

Least Squares Approximation

James D Emery

5/8/95, (Edited 11/1/06)

Contents

0.1	Elementary Formulation	1
0.2	A Geometric View of the Least Squares Problem	2
0.3	Linear Circle Fitting	9
0.4	Nonlinear Circle Fitting	12
0.5	Weighted Least Squares	16
0.6	Bilinear Interpolation Example	17
0.7	Fitting Functions With Horizontal Asymptotes, Using the general program lsqgen	17
0.8	A General LSQ Program lsqgen.ftn	19
0.9	Listing of lsqgen.ftn	19

0.1 Elementary Formulation

The traditional way of deriving least squares equations is to write the expression for the sum of the squares difference between the given "data" and the approximating function, and then to set the partial derivatives with respect to the coefficients of the approximating function to zero. Let us do this for the case of fitting a straight line to given data. Assume the model $f(x) = ax + b$ and minimize

$$r(a, b) = \sum_{i=1}^n (ax_i + b - y_i)^2$$

The conditions for a minimum are

$$\frac{\partial r}{\partial a} = \sum_{i=1}^n 2x_i(ax_i + b - y_i) = 0$$

$$\frac{\partial r}{\partial b} = \sum_{i=1}^n 2(ax_i + b - y_i) = 0$$

We get a two by two system of equations.

$$a \sum_{i=1}^n x_i^2 + b \sum_{i=1}^n x_i = \sum_{i=1}^n x_i y_i$$
$$a \sum_{i=1}^n x_i + b \sum_{i=1}^n 1 = \sum_{i=1}^n y_i$$

These equations are known as the normal equations of the problem. They have a unique solution if the determinant is not zero, that is if

$$n \sum_{i=1}^n x_i^2 - \left(\sum_{i=1}^n x_i \right)^2 \neq 0.$$

If the x values are not all equal this follows from the Cauchy-Schwartz inequality applied to the vectors $(1, 1, \dots, 1)$ and (x_1, x_2, \dots, x_n) . The general problem can be viewed more naturally as being geometric.

0.2 A Geometric View of the Least Squares Problem

The abstract linear least squares problem may be formulated as approximation in a vector space by some element of a subspace. Often this vector space is a space of functions. As examples the subspace could be generated by a bases such as

$$1, x, x^2, x^3, \dots,$$

or such as

$$1, \cos(\omega t), \sin(\omega t), \cos(2\omega t), \sin(\omega t), \dots$$

The first case would be a polynomial, or power series approximation. And the second would be a Fourier or trigonometric approximation. So consider a vector space V with an inner product of u , with v , written as (u, v) . Given a subspace S and an arbitrary element g of V , we are to find the element in S that best approximates g in the norm corresponding to the inner product. The L^2 norm for functions is based on the inner product

$$(f, g) = \int fg,$$

and for sequences is based on the inner product

$$(f, g) = \sum_{i=1}^n f_i g_i.$$

This L^2 norm corresponds directly to the "squares" part of the least squares approximation. But the theory carries through for an arbitrary inner product. The norm defined by an inner product is

$$\|f\| = (f, f)^{1/2}.$$

A solution $f \in S$, minimizes

$$(f - g, f - g) = \|f - g\|^2.$$

We will show that the problem is solved as the orthogonal projection of a vector into a subspace. One can think of this as analogous to the simple geometric problem of projecting a vector in space onto a plane. Think of a vector from the origin to a point, and think of a plane through the origin, not containing this vector. The plane is a vector space. A vector in the plane closest to the original vector is obviously the orthogonal projection of the vector onto the plane. The same thing happens in the general problem, where the plane becomes the subspace. For example the subspace might be the set of all cubic polynomials. And the problem is to best fit the data to a cubic polynomial.

Two vectors are orthogonal, i.e. perpendicular, if their inner product is zero. We require a preliminary theorem to prove the main proposition.

Pythagorean Theorem. If v_1 is orthogonal to v_2 , then

$$\|v_1 + v_2\|^2 = \|v_1\|^2 + \|v_2\|^2.$$

Proof.

$$(v_1 + v_2, v_1 + v_2) = (v_1, v_1) + 2(v_1, v_2) + (v_2, v_2) = (v_1, v_1) + (v_2, v_2).$$

Proposition. If $f \in S$ and $(g - f, h) = 0, \forall h \in S$ then f is a solution to the least squares problem.

Proof. Let $s \in S$. We have

$$\|g - s\|^2 = \|(g - f) + (f - s)\|^2 = \|g - f\|^2 + \|f - s\|^2 \geq \|g - f\|^2.$$

By assumption, $g - f$ is orthogonal to the subspace S , and $f - s$ is in S . So the second equality is a consequence of the Pythagorean Theorem. We have shown that

$$\|g - s\| \geq \|g - f\|, \forall s \in S.$$

so f is the best approximation to g in S and this completes the proof.

