1

Fitting functions to data

1.1 Exact fitting

1.1.1 Introduction

Suppose we have a set of real-number data pairs $x_i, y_i,\ i = 1, 2, \ldots, N$. These can be considered to be a set of points in the xy-plane. They can also be thought of as a set of values y of a function of x; see Fig. 1.1. A frequent challenge is to find some kind of function that represents a "best fit" to the data in some sense. If the data were fitted perfectly, then clearly the function f would have the property

$$f(x_i) = y_i, \qquad \text{for all } i = 1, \ldots, N. \tag{1.1}$$

When the number of pairs is small and they are reasonably spaced out in x, then it may be reasonable to do an exact fit that satisfies this equation.

1.1.2 Representing an exact fitting function linearly

We have an infinite choice of possible fitting functions. Those functions must have a number of different adjustable parameters that are set so as to adjust the

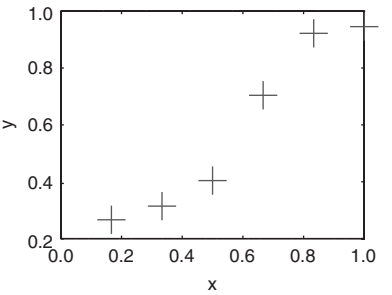

Figure 1.1 Example of data to be fitted with a curve.

https://doi.org/10.1017/CBO9781316155516.002 Published online by Cambridge University Press

function to fit the data. One example is a polynomial,

$$f(x) = c_1 + c_2 x + c_3 x^2 + \ldots + c_N x^{N-1}. \tag{1.2}$$

Here the c_i are the coefficients that must be adjusted to make the function fit the data. A polynomial whose coefficients are the adjustable parameters has a very useful property that it is linearly dependent upon the coefficients.

In order to fit eqs. (1.1) with the form of eq. (1.2) requires that N simultaneous equations be satisfied. Those equations can be written as an $N \times N$ matrix equation as follows:

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{N-1} \\ 1 & x_2 & x_2^2 & \ldots & x_2^{N-1} \\ \ldots & & & & \\ 1 & x_N & x_N^2 & \ldots & x_N^{N-1} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \ldots \\ c_N \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \ldots \\ y_N \end{pmatrix} \tag{1.3}$$

Here we notice that in order for this to be a square matrix system we need the number of coefficients to be equal to the number of data pairs N.

We also see that we could have used any set of N functions f_i as fitting functions, and written the representation:

$$f(x) = c_1 f_1(x) + c_2 f_2(x) + c_3 f_3(x) + \ldots + c_N f_N(x) \tag{1.4}$$

and then we would have obtained the matrix equation

$$\begin{pmatrix} f_1(x_1) & f_2(x_1) & f_3(x_1) & \ldots & f_N(x_1) \\ f_1(x_2) & f_2(x_2) & f_3(x_2) & \ldots & f_N(x_2) \\ \ldots & & & & \\ f_1(x_N) & f_2(x_N) & f_3(x_N) & \ldots & f_N(x_N) \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \ldots \\ c_N \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \ldots \\ y_N \end{pmatrix} \tag{1.5}$$

This is the most general form of representation of a fitting function that varies linearly with the unknown coefficients. The matrix[1] we will call $\mathbf{S}$. It has elements $S_{ij} = f_j(x_i)$

1.1.3 Solving for the coefficients

When we have a matrix equation of the form $\mathbf{Sc} = \mathbf{y}$, where $\mathbf{S}$ is a square matrix, then provided that the matrix is non-singular, that is, provided its

[1] Throughout this book, matrices and vectors in abstract multidimensional space are denoted by upright bold font. Vectors in physical three-dimensional space are instead denoted by italic bold font.

https://doi.org/10.1017/CBO9781316155516.002 Published online by Cambridge University Press

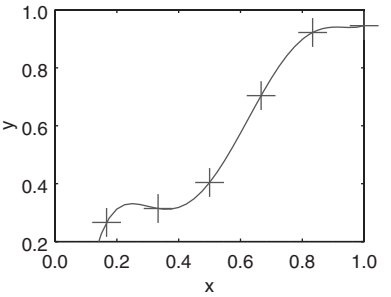

Figure 1.2 Result of the polynomial fit.

determinant is non-zero, $|\mathbf{S}| \neq 0$, it possesses an inverse $\mathbf{S}^{-1}$. Multiplying on the left by this inverse we get:

$$\mathbf{S}^{-1}\mathbf{S}\mathbf{c} = \mathbf{c} = \mathbf{S}^{-1}\mathbf{y}. \tag{1.6}$$

In other words, we can solve for $\mathbf{c}$, the unknown coefficients, by inverting the function matrix, and multiplying the values to be fitted, $\mathbf{y}$, by that inverse.

