Chapter 5

Least Squares

The term *least squares* describes a frequently used approach to solving overdetermined or inexactly specified systems of equations in an approximate sense. Instead of solving the equations exactly, we seek only to minimize the sum of the squares of the residuals.

The least squares criterion has important statistical interpretations. If appropriate probabilistic assumptions about underlying error distributions are made, least squares produces what is known as the *maximum-likelihood* estimate of the parameters. Even if the probabilistic assumptions are not satisfied, years of experience have shown that least squares produces useful results.

The computational techniques for linear least squares problems make use of orthogonal matrix factorizations.

5.1 Models and Curve Fitting

A very common source of least squares problems is curve fitting. Let t be the independent variable and let $y(t)$ denote an unknown function of t that we want to approximate. Assume there are m *observations*, i.e., values of y measured at specified values of t:

$$y_i = y(t_i),\ i = 1, \ldots, m.$$

The idea is to model $y(t)$ by a linear combination of n *basis functions*:

$$y(t) \approx \beta_1 \phi_1(t) + \ \cdots \ + \beta_n \phi_n(t).$$

The *design matrix* X is a rectangular matrix of order m by n with elements

$$x_{i,j} = \phi_j(t_i).$$

The design matrix usually has more rows than columns. In matrix-vector notation, the model is

$$y \approx X\beta.$$

The symbol $\approx$ stands for "is approximately equal to." We are more precise about this in the next section, but our emphasis is on *least squares* approximation.

The basis functions $\phi_j(t)$ can be nonlinear functions of t, but the unknown parameters, β_j, appear in the model linearly. The system of linear equations

$$X\beta \approx y$$

is *overdetermined* if there are more equations than unknowns. The MATLAB backslash operator computes a least squares solution to such a system.

```
beta = X\y
```

The basis functions might also involve some nonlinear parameters, $\alpha_1, \ldots, \alpha_p$. The problem is *separable* if it involves both linear and nonlinear parameters:

$$y(t) \approx \beta_1\phi_1(t,\alpha) + \ \cdots \ + \beta_n\phi_n(t,\alpha).$$

The elements of the design matrix depend upon both t and α:

$$x_{i,j} = \phi_j(t_i,\alpha).$$

Separable problems can be solved by combining backslash with the MATLAB function `fminsearch` or one of the nonlinear minimizers available in the Optimization Toolbox. The new Curve Fitting Toolbox provides a graphical interface for solving nonlinear fitting problems.

Some common models include the following:

- Straight line: If the model is also linear in t, it is a straight line:

$$y(t) \approx \beta_1 t + \beta_2.$$

- Polynomials: The coefficients β_j appear linearly. MATLAB orders polynomials with the highest power first:

$$\begin{aligned} \phi_j(t) &= t^{n-j}, \ j = 1, \ldots, n, \\ y(t) &\approx \beta_1 t^{n-1} + \cdots + \beta_{n-1}t + \beta_n. \end{aligned}$$

  The MATLAB function `polyfit` computes least squares polynomial fits by setting up the design matrix and using backslash to find the coefficients.

- Rational functions: The coefficients in the numerator appear linearly; the coefficients in the denominator appear nonlinearly:

$$\begin{aligned} \phi_j(t) &= \frac{t^{n-j}}{\alpha_1 t^{n-1} + \cdots + \alpha_{n-1}t + \alpha_n}, \\ y(t) &\approx \frac{\beta_1 t^{n-1} + \cdots + \beta_{n-1}t + \beta_n}{\alpha_1 t^{n-1} + \cdots + \alpha_{n-1}t + \alpha_n}. \end{aligned}$$

- Exponentials: The decay rates, λ_j, appear nonlinearly:

$$\begin{aligned} \phi_j(t) &= e^{-\lambda_j t}, \\ y(t) &\approx \beta_1 e^{-\lambda_1 t} + \cdots + \beta_n e^{-\lambda_n t}. \end{aligned}$$

- Log-linear: If there is only one exponential, taking logs makes the model linear but changes the fit criterion:

$$\begin{aligned} y(t) &\approx K e^{\lambda t}, \\ \log y &\approx \beta_1 t + \beta_2, \text{ with } \beta_1 = \lambda,\ \beta_2 = \log K. \end{aligned}$$

- Gaussians: The means and variances appear nonlinearly:

$$\begin{aligned} \phi_j(t) &= e^{-\left(\frac{t-\mu_j}{\sigma_j}\right)^2}, \\ y(t) &\approx \beta_1 e^{-\left(\frac{t-\mu_1}{\sigma_1}\right)^2} + \cdots + \beta_n e^{-\left(\frac{t-\mu_n}{\sigma_n}\right)^2}. \end{aligned}$$

5.2 Norms

The *residuals* are the differences between the observations and the model:

$$r_i = y_i - \sum_1^n \beta_j \phi_j(t_i, \alpha),\ i = 1, \ldots, m.$$

Or, in matrix-vector notation,

$$r = y - X(\alpha)\beta.$$

We want to find the α's and β's that make the residuals as small as possible. What do we mean by "small"? In other words, what do we mean when we use the '$\approx$' symbol? There are several possibilities.

- Interpolation: If the number of parameters is equal to the number of observations, we might be able to make the residuals zero. For linear problems, this will mean that $m = n$ and that the design matrix X is square. If X is nonsingular, the β's are the solution to a square system of linear equations:

$$\beta = X \backslash y.$$

- Least squares: Minimize the sum of the squares of the residuals:

$$\|r\|^2 = \sum_1^m r_i^2.$$

- Weighted least squares: If some observations are more important or more accurate than others, then we might associate different weights, w_j, with different observations and minimize

$$\|r\|_w^2 = \sum_1^m w_i r_i^2.$$

  For example, if the error in the ith observation is approximately e_i, then choose $w_i = 1/e_i$.

  Any algorithm for solving an unweighted least squares problem can be used to solve a weighted problem by scaling the observations and design matrix. We simply multiply both y_i and the ith row of X by w_i. In MATLAB, this can be accomplished with

```
X = diag(w)*X
y = diag(w)*y
```

- One-norm: Minimize the sum of the absolute values of the residuals:

$$\|r\|_1 = \sum_1^m |r_i|.$$

  This problem can be reformulated as a linear programming problem, but it is computationally more difficult than least squares. The resulting parameters are less sensitive to the presence of spurious data points or *outliers*.