Notice that a unique solution always exists because f is the unique orthogonal projection of g into S . For finite subspaces the solution can be formulated as a solution to a set of n linear equations in n unknowns. Let S equal the span of f_1, \dots, f_n . Let the solution be

$$f = c_1 f_1 + c_2 f_2 + \dots + c_n f_n.$$

Then the minimum condition is equivalent to

$$(f_i, c_1 f_1 + c_2 f_2 + \dots + c_n f_n - g) = 0, i = 1, \dots, n.$$

This is the same as

$$c_1 (f_i, f_1) + c_2 (f_i, f_2) + \dots + c_n (f_i, f_n) = (f_i, g), i = 1, \dots, n.$$

These n linear equations in n unknowns are called the normal equations of the problem. In the usual case, S is a space of discrete functions. These are functions defined on a finite domain. Suppose there are m data values so that the domain is

$$\{p_1, p_2, \dots, p_m\}.$$

We identify the function f_i with the vector

$$\begin{bmatrix} f_i(p_1) \\ f_i(p_2) \\ \dots \\ \dots \\ f_i(p_m) \end{bmatrix}$$

f_i is an m dimensional column vector of values of the i th function. We can formulate the minimum conditions with matrices. The inner product is then the transpose of the first vector times the second. We write the transpose of a vector v as v^t . We have $(f_i, f_j) = f_i^t f_j$ Then

$$c_1 (f_i, f_1) + c_2 (f_i, f_2) + \dots + c_n (f_i, f_n) = (f_i, g), i = 1, \dots, n.$$

Thus

$$\begin{bmatrix} f_i^t f_1 & \dots & f_i^t f_n \end{bmatrix} \begin{bmatrix} c_1 \\ \cdot \\ \cdot \\ \cdot \\ c_n \end{bmatrix} = f_i^t g$$

If we let A be an m row by n column matrix, whose i th column is f_i , then

$$A = \begin{bmatrix} f_1 & f_2 & \dots & f_n \end{bmatrix}$$

Written out

$$A = \begin{bmatrix} f_1(p_1) & f_2(p_1) & \dots & f_n(p_1) \\ f_1(p_2) & f_2(p_2) & \dots & f_n(p_2) \\ \dots & \dots & \dots & \dots \\ f_1(p_m) & f_2(p_m) & \dots & f_n(p_m) \end{bmatrix}$$

Also let

$$B = \begin{bmatrix} g(p_1) \\ \cdot \\ \cdot \\ \cdot \\ g(p_m) \end{bmatrix}$$

The normal equations become

$$A^t A \begin{bmatrix} c_1 \\ \cdot \\ \cdot \\ \cdot \\ c_n \end{bmatrix} = A^t B.$$

Note that the original approximation problem in this form is a system of m equations in n unknowns

$$A \begin{bmatrix} c_1 \\ \cdot \\ \cdot \\ \cdot \\ c_n \end{bmatrix} \approx B.$$

Any linear system of this form with $m > n$ can be interpreted as a least squares problem and has an approximate least squares solution. The matrices

A and B are a convenient input set to a general linear least squares solver (see the listing of subroutine `llsq`).

There is always a unique solution to the linear least squares problem. The solution is the orthogonal projection into the subspace. But there will be more than one solution to the normal equations if the given functions spanning the subspace are not linearly independent. The normal equations have a solution, so they are consistent. From the theory of linear equations, if the determinant D of the coefficient matrix of the normal equations is not zero, then there is a unique solution. Then we can solve the equations either by inverting the coefficient matrix, or by gaussian elimination. If D is zero, then there is more than one solution, such solution will involve one or more variables of arbitrary value. Gaussian elimination will fail. The $D = 0$ solution can be computed by using elementary row operations which can be done numerically or with various computer algebra programs. When we are concerned only with the discrete space, it does not matter that there are multiple solutions to the normal equations. Because any set of coefficients gives a linear combination equal to the unique projection into the subspace. The various solutions just give different linear combinations of dependent vectors that equal the same vector. On the other hand if points other than the sample points are in the relevant domain of the functions, then the multiple solutions may give function solutions that are not the same on this extended domain. To illustrate compare functions f and g where $f(x) = x(x - 1)$ is equal to zero on the domain $x = 0$ and $x = 1$, but it is not zero on the extended domain of all real numbers. Let g be the true zero function, $g(x) = 0$. The two functions agree on $\{0, 1\}$, but give different values on an extended domain. Frequently we want to use the least squares solution for interpolation between the given data points, and so the case of multiple solutions to the normal equations does have consequence.

We will show that if f_1, \dots, f_n are linearly independent then the normal equations have a unique solution. This is obvious because in this case f_1, \dots, f_n is a basis of S and the unique solution f in S has unique components with respect to this basis. It is also a direct consequence of the following proposition.

Proposition. if f_1, \dots, f_n are the linearly independent columns of a matrix A , which has $m > n$ rows, then $\det(A^t A)$ is not equal to zero.

Proof. Suppose the determinant is zero. Then there exists c_1, c_2, \dots, c_n , not

all zero such that

$$c_1 \begin{bmatrix} (f_1, f_1) \\ (f_2, f_1) \\ \dots \\ \dots \\ (f_n, f_1) \end{bmatrix} + c_2 \begin{bmatrix} (f_1, f_2) \\ (f_2, f_2) \\ \dots \\ \dots \\ (f_n, f_2) \end{bmatrix} + \dots + c_n \begin{bmatrix} (f_1, f_n) \\ (f_2, f_n) \\ \dots \\ \dots \\ (f_n, f_n) \end{bmatrix} = 0.$$

Let

$$v = c_1 f_1 + \dots + c_n f_n.$$

The first equation shows that $(f_i, v) = 0$, for $i = 1, \dots, n$. It follows that $(v, v) = 0$. This implies $v = 0$, and so each c_i is zero. This is a contradiction, so the proposition is true.