Once we have the values of $\mathbf{c}$ we can evaluate the function $f(x)$ (eq. 1.2) at any x-value we like. Fig. 1.2 shows the result of fitting a fifth order polynomial (with six terms including the 1) to the six points of our data. The line goes exactly through every point. But there's a significant problem that the line is unconvincingly curvy near its ends.[2] It's not a terribly good fit.

1.2 Approximate fitting

If we have lots of data which have scatter in them, arising from uncertainties or noise, then we almost certainly *do not* want to fit a curve so that it goes exactly through every point. For example, see Fig. 1.3. What do we do then? Well, it turns out that we can use almost exactly the same approach, except with *different* number of points (N) and terms (M) in our linear fit. In other words we use a representation

$$f(x) = c_1 f_1(x) + c_2 f_2(x) + c_3 f_3(x) + \ldots + c_M f_M(x), \tag{1.7}$$

in which usually $M < N$. We know now that we *can't* fit the data exactly. The set of equations we would have to satisfy to do so would be

[2] This problem with polynomial fitting is sometimes called Runge's phenomenon.

https://doi.org/10.1017/CBO9781316155516.002 Published online by Cambridge University Press

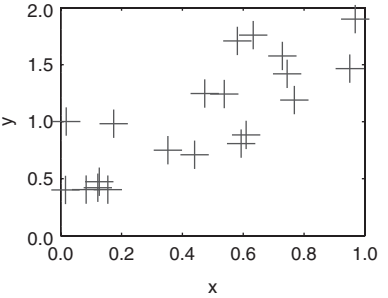

Figure 1.3 A cloud of points with uncertainties and noise, to be fitted with a function.

$$\begin{pmatrix} f_1(x_1) & f_2(x_1) & f_3(x_1) & \dots & f_M(x_1) \\ f_1(x_2) & f_2(x_2) & f_3(x_2) & \dots & f_M(x_2) \\ \dots & & & & \\ f_1(x_N) & f_2(x_N) & f_3(x_N) & \dots & f_M(x_N) \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \dots \\ c_M \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_N \end{pmatrix}, \quad (1.8)$$

in which the function matrix $\mathbf{S}$ is now not square but has dimensions $N \times M$. There are not enough coefficients c_j to be able to satisfy these equations exactly. They are over-specified. Moreover, a non-square matrix doesn't have an inverse.

But we are not interested in fitting these data exactly. We want to fit some sort of line through the points that best fits them.

1.2.1 Linear least squares

What do we mean by "best fit"? Especially when fitting a function of the linear form eq. (1.7), we usually mean that we want to minimize the vertical distance between the points and the line. If we had a fitted function $f(x)$, then for each data pair (x_i, y_i), the square of the vertical distance between the line and the point is $(y_i - f(x_i))^2$. So the sum, over all the points, of the square distance from the line is

$$\chi^2 = \sum_{i=1,N} (y_i - f(x_i))^2. \quad (1.9)$$

We use the square of the distances in part because they are always positive. We don't want to add positive and negative distances, because a negative distance is just as bad as a positive one and we don't want them to cancel out. We generally call χ^2 the "residual", or more simply the "chi-squared". It is an inverse measure of goodness of fit. The smaller it is the better. A linear

https://doi.org/10.1017/CBO9781316155516.002 Published online by Cambridge University Press

least-squares problem is: find the coefficients of our function f that minimize the residual χ^2.

1.2.2 SVD and the Moore–Penrose pseudo-inverse

We seem to have gone off in a different direction from our original way to solve for the fitting coefficients by inverting the square matrix $\mathbf{S}$. How is that related to the finding of the least-squares solution to the over-specified set of equations (1.8)?

The answer is a piece of matrix magic! It turns out that there *is* (contrary to what one is taught in an elementary matrix course) a way to define the inverse of a non-square matrix or of a singular square matrix. It is called the (Moore–Penrose) pseudo-inverse. And once found it can be used in essentially exactly the way it was for the non-singular square matrix in the earlier treatment. That is, we solve for the coefficients using $\mathbf{c} = \mathbf{S}^{-1}\mathbf{y}$, except that $\mathbf{S}^{-1}$ is now the pseudo-inverse.