- Infinity-norm: Minimize the largest residual:

$$\|r\|_\infty = \max_i |r_i|.$$

  This is also known as a Chebyshev fit and can be reformulated as a linear programming problem. Chebyshev fits are frequently used in the design of digital filters and in the development of approximations for use in mathematical function libraries.

The MATLAB Optimization and Curve Fitting Toolboxes include functions for one-norm and infinity-norm problems. We will limit ourselves to least squares in this book.

5.3 censusgui

The NCM program `censusgui` involves several different linear models. The data are the total population of the United States, as determined by the U.S. Census, for the years 1900 to 2000. The units are millions of people.

```
   t        y
1900     75.995
1910     91.972
1900    105.711
1930    123.203
1940    131.669
1950    150.697
1960    179.323
1970    203.212
1980    226.505
1990    249.633
2000    281.422
```

The task is to model the population growth and predict the population when $t = 2010$. The default model in `censusgui` is a cubic polynomial in t:

$$y \approx \beta_1 t^3 + \beta_2 t^2 + \beta_3 t + \beta_4.$$

There are four unknown coefficients, appearing linearly.

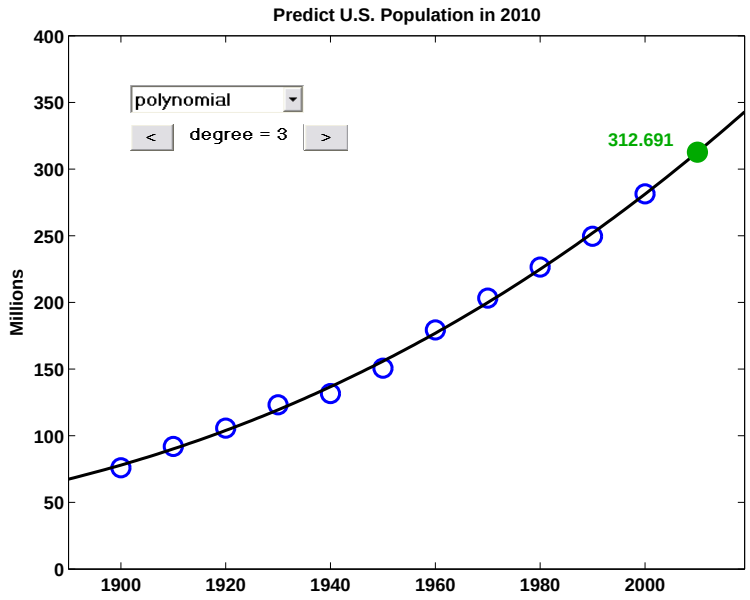

Figure 5.1. `censusgui`.

Numerically, it's a bad idea to use powers of t as basis functions when t is around 1900 or 2000. The design matrix is badly scaled and its columns are nearly linearly dependent. A much better basis is provided by powers of a translated and scaled t:

$$s = (t - 1950)/50.$$

This new variable is in the interval $-1 \le s \le 1$ and the model is

$$y \approx \beta_1 s^3 + \beta_2 s^2 + \beta_3 s + \beta_4.$$

The resulting design matrix is well conditioned.

Figure 5.1 shows the fit to the census data by the default cubic polynomial. The extrapolation to the year 2010 seems reasonable. The push buttons allow you to vary the degree of the polynomial. As the degree increases, the fit becomes more accurate in the sense that $\|r\|$ decreases, but it also becomes less useful because the variation between and outside the observations increases.

The `censusgui` menu also allows you to choose interpolation by `spline` and `pchip` and to see the log-linear fit

$$y \approx Ke^{\lambda t}.$$

Nothing in the `censusgui` tool attempts to answer the all-important question, "Which is the best model?" That's up to you to decide.

5.4 Householder Reflections

Householder reflections are matrix transformations that are the basis for some of the most powerful and flexible numerical algorithms known. We will use Householder reflections in this chapter for the solution of linear least squares problems and in a later chapter for the solution of matrix eigenvalue and singular value problems.

Formally, a Householder reflection is a matrix of the form

$$H = I - \rho u u^T,$$

where u is any nonzero vector and $\rho = 2/\|u\|^2$. The quantity uu^T is a matrix of rank one where every column is a multiple of u and every row is a multiple of u^T. The resulting matrix H is both symmetric and orthogonal, that is,

$$H^T = H$$

and

$$H^T H = H^2 = I.$$

In practice, the matrix H is never formed. Instead, the application of H to a vector x is computed by

$$\tau = \rho u^T x,$$
$$Hx = x - \tau u.$$

Geometrically, the vector x is projected onto u and then twice that projection is subtracted from x.

Figure 5.2 shows a vector u and a line labeled $u_\perp$ that is perpendicular to u. It also shows two vectors, x and y, and their images, Hx and Hy, under the

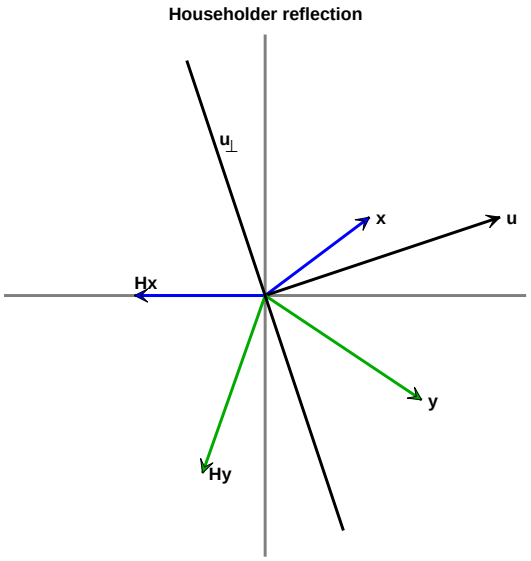

Figure 5.2. *Householder reflection.*

transformation H. The matrix transforms any vector into its mirror image in the line $u_\perp$. For any vector x, the point halfway between x and Hx, that is, the vector

$$x - (\tau/2)u,$$

is actually on the line $u_\perp$. In more than two dimensions, $u_\perp$ is the plane perpendicular to the defining vector u.