Example 1. We are to fit the function

$$y = f(x) = a \sin(x) + b \cos(x).$$

to the data

$$\begin{array}{cc} x & y \\ 1.0 & 3.0 \\ 2.5 & 5.6 \\ 3.4 & 7.8 \end{array} ,$$

Apply the sin function to the x values to get the first column of matrix A and the cos function to get the second column. Let vector B be the y values. The normal equations are

$$A^t A C = A^t B$$

or in terms of the components

$$\begin{bmatrix} 1.13154358 & 0.22224325 \\ 0.22224325 & 1.86845642 \end{bmatrix} C = \begin{bmatrix} 3.882636366 \\ -10.40652323 \end{bmatrix}$$

The solution is

$$C = \begin{bmatrix} 4.6334245 \\ -6.120705005 \end{bmatrix}$$

So

$$f(x) = 4.6334245 \sin(x) - 6.120705005 \cos(x)$$

The following program does the linear least squares computations.

```

c+ llsq  least squares solution of a*c=b (solving for c)
      subroutine llsq(a,ia,m,n,ws,c,b,ier)
c parameters
c   a-m by n matrix. declared row dimension ia.
c   ws-working storage vector of length m
c   c-vector of size n
c   b-vector of size m
c   ier-return parameter: ier=0 normal return,ier=1 normal
c     equations
c     nearly singular,ier=2 normal equations singular.
c
c     dimension a(ia,1),b(1),c(1),ws(1)
c   compute lower elements of jth column  of transpose(a)*a
      do 50 j=1,n
        do 18 i=j,n
          s=0.
          do 15 k=1,m
            s=s+a(k,i)*a(k,j)
15      continue
18      ws(i)=s
c
c   compute jth element of right side vector
      s=0.
      do 40 k=1,m
40      s=s+a(k,j)*b(k)
        c(j)=s
c
c   store lower elements of jth column in a
      do 19 i=j,n
19      a(i,j)=ws(i)
c
50      continue
c   fill in upper values
      do 60 i=1,n
        do 60 j=i,n
          a(i,j)=a(j,i)
60      continue
        ib=1

```

```

mm=1
eps=1.e-12
inv=0
c solve normal equations
  call gausse(a,ia,c,ib,n,mm,inv,eps,det,ier)
  return
end

```

0.3 Linear Circle Fitting

In some problems our basis functions may not be defined on a subset of the real numbers. An example is the case of finding a best fitting circle. We are given a set of points $p_i = (x_i, y_i)$. We have two basic functions defined on the set of points, namely a function that maps the point to its x coordinate and a function that maps the point to its y coordinate. Our space of functions will consist of algebraic combinations of these two basic functions. We may write the general equation of a circle as

$$(x - h)^2 + (y - k)^2 = r^2$$

Expanding this gives

$$-2hx - 2ky + (h^2 + k^2 - r^2) = x^2 + y^2.$$

Our functions are

$$f_1(p_i) = x_i, f_2(p_i) = y_i, f_3(p_i) = 1,$$

and

$$g(p_i) = x_i^2 + y_i^2.$$

The coefficients are

$$c_1 = -2h, c_2 = -2k, c_3 = (h^2 + k^2 - r^2).$$

The radius r and the center (h, k) are determined by the coefficients.

One might wonder what happens when the data points lie on a straight line. Then the geometric problem obviously has no solution, while the least squares problem has a solution, because we have shown that it always has a solution. In this case, the functions we have defined for this problem are not

linearly independent. Projection into the spanned subspace gives a minimal function f , but the coefficients of the basis functions are not unique. The solution space is of dimension two. There is an infinite set of coefficients that give the same f . But each set of three coefficients defines a different circle. We are free to choose one of the parameters. This is equivalent to choosing a pair of linearly independent functions. Each time we do this we are solving a different circle problem. For example, if we assume $h=0$ then we are finding the best fitting circle whose center lies on the y-axis.

It is interesting to discuss the meaning of "best fitting." If all points lie on the circle then this algorithm will find the circle. Further the mapping between data sets and circles is continuous, i. e., if the data points are changed by a small amount, then the fitted circle changes by a small amount. The solution we have described may be adequate, but the problem can be formulated more naturally to minimize the perpendicular distance from the fitted circle to the data. Unfortunately the problem becomes nonlinear. The nonlinear circle fitting problem is to minimize the function

$$f(h, k, r) = \sum_{i=1}^n (\|(h, k) - p_i\| - r)^2,$$

where (h, k) are the coordinates of the center, r is the radius, and $\{p_i\}$ is the set of data points. See the section "Nonlinear circle fitting." Here is the linear fitting routine.

```

c+ lsqcir  least squares fit of circle.
      subroutine lsqcir(x,y,n,c,r,ier)
      implicit real*8(a-h,o-z)
c  function      -to calculate the best fitting circle
c                to n points.
c  parameters    x,y-arrays containing the coordinates of
c                the n points.
c                c-array of dimension 2 containing the
c                coordinates of the center of the circle.
c                r-radius of circle.
c                ier-error parameter
c                =0 ,normal return
c                =1 ,equation system is singular
c  change to real*8 and use gaussr 9/19/91
      dimension x(*),y(*),c(*),a(3,3),b(3)

```

```

z=-(x(1)**2+y(1)**2)
b(1)=x(1)*z
b(2)=y(1)*z
b(3)=z
a(1,1)=x(1)**2
a(1,2)=x(1)*y(1)
a(1,3)=x(1)
a(2,2)=y(1)**2
a(2,3)=y(1)
a(3,3)=n
do 10 i=2,n
a(1,1)=a(1,1)+x(i)**2
a(1,2)=a(1,2)+x(i)*y(i)
a(1,3)=a(1,3)+x(i)
a(2,2)=a(2,2)+y(i)**2
a(2,3)=a(2,3)+y(i)
z=-(x(i)**2+y(i)**2)
b(1)=b(1)+z*x(i)
b(2)=b(2)+z*y(i)
10 b(3)=b(3)+z
a(2,1)=a(1,2)
a(3,1)=a(1,3)
a(3,2)=a(2,3)
nn=3
m=1
inv=-1
eps=1.e-15
ia=3
ib=3
idet=0
call gaussr(a,ia,b,ib,nn,m,inv,eps,idet,det,ier)
if(ier.gt.0)then
    ier=1
else
    ier=0
endif
c(1)=-b(1)/2.
c(2)=-b(2)/2.