The pseudo-inverse is best understood from a consideration of what is called the singular value decomposition (SVD) of a matrix. This is the embodiment of a theorem in matrix mathematics that states that any $N \times M$ matrix can always be expressed as the product of three other matrices with very special properties. For our $N \times M$ matrix $\mathbf{S}$ this expression is:

$$\mathbf{S} = \mathbf{U}\mathbf{D}\mathbf{V}^T, \tag{1.10}$$

where ^{T} denotes transpose, and

- $\mathbf{U}$ is an $N \times N$ orthonormal matrix
- $\mathbf{V}$ is an $M \times M$ orthonormal matrix
- $\mathbf{D}$ is an $N \times M$ diagonal matrix.

Orthonormal[3] means that the dot product of any column (regarded as a vector) with any other column is zero, and the dot product of a column with itself is unity. The inverse of an orthonormal matrix is its transpose. So

$$\underbrace{\mathbf{U}^T}_{N\times N}\underbrace{\mathbf{U}}_{N\times N} = \underbrace{\mathbf{I}}_{N\times N} \qquad \text{and} \qquad \underbrace{\mathbf{V}^T}_{M\times M}\underbrace{\mathbf{V}}_{M\times M} = \underbrace{\mathbf{I}}_{M\times M}, \tag{1.11}$$

A diagonal matrix has non-zero elements only on the diagonal. But if it is non-square, as it is if $M < N$, then it is padded with extra rows of zeros (or extra columns if $N < M$):

[3] Sometimes called simply "orthogonal," the real version of "unitary."

https://doi.org/10.1017/CBO9781316155516.002 Published online by Cambridge University Press

$$\mathbf{D} = \begin{pmatrix} d_1 & 0 & 0 & \dots & 0 \\ 0 & d_2 & 0 & & \\ 0 & & \ddots & & \\ \vdots & & & \ddots & 0 \\ & & 0 & 0 & d_M \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{1.12}$$

A sense of what the SVD is can be gained from by thinking[4] in terms of the eigenanalysis of the matrix $\mathbf{S}^T\mathbf{S}$. Its eigenvalues are d_i^2.

The pseudo-inverse can be considered to be

$$\mathbf{S}^{-1} = \mathbf{V}\mathbf{D}^{-1}\mathbf{U}^T. \tag{1.13}$$

Here, $\mathbf{D}^{-1}$ is an $M \times N$ diagonal matrix whose entries are the inverse of those of $\mathbf{D}$, i.e. $1/d_j$:

$$\mathbf{D}^{-1} = \begin{pmatrix} 1/d_1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1/d_2 & 0 & & & 0 \\ 0 & & \ddots & & & 0 \\ \vdots & & & \ddots & 0 & 0 \\ 0 & \dots & 0 & 0 & 1/d_M & 0 \end{pmatrix}. \tag{1.14}$$

It's clear that eq. (1.13) is in some sense an inverse of $\mathbf{S}$ because formally

$$\mathbf{S}^{-1}\mathbf{S} = (\mathbf{V}\mathbf{D}^{-1}\mathbf{U}^T)(\mathbf{U}\mathbf{D}\mathbf{V}^T) = \mathbf{V}\mathbf{D}^{-1}\mathbf{D}\mathbf{V}^T = \mathbf{V}\mathbf{V}^T = \mathbf{I}. \tag{1.15}$$

[4] **Enrichment:** A highly abbreviated outline of the SVD is as follows. The $M \times M$ matrix $\mathbf{S}^T\mathbf{S}$ is symmetric. Therefore, it has real eigenvalues d_i^2, which because of its form are non-negative. Its eigenvectors $\boldsymbol{v}_i$, satisfying $\mathbf{S}^T\mathbf{S}\boldsymbol{v}_i = d_i^2\boldsymbol{v}_i$, can be arranged into an orthonormal set in order of decreasing magnitude of d_i^2. The M eigenvectors can be considered the columns of an orthonormal matrix $\mathbf{V}$, which diagonalizes $\mathbf{S}^T\mathbf{S}$ so that $\mathbf{V}^T\mathbf{S}^T\mathbf{S}\mathbf{V} = \mathbf{D}^2$ is an $M \times M$ non-negative, diagonal matrix with diagonal values d_i^2. Since $(\mathbf{S}\mathbf{V})^T\mathbf{S}\mathbf{V}$ is diagonal, the columns of the $N \times M$ matrix $\mathbf{S}\mathbf{V}$ are orthogonal. Its columns corresponding to $d_i = 0$ are zero and are not useful to us. The useful columns corresponding to non-zero d_i ($i = 1, \dots, L$, say, $L \leq M$) can be normalized by dividing by d_i. Then, by appending $N - L$ normalized column N-vectors that are orthogonal to all the previous ones, we can construct a complete $N \times N$ orthonormal matrix $\mathbf{U} = [\mathbf{S}\mathbf{V}\mathbf{D}_L^{-1}, \mathbf{U}_{N-L}]$. Here $\mathbf{D}_L^{-1}$ denotes the $M \times L$ diagonal matrix with elements $1/d_i$ and $\mathbf{U}_{N-L}$ denotes the appended extra columns. Now consider $\mathbf{U}\mathbf{D}\mathbf{V}^T$. The appended $\mathbf{U}_{N-L}$ make zero contribution to this product because the lower rows of $\mathbf{D}$ which they multiply are always zero. The rest of the product is $\mathbf{S}\mathbf{V}\mathbf{D}_L^{-1}\mathbf{D}_L\mathbf{V}^T = \mathbf{S}$. Therefore we have constructed the singular value decomposition $\mathbf{S} = \mathbf{U}\mathbf{D}\mathbf{V}^T$.