Figure 5.2 also shows what happens if u bisects the angle between x and one of the axes. The resulting Hx then falls on that axis. In other words, all but one of the components of Hx are zero. Moreover, since H is orthogonal, it preserves length. Consequently, the nonzero component of Hx is $\pm\|x\|$.

For a given vector x, the Householder reflection that zeros all but the kth component of x is given by

$$\begin{aligned}
\sigma &= \pm\|x\|, \\
u &= x + \sigma e_k, \\
\rho &= 2/\|u\|^2 = 1/(\sigma u_k), \\
H &= I - \rho u u^T.
\end{aligned}$$

In the absence of roundoff error, either sign could be chosen for σ, and the resulting Hx would be on either the positive or the negative kth axis. In the presence of roundoff error, it is best to choose the sign so that

$$\text{sign } \sigma = \text{sign } x_k.$$

Then the operation $x_k + \sigma$ is actually an addition, not a subtraction.

5.5 The QR Factorization

If all the parameters appear linearly and there are more observations than basis functions, we have a linear least squares problem. The design matrix X is m by n with $m > n$. We want to solve

$$X\beta \approx y.$$

But this system is overdetermined—there are more equations than unknowns. So we cannot expect to solve the system exactly. Instead, we solve it in the least squares sense:

$$\min_\beta \|X\beta - y\|.$$

A theoretical approach to solving the overdetermined system begins by multiplying both sides by X^T. This reduces the system to a square, n-by-n system known as the *normal equations*:

$$X^T X\beta = X^T y.$$

If there are thousands of observations and only a few parameters, the design matrix X is quite large, but the matrix $X^T X$ is small. We have projected y onto the space spanned by the columns of X. Continuing with this theoretical approach, if the basis functions are independent, then $X^T X$ is nonsingular and

$$\beta = (X^T X)^{-1} X^T y.$$

This formula for solving linear least squares problems appears in most textbooks on statistics and numerical methods. However, there are several undesirable aspects to this theoretical approach. We have already seen that using a matrix inverse to solve a system of equations is more work and less accurate than solving the system by Gaussian elimination. But, more importantly, the normal equations are always more badly conditioned than the original overdetermined system. In fact, the condition number is squared:

$$\kappa(X^T X) = \kappa(X)^2.$$

With finite-precision computation, the normal equations can actually become singular, and $(X^T X)^{-1}$ nonexistent, even though the columns of X are independent.

As an extreme example, consider the design matrix

$$X = \begin{pmatrix} 1 & 1 \\ \delta & 0 \\ 0 & \delta \end{pmatrix}.$$

If δ is small, but nonzero, the two columns of X are nearly parallel but are still linearly independent. The normal equations make the situation worse:

$$X^T X = \begin{pmatrix} 1+\delta^2 & 1 \\ 1 & 1+\delta^2 \end{pmatrix}.$$

If $|\delta| < 10^{-8}$, the matrix $X^T X$ computed with double-precision floating-point arithmetic is exactly singular and the inverse required in the classic textbook formula does not exist.

MATLAB avoids the normal equations. The backslash operator not only solves square, nonsingular systems, but it also computes the least squares solution to rectangular, overdetermined systems:

$$\beta = X \backslash y.$$

Most of the computation is done by an *orthogonalization* algorithm known as the QR factorization. The factorization is computed by the built-in function `qr`. The NCM function `qrsteps` demonstrates the individual steps.

The two versions of the QR factorization are illustrated in Figure 5.3. Both versions have

$$X = QR.$$

In the full version, R is the same size as X and Q is a square matrix with as many rows as X. In the economy-sized version, Q is the same size as X and R is a square matrix with as many columns as X. The letter "Q" is a substitute for the letter "O" in "orthogonal" and the letter "R" is for "right" triangular matrix. The Gram–Schmidt process described in many linear algebra texts is a related, but numerically less satisfactory, algorithm that generates the same factorization.

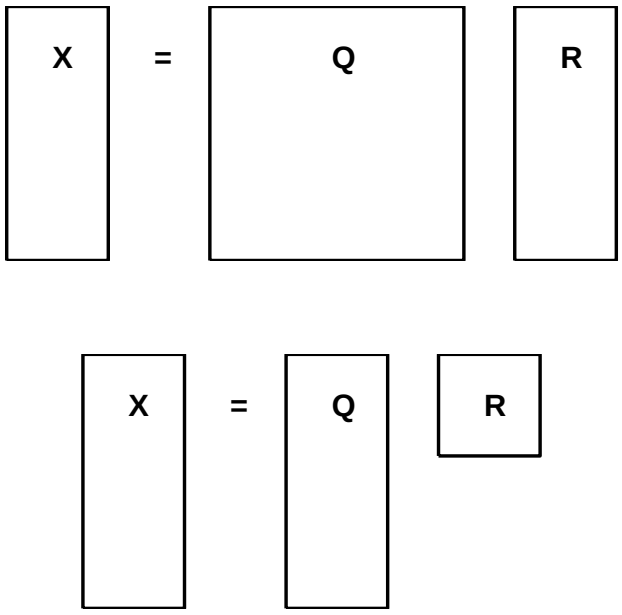

Figure 5.3. *Full and economy QR factorizations.*

A sequence of Householder reflections is applied to the columns of X to produce the matrix R:

$$H_n \cdots H_2 H_1 X = R.$$

The jth column of R is a linear combination of the first j columns of X. Consequently, the elements of R below the main diagonal are zero.

If the same sequence of reflections is applied to the right-hand side, the equations

$$X\beta \approx y$$

become

$$R\beta \approx z,$$

where

$$H_n \cdots H_2 H_1 y = z.$$

The first n of these equations is a small, square, triangular system that can be solved for β by back substitution with the subfunction `backsubs` in `bslashtx`. The coefficients in the remaining $m - n$ equations are all zero, so these equations are independent of β and the corresponding components of z constitute the transformed residual. This approach is preferable to the normal equations because Householder reflections have impeccable numerical credentials and because the resulting triangular system is ready for back substitution.