```

```

r=sqrt(c(1)**2+c(2)**2-b(3))
return
end

```

0.4 Nonlinear Circle Fitting

Let $d(p, c)$ be the distance from a point p to a circle c with center (h, k) and radius r .

$$d^2(p, c) = ([(h - x)^2 + (k - y)^2]^{1/2} - r)^2$$

Let v be defined by

$$v(x, y) = [(h - x)^2 + (k - y)^2]^{1/2}$$

Given n points $p_i = (x_i, y_i)$, we minimize

$$\begin{aligned} f(h, k, r) &= \sum_{i=1}^n d^2(p_i, c) \\ &= \sum_{i=1}^n (v_i - r)^2. \end{aligned}$$

The three necessary equations are

$$\partial f / \partial h = \sum_{i=1}^n 2(v_i - r)(h - x_i) / v_i = 0$$

$$\partial f / \partial k = \sum_{i=1}^n 2(v_i - r)(k - y_i) / v_i = 0$$

$$\partial f / \partial r = - \sum_{i=1}^n 2(v_i - r) = 0$$

We may solve the last equation for r ,

$$r = \frac{\sum_{j=1}^n v_j}{n}$$

Simplifying we have

$$\sum_{i=1}^n (nv_i - \sum_{j=1}^n v_j)(h - x_i) / v_i = 0$$

$$\sum_{i=1}^n (nv_i - \sum_{j=1}^n v_j)(k - y_i)/v_i = 0$$

We may write this as a vector equation

$$\mathbf{F}(h, k) = 0.$$

To solve this equation we may compute the Jacobian \mathbf{J} and use Newton's method

$$\mathbf{X}_{n+1} = \mathbf{X}_n - \mathbf{F}(h, k)\mathbf{J}^{-1},$$

where

$$\mathbf{X} = \begin{bmatrix} h \\ k \end{bmatrix}.$$

The following computer program solves the two variable equation using an IMSL root solver. Starting values are computed from the linear fitting routine LSQCIR.

```

c fitc2_for fit a circle 1-29-90 (nos/ve)
  parameter (n=2,m=5)
  dimension x(n),xguess(n),f(n)
  real xpp(m),ypp(m),c(2)
  external fcn,neqnf
  common /points/ npts, xp(200), yp(200)
  data (xpp(j),j=1,m)/1.,0.,-1.,.0,1./
  data (ypp(j),j=1,m)/9.,1.,0.,-1.,0./
  do 1 i=1,m
  xp(i)=xpp(i)
  yp(i)=ypp(i)
1  continue
  npts=m
  call lsqcir(xp,yp,m,c,r,ier)
  write(*,*)' linear solution: '
  write(*,*)' center = ',c(1),c(2)
  write(*,*)' radius = ',r
  write(*,*)' ier = ',ier
  call distc(xp,yp,m,c(1),c(2),r,dmn,dmx,dmean)
  write(*,*)' dmn,dmx,dmean = ',dmn,dmx,dmean
  er=.0000001
  itmax=200

```

```

xguess(1)=c(1)
xguess(2)=c(2)
call neqnf(fcn,er,n,itmax,xguess,x,fnorm)
c neqnf is an imsl root finder
c(1)=x(1)
c(2)=x(2)
r=234.
write(*,*)' nonlinear solution: '
write(*,*)' center = ',c(1),c(2)
sumv=0
do 5 i=1,m
sumv=sumv+sqrt((c(1)-xpp(i))**2+(c(2)-ypp(i))**2)
5 continue
r=sumv/m
write(*,*)' radius = ',r
call distc(xp,yp,m,c(1),c(2),r,dmn,dmx,dmean)
write(*,*)' dmn,dmx,dmean = ',dmn,dmx,dmean
fn=fcn(x,f,n)
do 20 i=1,n
write(*,*)' function ',i,f(i)
20 continue
end
c
function fcn(p,f,n)
common /points/ npts,x(200),y(200)
real p(n),f(n),h,k,r
h=p(1)
k=p(2)
write(*,*)' h k = ',h,k
sumv=0
do 5 i=1,npts
sumv=sumv+sqrt((h-x(i))**2+(k-y(i))**2)
5 continue
s1=0.
s2=0.
do 10 i=1,npts
v=sqrt((h-x(i))**2+(k-y(i))**2)
s1=s1+(npts*v-sumv)*(h-x(i))/v

```

```

        s2=s2+(npts*v-sumv)*(k-y(i))/v
10    continue
        f(1)=s1
        f(2)=s2
        fcn=1.
        return
        end

```

This program computes the distance function.

```

c+ distc  distances between points and circle
        subroutine distc(x,y,n,cx,cy,r,dmn,dmx,rms)
c  input:
c      x,y-points
c      n-number of points
c      cx,cy-center of circle
c  output:
c      dmn-minimum distance from a point to circle
c      dmx-maximum distance from a point to circle
c      rms-square root of the mean square distance
real x(n),y(n)
do 10 i=1,n
d2=(sqrt((x(i)-cx)**2+(y(i)-cy)**2)-r)**2
d=sqrt(d2)
if(i.eq.1)then
    sum=d2
    dmn=d
    dmx=d
else
    sum=sum+d2
    if(d.lt.dmn)dmn=d
    if(d.gt.dmx)dmx=d
endif
10    continue
rms=sqrt(sum/n)
return
end