https://doi.org/10.1017/CBO9781316155516.002 Published online by Cambridge University Press

If $M \leq N$ and none of the d_j is zero, then all the operations in this matrix multiplication reduction are valid, because

$$\underbrace{\mathbf{D}^{-1}}_{M\times N}\underbrace{\mathbf{D}}_{N\times M} = \underbrace{\mathbf{I}}_{M\times M}. \tag{1.16}$$

But see the enrichment section[5] for a detailed discussion of other cases.

The most important thing for our present purposes is that if $M \leq N$ then we can find a solution of the over-specified (rectangular matrix) fitting problem $\mathbf{Sc} = \mathbf{y}$ as $\mathbf{c} = \mathbf{S}^{-1}\mathbf{y}$, using the pseudo-inverse. The set of coefficients $\mathbf{c}$ we get corresponds to more than one possible set of y_i-values, but that does not matter.

Also, it can be shown[6], that the specific solution that is obtained by this matrix product is in fact the *least-squares solution* for $\mathbf{c}$; i.e. the solution that minimizes the residual χ^2. And if there is any freedom in the choice of $\mathbf{c}$, such that the residual is at its minimum for a range of different $\mathbf{c}$, then the solution which minimizes $|\mathbf{c}|^2$ is the one found.

The beauty of this fact is that one can implement a simple code, which calls a function `pinv` to find the pseudo-inverse, and it will work just fine if the matrix $\mathbf{S}$ is singular or even rectangular.

As a matter of computational efficiency, it should be noted that in Octave the backslash operator is equivalent to multiplying by the pseudo-inverse (i.e. `pinv(S)*y = S\y`), but is calculated far more efficiently.[7] So backslash is preferable in computationally costly code, because it is roughly five times

[5] **Enrichment:** If $M > N$, the combination $\mathbf{DD}^{-1}$, which arises from forming $\mathbf{SS}^{-1}$, is not an $M \times M$ identity matrix. Instead it has ones only, at most, for the first N of the diagonal positions, and zeros thereafter. It is an $N \times N$ identity matrix with extra zero rows and columns padding it out to $M \times M$. So the pseudo-inverse is a funny kind of inverse, which works only one way.

If $\mathbf{S}$ is square and *non-singular*, then the pseudo-inverse is exactly the same as the (normal) inverse.

If $\mathbf{S}$ were a *singular* square matrix, for example (and possibly in other situations), then at least one of the original d_j would be zero. We usually consider the singular values (d_j) to be arranged in descending order of size; so that the zero values come at the end. $\mathbf{D}^{-1}$ would then have an element $1/d_j$ that is *infinite*, and the formal manipulations would be unjustified. What the pseudo-inverse does in these tricky cases is put the value of the inverse $1/d_j$ equal to zero instead of infinity. In that case, once again, an incomplete identity matrix is produced, with extra diagonal zeros at the end. And it actually doesn't completely "work" as an inverse in either direction.

For those who know some linear algebra, what's happening is that the pseudo-inverse projects vectors in the range of the original matrix back into the complement of its nullspace.

[6] See for example, first edition W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vettering (1989), *Numerical Recipes*, Cambridge University Press, Cambridge, (henceforth cited simply as *Numerical Recipes*), Section 2.9.