The matrix Q in the QR factorization is

$$Q = (H_n \cdots H_2 H_1)^T.$$

To solve least squares problems, we do not have to actually compute Q. In other uses of the factorization, it may be convenient to have Q explicitly. If we compute just the first n columns, we have the economy-sized factorization. If we compute all m columns, we have the full factorization. In either case,

$$Q^T Q = I,$$

so Q has columns that are perpendicular to each other and have unit length. Such a matrix is said to have *orthonormal columns*. For the full Q, it is also true that

$$QQ^T = I,$$

so the full Q is an *orthogonal* matrix.

Let's illustrate this with a small version of the census example. We will fit the last six observations with a quadratic:

$$y(s) \approx \beta_1 s^2 + \beta_2 s + \beta_3.$$

The scaled time `s = ((1950:10:2000)' - 1950)/50` and the observations `y` are

```
     s        y
0.0000  150.6970
0.2000  179.3230
0.4000  203.2120
0.6000  226.5050
0.8000  249.6330
1.0000  281.4220
```

The design matrix is `X = [s.*s s ones(size(s))]`.

```
         0         0    1.0000
    0.0400    0.2000    1.0000
    0.1600    0.4000    1.0000
    0.3600    0.6000    1.0000
    0.6400    0.8000    1.0000
    1.0000    1.0000    1.0000
```

The M-file `qrsteps` shows the steps in the QR factorization.

```
qrsteps(X,y)
```

The first step introduces zeros below the diagonal in the first column of `X`.

```
   -1.2516   -1.4382   -1.7578
         0    0.1540    0.9119
         0    0.2161    0.6474
         0    0.1863    0.2067
         0    0.0646   -0.4102
         0   -0.1491   -1.2035
```

The same Householder reflection is applied to y.

```
-449.3721
 160.1447
 126.4988
  53.9004
 -57.2197
-198.0353
```

Zeros are introduced in the second column.

```
   -1.2516   -1.4382   -1.7578
         0   -0.3627   -1.3010
         0         0   -0.2781
         0         0   -0.5911
         0         0   -0.6867
         0         0   -0.5649
```

The second Householder reflection is also applied to y.

```
-449.3721
-242.3136
 -41.8356
 -91.2045
-107.4973
 -81.8878
```

Finally, zeros are introduced in the third column and the reflection applied to y. This produces the triangular matrix `R` and a modified right-hand side `z`.

```
R =
  -1.2516   -1.4382   -1.7578
        0   -0.3627   -1.3010
        0         0    1.1034
        0         0         0
        0         0         0
        0         0         0
z =
-449.3721
-242.3136
 168.2334
  -1.3202
  -3.0801
   4.0048
```

The system of equations $R\beta = z$ is the same size as the original, 6 by 3. We can solve the first three equations exactly (because $R(1:3, 1:3)$ is nonsingular).

```
beta = R(1:3,1:3)\z(1:3)

beta =
    5.7013
  121.1341
  152.4745
```

This is the same solution `beta` that the backslash operator computes with

```
beta = R\z
```

or

```
beta = X\y
```

The last three equations in $R\beta = z$ cannot be satisfied by any choice of β, so the last three components of `z` represent the residual. In fact, the two quantities

```
norm(z(4:6))
norm(X*beta - y)
```

are both equal to `5.2219`. Notice that even though we used the QR factorization, we never actually computed Q.

The population in the year 2010 can be predicted by evaluating

$$\beta_1 s^2 + \beta_2 s + \beta_3$$

at $s = (2010 - 1950)/50 = 1.2$. This can be done with `polyval`.

```
p2010 = polyval(beta,1.2)

p2010 =
  306.0453
```

`Censusgui` itself, fitting a quadratic to more data, predicts `311.5880`. Which do you think is going to be closer to the actual result of the 2010 census?

5.6 Pseudoinverse

The definition of the pseudoinverse makes use of the Frobenius norm of a matrix:

$$\|A\|_F = \left(\sum_i \sum_j a_{i,j}^2 \right)^{1/2}.$$

The MATLAB expression `norm(X,'fro')` computes the Frobenius norm. $\|A\|_F$ is the same as the 2-norm of the long vector formed from all the elements of A.

```
norm(A,'fro') = norm(A(:))
```

The *Moore–Penrose pseudoinverse* generalizes and extends the usual matrix inverse. The pseudoinverse is denoted by a dagger superscript,

$$Z = X^\dagger,$$

and computed by the MATLAB `pinv`.

```
Z = pinv(X)
```

If X is square and nonsingular, then the pseudoinverse and the inverse are the same:

$$X^\dagger = X^{-1}.$$

If X is m by n with $m > n$ and X has full rank, then its pseudoinverse is the matrix involved in the normal equations:

$$X^\dagger = (X^T X)^{-1} X^T.$$

The pseudoinverse has some, but not all, of the properties of the ordinary inverse. $X^\dagger$ is a left inverse because

$$X^\dagger X = (X^T X)^{-1} X^T X = I$$

is the n-by-n identity. But $X^\dagger$ is not a right inverse because

$$X X^\dagger = X (X^T X)^{-1} X^T$$

only has rank n and so cannot be the m-by-m identity.

The pseudoinverse does get as close to a right inverse as possible in the sense that, out of all the matrices Z that minimize

$$\|XZ - I\|_F,$$

$Z = X^\dagger$ also minimizes

$$\|Z\|_F.$$

It turns out that these minimization properties also define a unique pseudoinverse even if X is rank deficient.

Consider the 1-by-1 case. What is the inverse of a real (or complex) number x? If x is not zero, then clearly $x^{-1} = 1/x$. But if x is zero, x^{-1} does not exist. The pseudoinverse takes care of that because, in the scalar case, the unique number that minimizes both

$$|xz - 1| \text{ and } |z|$$

is

$$x^{\dagger} = \begin{cases} 1/x & : \quad x \neq 0, \\ 0 & : \quad x = 0. \end{cases}$$

The actual computation of the pseudoinverse involves the singular value decomposition, which is described in a later chapter. You can `edit pinv` or `type pinv` to see the code.

5.7 Rank Deficiency

If X is rank deficient, or has more columns than rows, the square matrix $X^T X$ is singular and $(X^T X)^{-1}$ does not exist. The formula

$$\beta = (X^T X)^{-1} X^T y$$

obtained from the normal equations breaks down completely.