```

0.5 Weighted Least Squares

We can give more importance to certain data points in the fitting process by assigning weights. We can modify our inner product and use the general theory. Define a diagonal n by n matrix W with diagonal elements w_i . Define a new inner product ρ by

$$\rho(f_i, f_j) = f_i^T W f_j.$$

The normal equations become

$$A^T W A x = A^T W b.$$

As an example of this technique we take the exponential fitting problem.

$$y = ae^{bx}$$

This is a nonlinear problem. We take logarithms to get a related linear problem

$$\ln(y_i) = \ln(a) + bx_i.$$

Let

$$\delta_i = \ln(a) + bx_i - \ln(y_i).$$

The linear least squares technique will minimize the sum of squares of the δ_i 's. But we want to minimize the sum of squares of the deviations of the original problem

$$d_i = ae^{bx_i} - y_i.$$

We can find an approximate relation between these deviations. Applying the exponential function we have

$$e^{\delta_i} = ae^{bx_i} / y_i.$$

If we replace the exponential function by the first two terms of its Taylor series expansion, we get

$$y_i(1 + \delta_i) = ae^{bx_i}.$$

Then

$$y_i \delta_i = ae^{bx_i} - y_i = d_i.$$

We can use $w_i = y_i^2$ and apply weighted least squares to improve the approximate solution to the nonlinear problem.

0.6 Bilinear Interpolation Example

Let the unit square be mapped to a region defined by four corner points $p_{00}, p_{10}, p_{01}, p_{11}$. Define the mapping by

$$q_1 = (1 - u)p_{00} + up_{10}$$

$$q_2 = (1 - u)p_{01} + up_{11}$$

and

$$p(u, v) = (1 - v)q_1 + vq_2.$$

This is a bilinear function. Expanding, the equation becomes

$$p(u, v) = p_{00} + (p_{00} + p_{10})u + (p_{01} - p_{00})v + (p_{00} - p_{10} - p_{01} + p_{11})uv.$$

Now suppose we are given a data set

$$\{(u_i, v_i, b_i) : i = 1, \dots, m\},$$

and we are to find a bilinear mapping approximately taking each pair (u_i, v_i) to the vector b_i . We obtain a set of m equations

$$c_1 + c_2u_i + c_3v_i + c_4u_iv_i = b_i,$$

where

$$p_{00} = c_1,$$

$$p_{10} = c_2 - c_1,$$

$$p_{01} = c_3 + c_1,$$

$$p_{11} = c_4 + c_2 + c_3 - c_1.$$

This problem can be solved by least squares. There are three independent systems for the x , y , and z components.

0.7 Fitting Functions With Horizontal Asymptotes, Using the general program lsqgen.

One approach to fitting when there are horizontal asymptotes is to swap the variables so that the asymptotes become vertical. A vertical asymptote is due to a pole. Thus consider the function

$$g(y) = x = \frac{a}{y} + \frac{b}{y - 1}.$$

The function has a pole at 0 and a pole at 1. Suppose a and b are 1. Then solving for y we get the inverse

$$f(x) = y = \frac{1}{2} \frac{x + 2 - \sqrt{x^2 + 4}}{x}.$$

Note that y goes to zero as x goes to positive infinity, and y goes to 1 as x goes to negative infinity.

The technique then is to fit a linear combination of poles using least squares, and then invert the resulting function. This is equivalent to finding symbolically the roots of a polynomial. In the example, the polynomial is of second degree. For higher degree cases one may have to use numerical methods to invert the function. Consider the data which has been switched to give vertical asymptotes:

x	y
0.20000000	7.5000000
0.50000000	4.5000000
2.10000000	1.8000000
3.00000000	1.0000000
4.00000000	-1.0000000
4.50000000	-3.0000000
4.80000000	-5.5000000

We shall try to fit the following linear combination of functions

$$g = a_1 + a_2x + a_3/x + a_4/x^2 + a_5/(x - 5) + a_6/(x - 5)^2,$$

using `lsqgen.ftn`. We get the following output:

```
% lsqgen
linear least squares curve fitting
maximum points= 200
maximum degree= 19
enter file name [p.dat]
pp.dat
7 points

f(x) =
f(x)= 3.48346534312582      f( 1) +
```

```

-.560314483245716      f( 2) +
0.932168101605212      f( 3) +
0.260354762617237E-03 f( 4) +
  2.63951411841873      f( 5) +
0.268481286458776      f( 6)
      data
      x          y          fit
0.20000000      7.5000000      7.5005059
0.50000000      4.5000000      4.4953854
2.10000000      1.8000000      1.8725003
3.00000000      1.0000000      0.86063678
4.00000000      -1.0000000     -0.89576712
4.50000000      -3.0000000     -3.0358916
4.80000000      -5.5000000     -5.4973696
sigma = 0.72564817782510E-01
      enter a plot file name [p.gi]

Enter plot window, default is:
[ 0.20000000      4.8000000      -5.5000000      7.5000000      ]
.1 4.9 -8 8
% pltgpr

```

We can try to invert this equation with Maple. The closed form inversion is too messy for practical use. Refer to the next section for a listing of `lsqgen.ftn`.

