the physical dimensions of the matrices u,v,w, as indicated above. It is necessary that mp≥ndata, np ≥ ma. The program returns values for the ma fit parameters a, and χ^2, chisq. The user supplies a subroutine funcs(x,afunc,ma) that returns the ma basis functions evaluated at $x =$ x in the array afunc.

```
      INTEGER i,j
      REAL sum,thresh,tmp,wmax,afunc(MMAX),b(NMAX)
      do 12 i=1,ndata                          Accumulate coefficients of the fitting ma-
          call funcs(x(i),afunc,ma)                trix.
          tmp=1./sig(i)
          do 11 j=1,ma
              u(i,j)=afunc(j)*tmp
          enddo 11
          b(i)=y(i)*tmp
      enddo 12
      call svdcmp(u,ndata,ma,mp,np,w,v)        Singular value decomposition.
      wmax=0.                                  Edit the singular values, given TOL from the
      do 13 j=1,ma                                 parameter statement, between here ...
          if(w(j).gt.wmax)wmax=w(j)
      enddo 13
      thresh=TOL*wmax
      do 14 j=1,ma
          if(w(j).lt.thresh)w(j)=0.
      enddo 14                                 ...and here.
      call svbksb(u,w,v,ndata,ma,mp,np,b,a)
      chisq=0.                                 Evaluate chi-square.
      do 16 i=1,ndata
          call funcs(x(i),afunc,ma)
          sum=0.
          do 15 j=1,ma
              sum=sum+a(j)*afunc(j)
          enddo 15
          chisq=chisq+((y(i)-sum)/sig(i))**2
      enddo 16
      return
      END
```



Back-substitution routine knows how to take care of zero w(i)’s.

Particular cases

$$y(x) = a\exp(-bx)$$

Some problems can be treated as linear via a substitution

$$\log[y(x)] = c - bx$$

A common mistake is to fit with "non-parameters"

$$y(x) = a\exp(-bx + d)$$

In this example a and d are indistinguishable and will lead singular normal equations.

Multidimensional fits correspond to measuring a single variable y as a function of more than one variable. The merit function in that case is,

$$\chi^2 = \sum_{i=1}^{N} \left[\frac{y_i - \sum_{k=1}^{M} a_k X_k(\mathbf{x}_i)}{\sigma_i} \right]^2$$

And the previous discussion goes mostly unchanged. In fact in both lfig and svdfit one can apply a hack setting x(i)=i and providing the subroutine funcs with the true vector values of your data points (say via a COMMON block in Fortran) then funcs will translate the fictitious x(i)' s to the actual data points before doing its work.

Nonlinear models

We now turn our attention to fitting when the model depends nonlinearly on the set of M parameters ak. The approach will be similar to before, defining a merit function. With nonlinear functions, the minimization must proceed iteratively. Given trial values we develop a procedure that improves the trial solution. The procedure is repeated until the function of merit effectively stops decreasing.

Close to the minimum, we can expand chi in Taylor series,

$$\chi^2(a) = \chi^2(a_{\min}) + \nabla\chi^2(a_{\min}) \bullet (a - a_{\min}) + \frac{1}{2}(a - a_{\min}) \bullet A \bullet (a - a_{\min})$$

Computing the gradient and setting it to zero,

$$\nabla\chi^2(a) = A \bullet (a - a_{\min}) \quad \text{so} \quad a_{\min} - a = -A^{-1} \bullet \nabla\chi^2(a)$$

So computationally we need the gradient of the merit function and its Hessian (matrix of second derivatives). Since the merit function is known analytically we have them in closed form.

The model to be fitted is

$$y = y(x; \mathbf{a})$$

and the χ^2 merit function is

$$\chi^2(\mathbf{a}) = \sum_{i=1}^{N} \left[\frac{y_i - y(x_i; \mathbf{a})}{\sigma_i} \right]^2$$

Gradient and Hessian,

$$\frac{\partial \chi^2}{\partial a_k} = -2 \sum_{i=1}^{N} \frac{[y_i - y(x_i; \mathbf{a})]}{\sigma_i^2} \frac{\partial y(x_i; \mathbf{a})}{\partial a_k} \qquad k = 1, 2, \ldots, M$$

$$\frac{\partial^2 \chi^2}{\partial a_k \partial a_l} = 2 \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \left[\frac{\partial y(x_i; \mathbf{a})}{\partial a_k} \frac{\partial y(x_i; \mathbf{a})}{\partial a_l} - [y_i - y(x_i; \mathbf{a})] \frac{\partial^2 y(x_i; \mathbf{a})}{\partial a_l \partial a_k} \right]$$

$$\beta_k \equiv -\frac{1}{2} \frac{\partial \chi^2}{\partial a_k} \qquad \alpha_{kl} \equiv \frac{1}{2} \frac{\partial^2 \chi^2}{\partial a_k \partial a_l} \qquad \sum_{l=1}^{M} \alpha_{kl} \, \delta a_l = \beta_k$$

This is all very neat, but the assumption that a quadratic function approximates well the merit function near its minimum is a BIG assumption. What if that is not the case?