In these degenerate situations, the least squares solution to the linear system

$$X\beta \approx y$$

is not unique. A *null vector* of X is a nonzero solution to

$$X\eta = 0.$$

Any multiple of any null vector can be added to β without changing how well $X\beta$ approximates y.

In MATLAB, the solution to

$$X\beta \approx y$$

can be computed with either backslash or the pseudoinverse, that is,

```
beta = X\y
```

or

```
beta = pinv(X)*y
```

In the full rank case, these two solutions are the same, although `pinv` does considerably more computation to obtain it. But in degenerate situations these two solutions are not the same.

The solution computed by backslash is called a *basic* solution. If r is the rank of X, then at most r of the components of

```
beta = X\y
```

are nonzero. Even the requirement of a basic solution does not guarantee uniqueness. The particular basic computation obtained with backslash is determined by details of the QR factorization.

The solution computed by `pinv` is the *minimum norm* solution. Out of all the vectors β that minimize $\|X\beta - y\|$, the vector

```
beta = pinv(X)*y
```

also minimizes $\|\beta\|$. This minimum norm solution is unique.

For example, let

$$X = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \\ 13 & 14 & 15 \end{pmatrix}$$

and

$$y = \begin{pmatrix} 16 \\ 17 \\ 18 \\ 19 \\ 20 \end{pmatrix}.$$

The matrix X is rank deficient. The middle column is the average of the first and last columns. The vector

$$\eta = \begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix}$$

is a null vector.

The statement

```
beta = X\y
```

produces a warning,

```
Warning: Rank deficient, rank = 2  tol =   2.4701e-014.
```

and the solution

```
beta =
   -7.5000
         0
    7.8333
```

As promised, this solution is basic; it has only two nonzero components. However, the vectors

```
beta =
         0
  -15.0000
   15.3333
```

and

```
beta =
  -15.3333
   15.6667
         0
```

are also basic solutions.

The statement

```
beta = pinv(X)*y
```

produces the solution

```
beta =
   -7.5556
    0.1111
    7.7778
```

without giving a warning about rank deficiency. The norm of the pseudoinverse solution

```
norm(pinv(X)*y) = 10.8440
```

is slightly less than the norm of the backslash solution

```
norm(X\y) = 10.8449
```

Out of all the vectors β that minimize $\|X\beta - y\|$, the pseudoinverse has found the shortest. Notice that the difference between the two solutions,

```
X\y - pinv(X)*y =
    0.0556
   -0.1111
    0.0556
```

is a multiple of the null vector η.

If handled with care, rank deficient least squares problems can be solved in a satisfactory manner. Problems that are nearly, but not exactly, rank deficient are more difficult to handle. The situation is similar to square linear systems that are badly conditioned, but not exactly singular. Such problems are not *well posed* numerically. Small changes in the data can lead to large changes in the computed solution. The algorithms used by both backslash and pseudoinverse involve decisions about linear independence and rank. These decisions use somewhat arbitrary tolerances and are particularly susceptible to both errors in the data and roundoff errors in the computation.

Which is "better," backslash or pseudoinverse? In some situations, the underlying criteria of basic solution or minimum norm solution may be relevant. But most problem formulations, particularly those involving curve fitting, do not include such subtle distinctions. The important fact to remember is that the computed solutions are not unique and are not well determined by the data.

5.8 Separable Least Squares

MATLAB provides several functions for solving nonlinear least squares problems. Older versions of MATLAB have one general-purpose, multidimensional, nonlinear minimizer, `fmins`. In more recent versions of MATLAB, `fmins` has been updated and renamed `fminsearch`. The Optimization Toolbox provides additional capabilities, including a minimizer for problems with constraints, `fmincon`; a minimizer for unconstrained problems, `fminunc`; and two functions intended specifically for nonlinear least squares, `lsqnonlin` and `lsqcurvefit`. The Curve Fitting Toolbox provides a GUI to facilitate the solution of many different linear and nonlinear fitting problems.

In this introduction, we focus on the use of `fminsearch`. This function uses a direct search technique known as the Nelder–Meade algorithm. It does not attempt to approximate any gradients or other partial derivatives. It is quite effective on small problems involving only a few variables. Larger problems with more variables are better handled by the functions in the Optimization and Curve Fitting Toolboxes.

Separable least squares curve-fitting problems involve both linear and nonlinear parameters. We could ignore the linear portion and use `fminsearch` to search for all the parameters. But if we take advantage of the separable structure, we obtain a more efficient, robust technique. With this approach, `fminsearch` is used to search for values of the nonlinear parameters that minimize the norm of the residual. At each step of the search process, the backslash operator is used to compute values of the linear parameters.

Two blocks of MATLAB code are required. One block can be a function, a script, or a few lines typed directly in the Command Window. It sets up the problem, establishes starting values for the nonlinear parameters, calls `fminsearch`, processes the results, and usually produces a plot. The second block is the objective function that is called by `fminsearch`. This function is given a vector of values of the nonlinear parameters, `alpha`. It should compute the design matrix `X` for these parameters, use backslash with `X` and the observations to compute values of the linear parameters `beta`, and return the resulting residual norm.

Let's illustrate all this with `expfitdemo`, which involves observations of radioactive decay. The task is to model the decay by a sum of two exponential terms with unknown rates λ_j:

$$y \approx \beta_1 e^{-\lambda_1 t} + \beta_2 e^{-\lambda_2 t}.$$

Consequently, in this example, there are two linear parameters and two nonlinear parameters. The demo plots the various fits that are generated during the nonlinear minimization process. Figure 5.4 shows plots of both the data and the final fit.