0.8 A General LSQ Program `lsqgen.ftn`

`lsqgen.ftn` defines a set of functions in a function subroutine located at the bottom of the listing. To change the LSQ problem, one changes these definitions, and the variable n , which is the number of functions in the linear combination. `lsqgen.ftn` also generates a plotfile (.gi file).

0.9 Listing of `lsqgen.ftn`

```

c lsqgen.ftn general linear least squares program 5/5/95
c functions defined in f at bottom of listing
      implicit real*8 (a-h,o-z)

```

```

common /block1/x1,n
parameter (numpts=200,nfncts=20)
dimension x(numpts),y(numpts),xt(numpts),yt(numpts)
dimension a(numpts,nfncts),b(numpts),c(nfncts),ws(numpts)
dimension ain(5)
character*20 ans
character*30 fn,pltfn
ia=numpts
zero=0.
write(*,*)' linear least squares curve fitting'
write(*,*)' maximum points= ',numpts
write(*,*)' maximum degree= ',(nfncts-1)
c ccccccccccccc read data points
m=0
write(*,*)' enter file name [p.dat]'
read(*,'(a)')fn
if(lenstr(fn).lt.1)then
  fn='p.dat'
endif
open(3,file=fn,status='unknown')
10 continue
call readr(3,ain,nr)
if(nr.le.0)then
  close(3)
  write(*,*)m,' points'
  if(m .eq. 0)then
    stop
  endif
  go to 20
endif
m=m+1
if(m.gt.numpts)then
  write(*,*)' the file contains more than the allowed '
  write(*,*)' number of points.'
  write(*,*)m,' points read'
  close(3)
  stop
endif
x(m)=ain(1)
y(m)=ain(2)
go to 10
20 continue
c ccccccccccccc construct and solve normal equations
c make a dummy call to f to define n
a(1,1)=f(1,x(1))
do 40 i=1,m
  b(i)=y(i)
  do 30 j=1,n
    a(i,j)=f(j,x(i))
30 continue
40 continue
call llsq(a,ia,m,n,ws,c,b,ier)
if(ier .ne. 0)then
  write(*,*)' llsq has returned an error condition'
  write(*,*)' ier= ',ier
  write(*,*)' normal equations are singular, or nearly singular'
  stop

```

```

endif
write(*,*)
write(*,*)' f(x) ='
do 50 i=1,n
  if(c(i) .ne. zero)then
    if(i .eq. 1)then
      write(*,'(a,g21.15,a,i2,a)')'f(x)=' ,c(i),' f(' ,i,') +'
    else
      if(i .ne. n)then
        write(*,'(6x,g21.15,a,i2,a)')c(i),' f(' ,i,') +'
      else
        write(*,'(6x,g21.15,a,i2,a)')c(i),' f(' ,i,') +'
      endif
    endif
  endif
endif
50 continue
write(*,*)'          data'
write(*,*)'      x          y          fit'
sigma=0.
do 60 i=1,m
  xx=x(i)
  yy=y(i)
  call flc(c,n,xx,fx)
  write(*,'(3(g15.8,1x)')')xx,yy,fx
  sigma=sigma+(y(i)-fx)**2
60 continue
sigma=sqrt(sigma/m)
write(*,'(a,g21.14)')'sigma = ',sigma
c ccccccccccccccccccccccccc plot of data and fitted function
write(*,*)' enter a plot file name [p.gi]'
read(*,'(a)')pltfn
if(lenstr(pltfn).lt.1)then
  pltfn='p.gi'
endif
open(2,file=pltfn,status='unknown')
call plotgi(c,n,x,y,m)
end
c+ plotgi plot gi file
subroutine plotgi(c,n,x,y,m)
  implicit real*8 (a-h,o-z)
  dimension x(*),y(*),c(*)
  dimension ain(4)
  nc=4
  x1=1.0e30
  x2=-x1
  y1=x1
  y2=x2
  do 5 i=1,m
    if(x(i).gt.x2)x2=x(i)
    if(x(i).lt.x1)x1=x(i)
    if(y(i).gt.y2)y2=y(i)
    if(y(i).lt.y1)y1=y(i)
5  continue
c  dx=(x2-x1)/20.
c  x1=x1-dx
c  x2=x2+dx
c  dy=(y2-y1)/20.

```

```

c      y1=y1-dy
c      y2=y2+dy
      write(2,'(a)')'v-1 1 -1 1'
      write(*,*)'Enter plot window, default is:'
      write(*,'(a,4(g15.8,1x),a)')'[',x1,x2,y1,y2,']'
      call readr(0,ain,nr)
      if(nr .eq. 4)then
        x1=ain(1)
        x2=ain(2)
        y1=ain(3)
        y2=ain(4)
      endif
      write(2,'(a,4(g12.5,1x))')'w',x1,x2,y1,y2
      do 10 i=1,m
        write(2,'(a,g12.5,1x,g12.5,a)')'s',x(i),y(i),' 1 .01'
10     continue
        nc = mod(nc,5)+3
        write(2,'(a,i2)')'c',nc
        npts=200
        do 30 i=1,npts
          u=(i-1)*(x2-x1)/(npts-1)+x1
          call flc(c,n,u,v)
          if(i .eq. 1)then
            write(2,'(a,g12.5,1x,g12.5)')'m',u,v
          else
            write(2,'(a,g12.5,1x,g12.5)')'d',u,v
          endif
30     continue
      end
c
c+ gausse  solution of linear system, matrix inverse, determinant
      subroutine gausse(a,ia,b,ib,n,m,inv,eps,det,ier)
        implicit real*8 (a-h,o-z)
c  algorithm      -gaussian elimination with partial pivoting.
c  parameters     a-n by n matrix containing the coefficients of
c                 the linear system.
c                 ia-row dimension of a in calling program
c                 b-n by m matrix containing the m right sides
c                 of the equations, on entering.  on returning, b
c                 contains the solutions.  the inverse of a is
c                 returned in b when that option is selected.
c                 ib-row dimension of b in calling program
c                 n-row and column dimension of a.
c                 m-column dimension of b.
c                 inv-when inv=1 the inverse is calculated
c                    and returned in b
c                 eps-each equation is normalized so that the
c                    coefficients are .le. 1 in magnitude.
c                    when a pivot is less than eps the matrix is
c                    considered nearly singular.
c                    if a pivot is zero the matrix is singular.
c                 det-determinant of a.
c                 ier-return parameter, ier=0 is normal return
c                    ier=1 when matrix is nearly singular
c                    ier=2 when the matrix is singular
c
      logical p