[7] By *QR* decomposition.

https://doi.org/10.1017/CBO9781316155516.002 Published online by Cambridge University Press

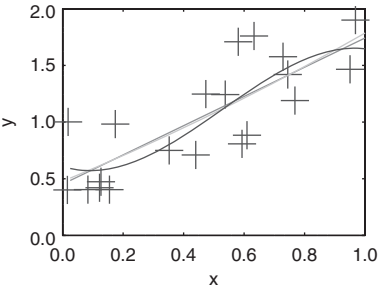

Figure 1.4 The cloud of points fitted with linear, quadratic, and cubic polynomials.

faster. You probably won't notice the difference for matrix dimensions smaller than a few hundred.

1.2.3 Smoothing and regularization

As we illustrate in Fig. 1.4, by choosing the number of degrees of freedom of the fitting function the smoothness of the fit can be adjusted to the data. However, the choice of basis functions then constrains one in a way that has been pre-specified. It might not in fact be the best way to smooth the data to fit them by (say) a straight line or a parabola.

A better way to smooth is by "regularization" in which we add some measure of roughness to the residual we are seeking to minimize. The roughness (which is the inverse of the smoothness) is a measure of how wiggly the fit line is. It can in principle be pretty much anything that can be written in the form of a matrix times the fit coefficients. I'll give an example in a moment. Let's assume the roughness measure is homogeneous, in the sense that we are trying to make it as near zero as possible. Such a target would be $\mathbf{Rc} = 0$, where $\mathbf{R}$ is a matrix of dimension $N_R \times M$, where N_R is the number of distinct roughness constraints. Presumably we can't satisfy this equation perfectly because a fully smooth function would have no variation, and be unable to fit the data. But we want to minimize the square of the roughness $(\mathbf{Rc})^T\mathbf{Rc}$. We can try to fulfil the requirement to fit the data, and to minimize the roughness, in a least-squares sense by constructing an expanded compound matrix system combining the original equations and the regularization; thus: [8]

[8] This notation means the first N rows of the compound matrix consist of $\mathbf{S}$, and the next N_R rows are $\lambda\mathbf{R}$.

https://doi.org/10.1017/CBO9781316155516.002 Published online by Cambridge University Press

$$\begin{pmatrix} \mathbf{S} \\ \lambda\mathbf{R} \end{pmatrix} \mathbf{c} = \begin{pmatrix} \mathbf{y} \\ \mathbf{0} \end{pmatrix}. \tag{1.17}$$

If we solve this system in a least-squares sense by using the pseudo-inverse of the compound matrix $\begin{pmatrix} \mathbf{S} \\ \lambda\mathbf{R} \end{pmatrix}$, then we will have found the coefficients that "best" make the roughness zero as well as fitting the data: in the sense that the total residual

$$\chi^2 = \sum_{i=1,N} (y_i - f(x_i))^2 + \lambda^2 \sum_{k=1,N_R} \left(\sum_j R_{kj} c_j \right)^2 \tag{1.18}$$

is minimized. The value of λ controls the weight of the smoothing. If it is large, then we prefer smoother solutions. If it is small or zero, we do negligible smoothing.

As a specific one-dimensional example, we might decide that the roughness we want to minimize is represented by the second derivative of the function: d^2f/dx^2. Making this quantity on average small has the effect of minimizing the wiggles in the function, so it is an appropriate roughness measure. We could therefore choose $\mathbf{R}$ such that it represented that derivative at a set of chosen points x_k, $k = 1, N_R$ (not the same as the data points x_i) in which case:

$$R_{kj} = \left. \frac{d^2 f_j}{dx^2} \right|_{x_k}. \tag{1.19}$$

The x_k might, for example, be equally spaced over the x-interval of interest, in which case[9] the squared roughness measure could be considered a discrete approximation to the integral, over the interval, of the quantity $(d^2f/dx^2)^2$.

1.3 Tomographic image reconstruction

Consider the problem of x-ray tomography. We make many measurements of the integrated density of matter along chords in a plane section through some object whose interior we wish to reconstruct. These are generally done by measuring the attenuation of x-rays along each chord, but the mathematical technique is independent of the physics. We seek a representation of the density

[9] This regularization is equivalent to what is sometimes called a "smoothing spline." In the limit of large smoothing parameter λ, the function f is a straight line (zero second derivative everywhere). In the limit of small λ, it is a cubic spline interpolation through all the values (x_i, y_i).

https://doi.org/10.1017/CBO9781316155516.002 Published online by Cambridge University Press