The main function begins by specifying 21 observations, t and y.

```
function expfitdemo
t = (0:.1:2)';
y = [5.8955 3.5639 2.5173 1.9790 1.8990 1.3938 1.1359 ...
     1.0096 1.0343 0.8435 0.6856 0.6100 0.5392 0.3946 ...
     0.3903 0.5474 0.3459 0.1370 0.2211 0.1704 0.2636]';
```

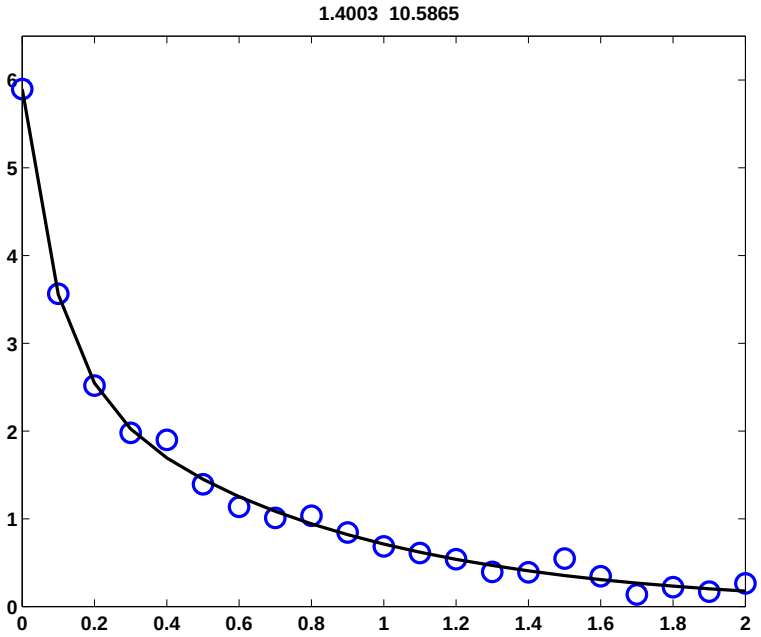

Figure 5.4. `expfitdemo`.

The initial plot uses `o`'s for the observations, creates an all-zero placeholder for what is going to become the evolving fit, and creates a title that will show the values of `lambda`. The variable `h` holds the handles for these three graphics objects.

```
clf
shg
set(gcf,'doublebuffer','on')
h = plot(t,y,'o',t,0*t,'-');
h(3) = title('');
axis([0 2 0 6.5])
```

The vector `lambda0` specifies initial values for the nonlinear parameters. In this example, almost any choice of initial values leads to convergence, but in other situations, particularly with more nonlinear parameters, the choice of initial values can be much more important. The call to `fminsearch` does most of the work. The observations `t` and `y`, as well as the graphics handle `h`, are passed as extra parameters.

```
lambda0 = [3 6]';
lambda = fminsearch(@expfitfun,lambda0,[],t,y,h)
set(h(2),'color','black')
```

The objective function is named `expfitfun`. It can handle n exponential basis functions; we will be using $n = 2$. The first input parameter is a vector

provided by `fminsearch` that contains values of the n decay rates, λ_j. The other parameters are vectors containing the independent and dependent variables, t and y, and the graphics handle. The function computes the design matrix, uses backslash to compute β, evaluates the resulting model, and returns the norm of the residual.

```
function res = expfitfun(lambda,t,y,h)
m = length(t);
n = length(lambda);
X = zeros(m,n);
for j = 1:n
   X(:,j) = exp(-lambda(j)*t);
end
beta = X\y;
z = X*beta;
res = norm(z-y);
```

The objective function also updates the plot of the fit and the title and pauses long enough for us to see the progress of the computation.

```
set(h(2),'ydata',z);
set(h(3),'string',sprintf('%8.4f %8.4f',lambda))
pause(.1)
```

5.9 Further Reading

The reference books on matrix computation [4, 5, 6, 7, 8, 9] discuss least squares. An additional reference is Björck [1].

Exercises

5.1. Let `X` be the `n`-by-`n` matrix generated by

```
[I,J] = ndgrid(1:n);
X = min(I,J) + 2*eye(n,n) - 2;
```

(a) How does the condition number of `X` grow with `n`?
(b) Which, if any, of the triangular factorizations `chol(X)`, `lu(X)`, and `qr(X)` reveal the poor conditioning?

5.2. In `censusgui`, change the 1950 population from 150.697 million to 50.697 million. This produces an extreme *outlier* in the data. Which models are the most affected by this outlier? Which models are the least affected?

5.3. If `censusgui` is used to fit the U.S. Census data with a polynomial of degree eight and the fit is extrapolated beyond the year 2000, the predicted population actually becomes zero before the year 2020. On what year, month, and day does that fateful event occur?

5.4. Here are some details that we skipped over in our discussion of Householder reflections. At the same time, we extend the description to include complex matrices. The notation u^T for transpose is replaced with the MATLAB notation u' for complex conjugate transpose. Let x be any nonzero m-by-1 vector and let e_k denote the kth unit vector, that is, the kth column of the m-by-m identity matrix. The sign of a complex number $z = re^{i\theta}$ is

$$\text{sign}(z) = z/|z| = e^{i\theta}.$$

Define σ by

$$\sigma = \text{sign}(x_k)\|x\|.$$

Let

$$u = x + \sigma e_k.$$

In other words, u is obtained from x by adding σ to its kth component.
(a) The definition of ρ uses $\bar{\sigma}$, the complex conjugate of σ:

$$\rho = 1/(\bar{\sigma} u_k).$$

Show that

$$\rho = 2/\|u\|^2.$$

(b) The Householder reflection generated by the vector x is

$$H = I - \rho u u'.$$

Show that

$$H' = H$$

and that

$$H'H = I.$$

(c) Show that all the components of Hx are zero, except for the kth. In other words, show that

$$Hx = -\sigma e_k.$$

(d) For any vector y, let

$$\tau = \rho u' y.$$

Show that

$$Hy = y - \tau u.$$

5.5. Let

$$x = \begin{pmatrix} 9 \\ 2 \\ 6 \end{pmatrix}.$$

(a) Find the Householder reflection H that transforms x into

$$Hx = \begin{pmatrix} -11 \\ 0 \\ 0 \end{pmatrix}.$$

(b) Find nonzero vectors u and v that satisfy

$$Hu = -u,$$
$$Hv = v.$$

5.6. Let

$$X = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \\ 13 & 14 & 15 \end{pmatrix}.$$

(a) Verify that X is rank deficient.