```

```

        dimension a(ia,*),b(ib,*)
        zero=0.
        ier=0
        if(inv.ne.1)go to 15
c      set b equal to the identity
        do 10 i=1,n
        do 10 j=1,n
        b(i,j)=0.
        if(i.eq.j)b(i,j)=1.
    10  continue
        m=n
    15  continue
c  normalize rows
        do 20 i=1,n
        biggest=abs(a(i,1))
        do 16 j=2,n
        ab=abs(a(i,j))
        if(ab.gt.biggest)biggest=ab
    16  continue
        p=biggest.eq.zero
        if(p)ier=2
        if(p)det=0.
        if(p)return
        do 18 j=1,n
    18  a(i,j)=a(i,j)/biggest
        do 19 j=1,m
    19  b(i,j)=b(i,j)/biggest
    20  continue
        det=1.
        j=1
    30  kk=j+1
        l=j
        do 32 i=kk,n
    32  if(abs(a(i,j)).gt.abs(a(l,j)))l=i
        p=abs(a(l,j)).eq.zero
        if(p)ier=2
        if(p)det=0.
        if(p)return
        p=abs(a(l,j)).le.eps
        if(p)ier=1
    34  if(l.eq.j)go to 40
c  interchange rows
        do 37 k=1,n
        c=a(l,k)
        a(l,k)=a(j,k)
    37  a(j,k)=c
        do 39 k=1,m
        c=b(l,k)
        b(l,k)=b(j,k)
    39  b(j,k)=c
        det=det*(-1.)
    40  det=det*a(j,j)
c  divide row by pivot
        c=a(j,j)
        do 50 k=j,n
    50  a(j,k)=a(j,k)/c
        do 55 k=1,m

```

```

55 b(j,k)=b(j,k)/c
c   add multiple of row j
    jj=j+1
    do 70 i=jj,n
      am=a(i,j)
      do 60 k=1,n
60  a(i,k)=a(i,k)-am*a(j,k)
      do 70 k=1,m
70  b(i,k)=b(i,k)-am*b(j,k)
      j=j+1
      if(j.ne.n)go to 30
      am=a(n,n)
      p=abs(am).eq.zero
      if(p)ier=2
      if(p)det=0.
      if(p)return
      p=abs(am).le.eps
      if(p)ier=1
73  det=det*am
      do 80 k=1,m
80  b(n,k)=b(n,k)/am
c   back substitution
    nn=n-1
    do 100 i=1,nn
      ni=n-i
      do 100 j=1,m
        nj=ni+1
        do 100 ki=nj,n
100  b(ni,j)=b(ni,j)-a(ni,ki)*b(ki,j)
      return
    end
c
c+ llsq  least squares solution of a*c=b (solving for c)
      subroutine llsq(a,ia,m,n,ws,c,b,ier)
        implicit real*8 (a-h,o-z)
c parameters
c   a-m by n matrix. declared row dimension ia.
c   ws-working storage vector of length m
c   c-vector of size n
c   b-vector of size m
c   ier-return parameter: ier=0 normal return,ier=1 normal equations
c   nearly singular,ier=2 normal equations singular.
c
      dimension a(ia,*),b(*),c(*),ws(*)
c   compute lower elements of jth column of transpose(a)*a
      do 50 j=1,n
        do 18 i=j,n
          s=0.
          do 15 k=1,m
            s=s+a(k,i)*a(k,j)
15      continue
18      ws(i)=s
c
c   compute jth element of right side vector
      s=0.
      do 40 k=1,m
40      s=s+a(k,j)*b(k)

```

```

        c(j)=s
c
c store lower elements of jth column in a
  do 19 i=j,n
19   a(i,j)=ws(i)
c
50   continue
c fill in upper values
  do 60 i=1,n
  do 60 j=i,n
  a(i,j)=a(j,i)
60   continue
  ib=1
  mm=1
  eps=1.e-12
  inv=0
c solve normal equations
  mprnt=0
  if(mprnt.eq.1)then
    write(*,*)' coefficients of normal equations a= '
    do 500 i=1,n
      write(*,'(7(1x,g11.4))')(a(i,j),j=1,n)
500   continue
    read(*,*)
    write(*,*)' b= '
    do 510 i=1,n
      write(*,'(1(1x,g11.4))')c(i)
510   continue
    read(*,*)
  endif
  call gausse(a,ia,c,ib,n,mm,inv,eps,det,ier)
  return
end

c+ readr read a row of floating point numbers
  subroutine readr(nf, a, nr)
  implicit real*8(a-h,o-z)
c numbers are separated by spaces
c examples of valid numbers are:
c 12.13 34 45e4 4.78e-6 4e2,5.6D-23,10000.d015
c nf=file number, 0 for standard input file
c a=array of returned numbers
c nr=number of values in returned array,
c or 0 for empty or blank line,
c or -1 for end of file on unit nf.
c requires functions val and length
  dimension a(*)
  character*200 b
  character*200 c
  character*1 d
  c=' '
  if(nf.eq.0)then
    read(*,'(a)',end=99)b
  else
    read(nf,'(a)',end=99)b
  endif
  nr=0
  l=lenstr(b)