In that case the best course of action is to use the "*steepest descent formula*", noting that the gradient will generically point you towards the minimum of the function and proposing

$$\delta a_l = \text{constant} \times \beta_l$$

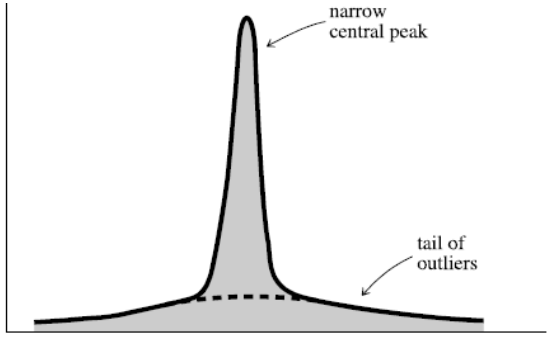
Now the question is, what is the value of the constant? A rule of thumb is that it should be "small" in the sense that it does not shoot you past the root. Using nothing more than dimensional analysis, Marquardt, based on a suggestion of Levenberg proposes using the diagonal coefficients of the matrix α_{kl} times a fudge factor l>>1 to cut down the step and avoid overstepping the minimum.

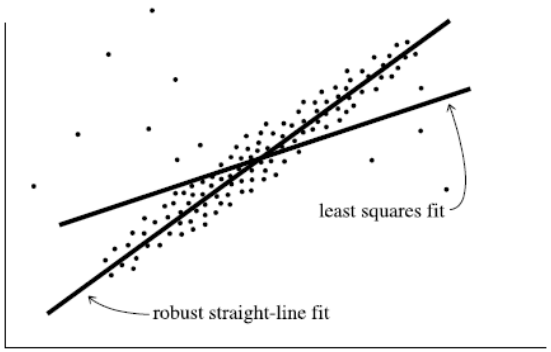
One needs a criterion to stop the iteration. It should be kept in mind that even if we were to find the exact minimum, the resulting values of a's are at best a statistical estimate, so it is not really helpful to insist on too much accuracy in finding the minimum.

It turns out the Levenberg-Marquardt method works very well and is the one implemented in numerical recipes and in the package MINPACK.

Robust estimation

The concept of robustness was coined in statistics by G.E.P. Box in 1953. Various definitions of varying levels of rigor can be offered for the term, but in referring to a statistical estimator it means "insensitive to small departures from the idealized assumptions". The Word "small" has two interpretations here: either fractionally small departures for all variables or large departures for a small number of variables (outliers). Robustness in the face of the latter is what is most sought after, since outliers are what creates most stress to statistical methods.





Most robust methods can be grouped into three types:

-M-estimates follow maximum likelihood arguments similar to the ones we considered, just using distributions that are more realistic, with actual tails.

-L-estimatse are "linear combinations of order statistics". Examples are the median and Tukey's trimean, which is the weighted average of the first, second and third quartile points in a distribution, with weights ¼, ½ and 1/4.

-R-estimates are based on rank tests. The Kolmogorov-Smirnov and Spearman rank order tests are examples.

For M estimation we repeat the same story as before, with better distributions, i.e.

$$P = \prod^{N} \{\exp\left[-\rho(y_i, y\left\{x_i; \mathbf{a}\right\})\right] \Delta y\}$$

So we minimize $$\sum_{i=1}^{N} \rho(y_i, y\left\{x_i; \mathbf{a}\right\})$$

Sometimes the dependence on the measured and predicted values is through differences,

$$\sum_{i=1}^{N} \rho(y_i, y\left\{x_i; \mathbf{a}\right\})$$

Defining $$z \equiv [y_i - y(x_i)]/\sigma_i. \qquad \psi(z) \equiv \frac{d\rho(z)}{dz}$$

The problem to be solved is,

$$0 = \sum_{i=1}^{N} \frac{1}{\sigma_i} \psi\left(\frac{y_i - y(x_i)}{\sigma_i}\right)\left(\frac{\partial y(x_i; \mathbf{a})}{\partial a_k}\right)$$

An example of a distribution with fatter tails is the two sided exponential,

$$\text{Prob } \{y_i - y(x_i)\} \sim \exp\left(-\left|\frac{y_i - y(x_i)}{\sigma_i}\right|\right)$$

then, by contrast,

$$\rho(x) = |z| \qquad \psi(z) = \text{sgn}(z) \qquad \text{(double exponential)}$$

Compare to normal:

$$\rho(z) = \frac{1}{2}z^2 \qquad \psi(z) = z \qquad \text{(normal)}$$

Fitting a line by minimizing the absolute deviation

Say the model is a straight line, $y(x; a, b) = a + bx$

And our merit function with fat tails to be minimized is, $\sum_{i=1}^{N} |y_i - a - bx_i|$