Consider three choices for the pseudoinverse of `X`.

```
Z = pinv(X)       % The actual pseudoinverse
B = X\eye(5,5)    % Backslash
S = eye(3,3)/X    % Slash
```

(b) Compare the values of

$$\|Z\|_F, \quad \|B\|_F, \quad \text{and } \|S\|_F;$$
$$\|XZ - I\|_F, \quad \|XB - I\|_F, \quad \text{and } \|XS - I\|_F;$$
$$\|ZX - I\|_F, \quad \|BX - I\|_F, \quad \text{and } \|SX - I\|_F.$$

Verify that the values obtained with Z are less than or equal to the values obtained with the other two choices. Actually minimizing these quantities is one way of characterizing the pseudoinverse.
(c) Verify that Z satisfies all four of the following conditions, and that B and S fail to satisfy at least one of the conditions. These conditions are known as the Moore–Penrose equations and are another way to characterize a unique pseudoinverse.

XZ is symmetric.
ZX is symmetric.
$XZX = X$.
$ZXZ = Z$.

5.7. Generate 11 data points, $t_k = (k-1)/10$, $y_k = \text{erf}(t_k)$, $k = 1, \ldots, 11$.
(a) Fit the data in a least squares sense with polynomials of degrees 1 through 10. Compare the fitted polynomial with $\text{erf}(t)$ for values of t between the data points. How does the maximum error depend on the polynomial degree?
(b) Because $\text{erf}(t)$ is an odd function of t, that is, $\text{erf}(x) = -\text{erf}(-x)$, it is reasonable to fit the data by a linear combination of odd powers of t:

$$\text{erf}(t) \approx c_1 t + c_2 t^3 + \cdots + c_n t^{2n-1}.$$

Again, see how the error between data points depends on n.

(c) Polynomials are not particularly good approximants for $\mathrm{erf}(t)$ because they are unbounded for large t, whereas $\mathrm{erf}(t)$ approaches 1 for large t. So, using the same data points, fit a model of the form

$$\mathrm{erf}(t) \approx c_1 + e^{-t^2}(c_2 + c_3 z + c_4 z^2 + c_5 z^3),$$

where $z = 1/(1+t)$. How does the error between the data points compare with the polynomial models?

5.8. Here are 25 observations, y_k, taken at equally spaced values of t.

```
t = 1:25
y = [ 5.0291   6.5099   5.3666   4.1272   4.2948
      6.1261  12.5140  10.0502   9.1614   7.5677
      7.2920  10.0357  11.0708  13.4045  12.8415
     11.9666  11.0765  11.7774  14.5701  17.0440
     17.0398  15.9069  15.4850  15.5112  17.6572]
y = y';
y = y(:);
```

(a) Fit the data with a straight line, $y(t) = \beta_1 + \beta_2 t$, and plot the residuals, $y(t_k) - y_k$. You should observe that one of the data points has a much larger residual than the others. This is probably an *outlier*.
(b) Discard the outlier, and fit the data again by a straight line. Plot the residuals again. Do you see any pattern in the residuals?
(c) Fit the data, with the outlier excluded, by a model of the form

$$y(t) = \beta_1 + \beta_2 t + \beta_3 \sin t.$$

(d) Evaluate the third fit on a finer grid over the interval $[0, 26]$. Plot the fitted curve, using line style `'-'`, together with the data, using line style `'o'`. Include the outlier, using a different marker, `'*'`.

5.9. *Statistical Reference Datasets.* NIST, the National Institute of Standards and Technology, is the branch of the U.S. Department of Commerce responsible for setting national and international standards. NIST maintains Statistical Reference Datasets, StRD, for use in testing and certifying statistical software. The home page on the Web is [3]. Data sets for linear least squares are under "Linear Regression." This exercise involves two of the NIST reference data sets:

- `Norris`: linear polynomial for calibration of ozone monitors;
- `Pontius`: quadratic polynomial for calibration of load cells.

For each of these data sets, follow the Web links labeled

- Data File (ASCII Format),
- Certified Values, and
- Graphics.

Download each ASCII file. Extract the observations. Compute the polynomial coefficients. Compare the coefficients with the certified values. Make plots similar to the NIST plots of both the fit and the residuals.

5.10. *Filip data set.* One of the Statistical Reference Datasets from the NIST is the "Filip" data set. The data consist of several dozen observations of a variable y at different values of x. The task is to model y by a polynomial of degree 10 in x.

This data set is controversial. A search of the Web for "filip strd" will find several dozen postings, including the original page at NIST [3]. Some mathematical and statistical packages are able to reproduce the polynomial coefficients that NIST has decreed to be the "certified values." Other packages give warning or error messages that the problem is too badly conditioned to solve. A few packages give different coefficients without warning. The Web offers several opinions about whether or not this is a reasonable problem. Let's see what MATLAB does with it.

The data set is available from the NIST Web site. There is one line for each data point. The data are given with the first number on the line a value of y, and the second number the corresponding x. The x-values are not monotonically ordered, but it is not necessary to sort them. Let n be the number of data points and $p = 11$ the number of polynomial coefficients.

(a) As your first experiment, load the data into MATLAB, plot it with `'.'` as the line type, and then invoke the **Basic Fitting** tool available under the **Tools** menu on the figure window. Select the 10th-degree polynomial fit. You will be warned that the polynomial is badly conditioned, but ignore that for now. How do the computed coefficients compare with the certified values on the NIST Web page? How does the plotted fit compare with the graphic on the NIST Web page? The basic fitting tool also displays the norm of the residuals, $\|r\|$. Compare this with the NIST quantity "Residual Standard Deviation," which is

$$\frac{\|r\|}{\sqrt{n-p}}.$$

(b) Examine this data set more carefully by using six different methods to compute the polynomial fit. Explain all the warning messages you receive during these computations.

- Polyfit: Use `polyfit(x,y,10)`.
- Backslash: Use `X\y`, where X is the n-by-p truncated Vandermonde matrix with elements

$$X_{i,j} = x_i^{p-j},\ i = 1, \ldots, n,\ j = 1, \ldots, p.$$

- Pseudoinverse: Use `pinv(X)*y`.
- Normal equations: Use `inv(X'*X)*X'*y`.
- Centering: Let $\mu = \text{mean}(x),\ \sigma = \text{std}(x),\ t = (x - \mu)/\sigma$. Use `polyfit(t,y,10)`.

- Certified coefficients: Obtain the coefficients from the NIST Web page.