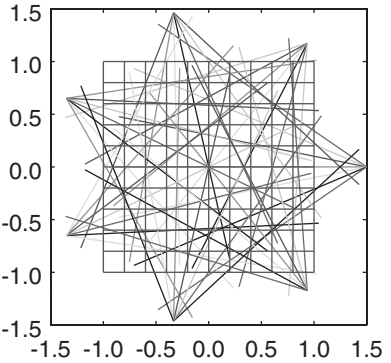

Figure 1.5 Illustrative layout of tomographic reconstruction of density in a plane using multiple fans of chordal observations.

of the object in the form

$$\rho(x,y) = \sum_{j=1,M} c_j \rho_j(x,y), \tag{1.20}$$

where $\rho_j(x,y)$ are basis functions over the plane. They might actually be as simple as pixels over mesh x_k and y_l, such that $\rho_j(x,y) \rightarrow \rho_{kl}(x,y) = 1$ when $x_k < x < x_{k+1}$ and $y_l < y < y_{l+1}$, and zero otherwise. However, the form of basis function that won A. M. Cormack the Nobel prize for medicine in his implementation of "computerized tomography" (the CT scan) was much more cleverly chosen to build the smoothing into the basis functions. Be careful thinking about multidimensional fitting. For constructing fitting matrices, the list of basis functions should be considered to be logically arranged from 1 to M in a single index j so that the coefficients are a single column vector. But the physical arrangement of the basis functions might more naturally be expressed using two indices k, l referring to the different spatial dimensions. If so then they must be mapped in some consistent manner to the vector column.

Each chord along which measurements are made passes through the basis functions (e.g. the pixels), and for a particular set of coefficients c_j we therefore get a chordal measurement value

$$v_i = \int_{l_i} \rho d\ell = \int_{l_i} \sum_{j=1,M} c_j \rho_j(x,y) d\ell = \sum_{j=1,M} \int_{l_i} \rho_j(x,y) d\ell \; c_j = \mathbf{Sc}, \tag{1.21}$$

where the $N \times M$ matrix $\mathbf{S}$ is formed from the integrals along each of the N lines of sight l_i, so that $S_{ij} = \int_{l_i} \rho_j(x,y) d\ell$. It represents the contribution of basis function j to measurement i. Our fitting problem is thus rendered into the

https://doi.org/10.1017/CBO9781316155516.002 Published online by Cambridge University Press

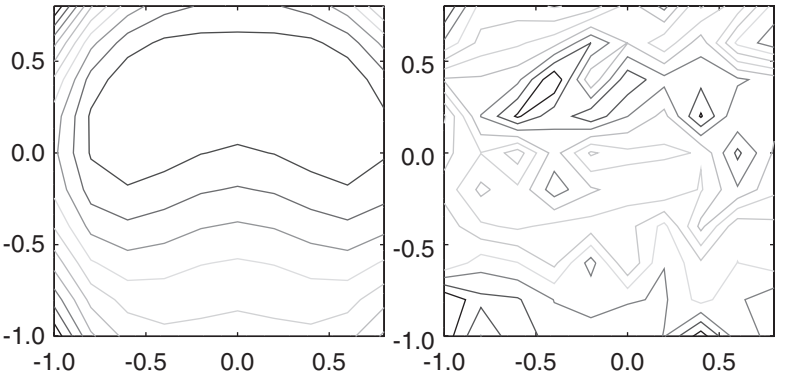

Figure 1.6 Contour plots of the initial test ρ-function (left) used to calculate the chordal integrals, and its reconstruction based upon inversion of the chordal data (right). The number of pixels (100) exceeds the number of views (49), and the number of singular values used in the pseudo-inverse is restricted to 30. Still they do not agree well, because various artifacts appear. Reducing the number of singular values does not help.

standard form:

$$\mathbf{Sc} = \mathbf{v}, \tag{1.22}$$

in which a rather large number M of basis functions might be involved. We can solve this by pseudo-inverse: $\mathbf{c} = \mathbf{S}^{-1}\mathbf{v}$, and, if the system is overdetermined such that the effective number of different chords is larger than the number of basis functions, it will probably work.

The problem is, however, usually *underdetermined*, in the sense that we don't really have enough independent chordal measurements to determine the density in each pixel (for example). This is true even if we apparently have more measurements than pixels, because generally there is a finite noise or uncertainty level in the chordal measurements that becomes amplified by the inversion process. This is illustrated by a simple test as shown in Fig. 1.6.

We then almost certainly want to smooth the representation otherwise all sorts of meaningless artifacts will appear in our reconstruction that have no physical existence. If we try to do this by forming a pseudo-inverse in which a smaller number of singular values are retained, and the others put to zero, there is no guarantee that this will get rid of the roughness. Fig. 1.6 gives an example.