We now use a trick: we note that the median is the value that minimizes the sum of the absolute deviations,

$$\sum_i |c_i - c_M|$$

Therefore, for fixed b, the value of a that minimizes the merit functions is simply given by,

$$a = \text{median}\,\{y_i - bx_i\}$$

$$0 = \sum_{i=1}^{N} x_i \,\text{sgn}(y_i - a - bx_i)$$

And this can be solved by bracketing and bisection.

```
      SUBROUTINE medfit(x,y,ndata,a,b,abdev)
      INTEGER ndata,NMAX,ndatat
      PARAMETER (NMAX=1000)
      REAL a,abdev,b,x(ndata),y(ndata),
     *     arr(NMAX),xt(NMAX),yt(NMAX),aa,abdevt
      COMMON /arrays/ xt,yt,arr,aa,abdevt,ndatat
```

USES rofunc

Fits $y = a + bx$ by the criterion of least absolute deviations. The arrays `x(1:ndata)` and `y(1:ndata)` are the input experimental points. The fitted parameters `a` and `b` are output, along with `abdev`, which is the mean absolute deviation (in y) of the experimental points from the fitted line. This routine uses the routine `rofunc`, with communication via a common block.

```
      INTEGER j
      REAL b1,b2,bb,chisq,del,f,f1,f2,sigb,sx,sxx,sxy,sy,rofunc
      sx=0.
      sy=0.
      sxy=0.
      sxx=0.
      do 11 j=1,ndata                          As a first guess for a and b, we will find the least-
        xt(j)=x(j)                               squares fitting line.
        yt(j)=y(j)
        sx=sx+x(j)
        sy=sy+y(j)
        sxy=sxy+x(j)*y(j)
        sxx=sxx+x(j)**2
      enddo 11
      ndatat=ndata
      del=ndata*sxx-sx**2
      aa=(sxx*sy-sx*sxy)/del                   Least-squares solutions.
      bb=(ndata*sxy-sx*sy)/del
      chisq=0.
      do 12 j=1,ndata
        chisq=chisq+(y(j)-(aa+bb*x(j)))**2
      enddo 12
      sigb=sqrt(chisq/del)                     The standard deviation will give some idea of how
      b1=bb                                      big an iteration step to take.
```

```
      f1=rofunc(b1)
      if(sigb.gt.0.)then
          b2=bb+sign(3.*sigb,f1)          Guess bracket as 3-σ away, in the downhill direction
          f2=rofunc(b2)                       known from f1.
          if(b2.eq.b1)then
              a=aa
              b=bb
              abdev=abdevt/ndata
              return
          endif
1         if(f1*f2.gt.0.)then             Bracketing.
              bb=b2+1.6*(b2-b1)
              b1=b2
              f1=f2
              b2=bb
              f2=rofunc(b2)
              goto 1
          endif
          sigb=0.01*sigb                  Refine until error a negligible number of standard de-
2         if(abs(b2-b1).gt.sigb)then          viations.
              bb=b1+0.5*(b2-b1)           Bisection.
              if(bb.eq.b1.or.bb.eq.b2)goto 3
              f=rofunc(bb)
              if(f*f1.ge.0.)then
                  f1=f
                  b1=bb
              else
                  f2=f
                  b2=bb
              endif
              goto 2
          endif

      endif
3     a=aa
      b=bb
      abdev=abdevt/ndata
      return
      END
```

```
      FUNCTION rofunc(b)
      INTEGER NMAX
      REAL rofunc,b,EPS
      PARAMETER (NMAX=1000,EPS=1.e-7)
C     USES select
```

Evaluates the right-hand side of equation (15.7.16) for a given value of b. Communication with the program `medfit` is through a common block.

```
      INTEGER j,ndata
      REAL aa,abdev,d,sum,arr(NMAX),x(NMAX),y(NMAX),select
      COMMON /arrays/ x,y,arr,aa,abdev,ndata
      do 11 j=1,ndata
        arr(j)=y(j)-b*x(j)
      enddo 11
      if (mod(ndata,2).eq.0) then
        j=ndata/2
        aa=0.5*(select(j,ndata,arr)+select(j+1,ndata,arr))
      else
        aa=select((ndata+1)/2,ndata,arr)
      endif
      sum=0.
      abdev=0.
      do 12 j=1,ndata
        d=y(j)-(b*x(j)+aa)
        abdev=abdev+abs(d)
        if (y(j).ne.0.) d=d/abs(y(j))
        if (abs(d).gt.EPS) sum=sum+x(j)*sign(1.0,d)
      enddo 12
      rofunc=sum
      return
      END
```

Summary:

- Generalizations of least squares to combinations of functions work more or less like the linear case, but may lead to singular systems of linear equations.
- Robust estimates have low sensitivity to departures from the underlying assumptions.