```

```

if(l.ge.200)then
  write(*,*)' error in readr subroutine '
  write(*,*)' record is too long '
endif
do 1 i=1,l
  d=b(i:i)
  if (d.ne.' ') then
    k=lenstr(c)
    if (k.gt.0)then
      c=c(1:k)//d
    else
      c=d
    endif
  endif
  if( (d.eq.' ').or.(i.eq.l)) then
    if (c.ne.' ') then
      nr=nr+1
      call valsub(c,a(nr),ier)
      c=' '
    endif
  endif
1 continue
return
99 nr=-1
return
end

c
c+ lenstr nonblank length of string
  function lenstr(s)
c length of the substring of s obtained by deleting all
c trailing blanks from s. thus the length of a string
c containing only blanks will be 0.
  character s(*)
  lenstr=0
  n=len(s)
  do 10 i=n,1,-1
  if(s(i:i).ne.' ')then
    lenstr=i
    return
  endif
10 continue
return
end

c+ valsub converts string to floating point number (r*8)
  subroutine valsub(s,v,ier)
  implicit real*8(a-h,o-z)
c examples of valid strings are: 12.13 34 45e4 4.78e-6 4E2
c the string is checked for valid characters,
c but the string can still be invalid.
c s-string
c v-returned value
c ier- 0 normal
c 1 if invalid character found, v returned 0
c
  logical p
  character s*(*),c*50,t*50,ch*15
  character z*1

```

```

data ch/'1234567890+- .eE'/
v=0.
ier=1
l=lenstr(s)
if(l.eq.0)return
p=.true.
do 10 i=1,l
z=s(i:i)
if((z.eq.'D').or.(z.eq.'d'))then
s(i:i)='e'
endif
p=p.and.(index(ch,s(i:i)).ne.0)
10 continue
if(.not.p)return
n=index(s,'.')
if(n.eq.0)then
n=index(s,'e')
if(n.eq.0)n=index(s,'E')
if(n.eq.0)n=index(s,'d')
if(n.eq.0)n=index(s,'D')
if(n.eq.0)then
s=s(1:l)///'.'
else
t=s(n:l)
s=s(1:(n-1))///'.'//t
endif
l=l+1
endif
write(c,'(a30)')s(1:l)
read(c,'(g30.23)')v
ier=0
return
end

c+ str floating point number to string
subroutine str(x,s)
implicit real*8(a-h,o-z)
character s*25,c*25,b*25,e*25
zero=0.
if(x.eq.zero)then
s='0'
return
endif
write(c,'(g11.4)')x
read(c,'(a25)')b
l=lenstr(b)
do 10 i=1,l
n1=i
if(b(i:i).ne.' ')go to 20
10 continue
20 continue
if(b(n1:n1).eq.'0')n1=n1+1
b=b(n1:l)
l=l+1-n1
k=index(b,'E')
if(k.gt.0)e=b(k:l)
if(k.gt.0)then
s=b(1:(k-1))

```

```

        k1=index(b,'E+0')
        if(k1.gt.0)then
            e='E'//b((k1+3):1)
        else
            k1=index(b,'E+')
            if(k1.gt.0)e='E'//b((k1+2):1)
        endif
        k1=index(b,'E-0')
        if(k1.gt.0)e='E-'//b((k1+3):1)
        l=k-1
    else
        s=b
    endif
    endif
    j=index(s, '.')
    n2=1
    if(j.ne.0)then
        do 30 i=1,l
            n2=1+1-i
            if(s(n2:n2).ne.'0')go to 40
30        continue
    endif
40    continue
    s=s(1:n2)
    if(s(n2:n2).eq.'.')then
        s=s(1:(n2-1))
        n2=n2-1
    endif
    if(k.gt.0)s=s(1:n2)//e
    return
end
c+ flc value of linear combination function: flc(x) = c_{i} f_i(x)
subroutine flc(c,n,x,fx)
    implicit real*8 (a-h,o-z)
    dimension c(*)
    fx=0.
    do 10 i=1,n
        fx = fx + c(i)*f(i,x)
10    continue
    return
end
c+ f definition of the ith fitting function: f_i(x)
function f(i,x)
    implicit real*8 (a-h,o-z)
    common /block1/x1,n
c    define parameters x1 = pole
    x1 = 5.
c    define number of functions n, main program will make a dummy call
c    to obtain n
    n=6
    zero=0.
    if(i .eq. 1)then
        f=1.
    endif
    if(i .eq. 2)then
        f=x
    endif
    if(i .eq. 3)then

```

```
if(x .ne. zero)then
  f=1./x
endif
endif
if(i .eq. 4)then
  if(x .ne. zero)then
    f=(1./x)**2
  endif
endif
if(i .eq. 5)then
  r=x-x1
  if(r .ne. zero)then
    f=(1./r)
  endif
endif
if(i .eq. 6)then
  r=x-x1
  if(r .ne. zero)then
    f=(1./r)**2
  endif
endif
return
end
```