If instead we smooth the reconstruction by regularization, using as our measure of roughness the discrete (two-dimensional) Laplacian ($\nabla^2\rho$) evaluated at each pixel, we get a far better result, as shown in Fig. 1.7. It turns out that this

https://doi.org/10.1017/CBO9781316155516.002 Published online by Cambridge University Press

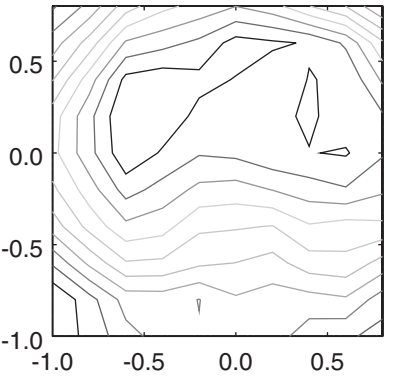

Figure 1.7 Reconstruction using a regularization smoothing based upon $\nabla^2\rho$. The contours are much nearer to reality.

good result is rather insensitive to the value of λ^2 over two or three orders of magnitude.

1.4 Efficiency and non-linearity

Using the inverse or pseudo-inverse to solve for the coefficients of a fitting function is intuitive and straightforward. However, in many cases it is *not* the most computationally efficient approach. For moderate size problems, modern computers have more than enough power to overcome the inefficiencies, but in a situation with multiple dimensions, such as tomography, it is easy for the matrix that needs to be inverted to become enormous, because that matrix's side length is the *total number* of pixels or elements in the fit, which may be, for example, the product of the side lengths `nx`×`ny`. The giant matrix that has to be inverted may be very "sparse," meaning that all but a very few of its elements are zero. It can then become overwhelming in terms of storage and cpu to use the direct inversion methods we have discussed here. We'll see other approaches later.

Some fitting problems are *non-linear*. For example, suppose one had a photon spectrum of a particular spectral line to which one wished to fit a Gaussian function of particular center, width, and height. That's a problem that cannot be expressed as a linear sum of functions. In that case fitting becomes more elaborate,[10] and less reliable. There are some potted fitting programs out there, but it's usually better if you can avoid them.

[10] This topic, and many others in data fitting, is addressed for example in S. Brandt (2014), *Data Analysis: Statistical and Computational Methods for Scientists and Engineers*, fourth edition, Springer, New York.

https://doi.org/10.1017/CBO9781316155516.002 Published online by Cambridge University Press

Worked example. Fitting sinusoidal functions

Suppose we wish to fit a set of data x_i, y_i spread over the range of independent variables $a \leq x \leq b$. And suppose we know the function is zero at the boundaries of the range, at $x = a$ and $x = b$. It makes sense to incorporate our knowledge of the boundary values into the choice of functions to fit, and choose those functions f_n to be zero at $x = a$ and $x = b$. There are numerous well-known sets of functions that have the property of being zero at two separated points. The points where standard functions are zero are of course not some arbitrary a and b. But we can scale the independent variable x so that a and b are mapped to the appropriate points for any choice of function set.

Suppose the functions that we decide to use for fitting are sinusoids:[11] $f_n = \sin(n\theta)$ all of which are zero at $\theta = 0$ and $\theta = \pi$. We can make this set fit our x range by using the scaling

$$\theta = \pi(x - a)/(b - a), \tag{1.23}$$

so that θ ranges from 0 to π as x ranges from a to b. Now we want to find the best fit to our data in the form

$$f(x) = c_1 \sin(\theta) + c_2 \sin(2\theta) + c_3 \sin(3\theta) + \cdots + c_M \sin(M\theta). \tag{1.24}$$

We therefore want the least-squares solution for the c_i of

$$\mathbf{Sc} = \begin{pmatrix} \sin(1\theta_1) & \sin(2\theta_1) & \dots & \sin(M\theta_1) \\ \sin(1\theta_2) & \sin(2\theta_2) & \dots & \sin(M\theta_2) \\ \dots & & & \\ \sin(1\theta_N) & \sin(2\theta_N) & \dots & \sin(M\theta_N) \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \dots \\ c_M \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_N \end{pmatrix} = \mathbf{y} \tag{1.25}$$

We find this solution by the following procedure.

1. If necessary, construct column vectors $\mathbf{x}$ and $\mathbf{y}$ from the data.
2. Calculate the scaled vector $\boldsymbol{\theta}$ from $\mathbf{x}$.
3. Construct the matrix $\mathbf{S}$ whose ijth entry is $\sin(j\theta_i)$
4. Least-squares-solve $\mathbf{Sc} = \mathbf{y}$ (e.g. by pseudo-inverse) to find $\mathbf{c}$.
5. Evaluate the fit at any x by substituting the expression for θ, eq. (1.23), into eq. (1.24).

This process may be programmed in a mathematical system like Matlab or Octave, which has built-in matrix multiplication, very concisely[12] as follows (entries following % are comments).

[11] Of course Fourier analysis is usually approached differently, as briefly discussed in the final chapter. We are here just using sine as an example function, in part because it is familiar.

[12] There are lots of other correct but more verbose ways of doing it.

https://doi.org/10.1017/CBO9781316155516.002 Published online by Cambridge University Press

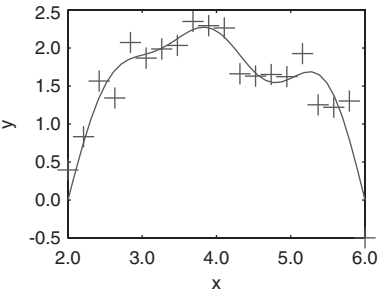

Figure 1.8 The result of the fit of sinusoids up to $M = 5$ to a noisy dataset of size $N = 20$. The points are the input data. The curve is constructed by using the `yfit` expression on an `xfit` array of some convenient length spanning the x-range, and then simply plotting `yfit` versus `xfit`.

```
  % Suppose x and y exist as column vectors of length N. (Nx1 matrices)
j=[1:M];                % Create a 1xM matrix containing numbers 1 to M.
theta=pi*(x-a)/(b-a);   % Scale x to obtain the column vector theta.
S=sin(theta*j);         % Construct the matrix S using an outer product.
Sinv=pinv(S);           % Pseudo invert it.
c=Sinv*y;               % Matrix multiply y to find the coefficients c.
```

The fit can then be evaluated for any x value (or array) `xfit`, in the form effectively of a scalar product of $\sin(\theta\mathbf{j})$ with $\mathbf{c}$. The code is likewise astonishingly brief, and will need careful thought (especially noting what the dimensions of the matrices are) to understand what is actually happening:

```
yfit=sin(pi*(xfit-a)/(b-a)*j)*c; % Evaluate the yfit at any xfit.
```

An example is shown in Fig. 1.8.

Exercise 1. Data fitting

1. Given a set of N values y_i of a function $y(x)$ at the positions x_i, write a short code to fit a polynomial having order one less than N (so there are N coefficients of the polynomial) to the data.

Obtain a set of ($N =$) 6 numbers from http://silas.psfc.mit.edu/22.15/15numbers.html (or if that is not accessible use $y_i = [0.892, 1.44, 1.31, 1.66, 1.10, 1.19]$). Take the values y_i to be at the positions $x_i = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]$. Run your code on these data and find the coefficients c_j.

Plot together (on the same plot) the resulting fitted polynomial representing $y(x)$ (with sufficient resolution to give a smooth curve) and the original data points, over the domain $0 \leq x \leq 1$.

https://doi.org/10.1017/CBO9781316155516.002 Published online by Cambridge University Press

Submit the following as your solution:

1. Your code in a computer format that is capable of being executed.
2. The numeric values of your coefficients c_j, $j = 1, N$.
3. Your plot.
4. Brief commentary ($<$ 300 words) on what problems you faced and how you solved them.

2. Save your code from part 1. Make a copy of it with a new name and change the new code as needed to fit (in the linear least-squares sense) a polynomial of order possibly lower than $N - 1$ to a set of data x_i, y_i (for which the points are in no particular order).

Obtain a pair of data sets of length ($N =$) 20 numbers x_i, y_i from the same URL by changing the entry in the "Number of Numbers" box. (Or, if that is inaccessible, generate your own data set from random numbers added to a line.) Run your code on these data to produce the fitting coefficients c_j when the number of coefficients of the polynomial is ($M =$) (a) 1, (b) 2, (c) 3. That is: constant, linear, quadratic.

Plot the fitted curves and the original data points on the same plot(s) for all three cases.

Submit the following as your solution:

1. Your code in a computer format that is capable of being executed.
2. Your coefficients c_j, $j = 1, M$, for three cases (a), (b), (c).
3. Your plot(s).
4. Very brief remarks on the extent to which the coefficients are the same for the three cases.
5. Can your code from this part also solve the problem of part 1?

https://doi.org/10.1017/CBO9781316155516.002 Published online by Cambridge University Press