(c) What are the norms of the residuals for the fits computed by the six different methods?
(d) Which one of the six methods gives a very poor fit? (Perhaps the packages that are criticized on the Web for reporting bad results are using this method.)
(e) Plot the five good fits. Use dots, `'.'`, at the data values and curves obtained by evaluating the polynomials at a few hundred points over the range of the x's. The plot should look like Figure 5.5. There are five different plots, but only two visually distinct ones. Which methods produce which plots?
(f) Why do polyfit and backslash give different results?

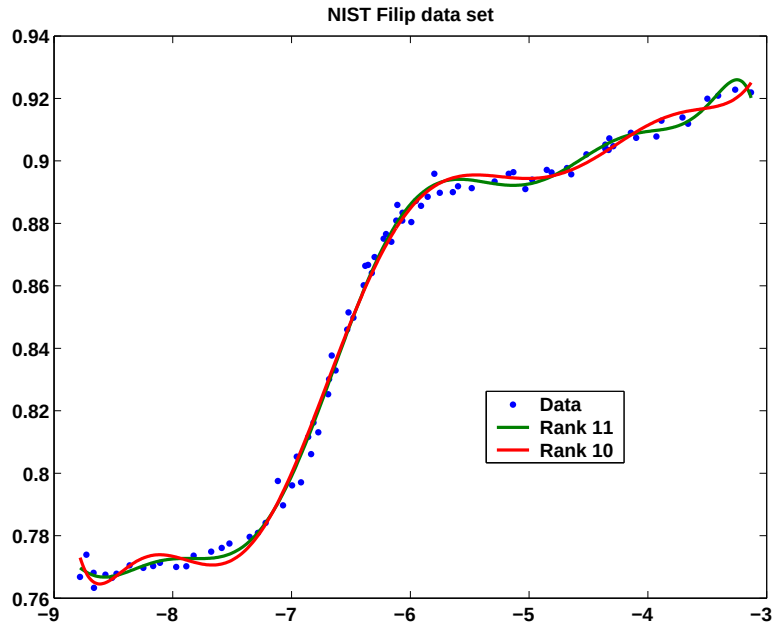

Figure 5.5. *NIST Filip standard reference data set.*

5.11. *Longley data set.* The Longley data set of labor statistics was one of the first used to test the accuracy of least squares computations. You don't need to go to the NIST Web site to do this problem, but if you are interested in the background, you should see the Longley page at [3]. The data set is available in `NCM` in the file `longley.dat`. You can bring the data into MATLAB with

```
load longley.dat
y = longley(:,1);
X = longley(:,2:7);
```

There are 16 observations of 7 variables, gathered over the years 1947 to 1962. The variable y and the 6 variables making up the columns of the data matrix

X are

y = Total Derived Employment,
x_1 = GNP Implicit Price Deflater,
x_2 = Gross National Product,
x_3 = Unemployment,
x_4 = Size of Armed Forces,
x_5 = Noninstitutional Population Age 14 and Over,
x_6 = Year.

The objective is to predict y by a linear combination of a constant and the six x's:

$$y \approx \beta_0 + \sum_1^6 \beta_k x_k.$$

(a) Use the MATLAB backslash operator to compute $\beta_0, \beta_1, \ldots, \beta_6$. This involves augmenting X with a column of all 1's, corresponding to the constant term.
(b) Compare your β's with the certified values [3].
(c) Use `errorbar` to plot y with error bars whose magnitude is the difference between y and the least squares fit.
(d) Use `corrcoef` to compute the correlation coefficients for X without the column of 1's. Which variables are highly correlated?
(e) Normalize the vector y so that its mean is zero and its standard deviation is one. You can do this with

```
y = y - mean(y);
y = y/std(y)
```

Do the same thing to the columns of X. Now plot all seven normalized variables on the same axis. Include a `legend`.

5.12. Planetary orbit [2]. The expression $z = ax^2+bxy+cy^2+dx+ey+f$ is known as a *quadratic form*. The set of points (x, y), where $z = 0$, is a *conic section*. It can be an ellipse, a parabola, or a hyperbola, depending on the sign of the discriminant $b^2 - 4ac$. Circles and lines are special cases. The equation $z = 0$ can be normalized by dividing the quadratic form by any nonzero coefficient. For example, if $f \neq 0$, we can divide all the other coefficients by f and obtain a quadratic form with the constant term equal to one. You can use the MATLAB `meshgrid` and `contour` functions to plot conic sections. Use `meshgrid` to create arrays `X` and `Y`. Evaluate the quadratic form to produce `Z`. Then use `contour` to plot the set of points where `Z` is zero.

```
[X,Y] = meshgrid(xmin:deltax:xmax,ymin:deltay:ymax);
Z = a*X.^2 + b*X.*Y + c*Y.^2 + d*X + e*Y + f;
contour(X,Y,Z,[0 0])
```

A planet follows an elliptical orbit. Here are ten observations of its position in the (x, y) plane:

```
x = [1.02 .95 .87 .77 .67 .56 .44 .30 .16 .01]';
y = [0.39 .32 .27 .22 .18 .15 .13 .12 .13 .15]';
```

(a) Determine the coefficients in the quadratic form that fits these data in the least squares sense by setting one of the coefficients equal to one and solving a 10-by-5 overdetermined system of linear equations for the other five coefficients. Plot the orbit with x on the x-axis and y on the y-axis. Superimpose the ten data points on the plot.
(b) This least squares problem is nearly rank deficient. To see what effect this has on the solution, perturb the data slightly by adding to each coordinate of each data point a random number uniformly distributed in the interval $[-.0005, .0005]$. Compute the new coefficients resulting from the perturbed data. Plot the new orbit on the same plot with the old orbit. Comment on your comparison of the sets of coefficients and the orbits.

Bibliography

[1] A. Björck, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, 1996.

[2] M. T. Heath, *Scientific Computing: An Introductory Survey*, McGraw–Hill, New York, 1997.

[3] National Institute of Standards and Technology, *Statistical Reference Datasets*.
`http://www.itl.nist.gov/div898/strd`
`http://www.itl.nist.gov/div898/strd/lls/lls.shtml`
`http://www.itl.nist.gov/div898/strd/lls/data/Longley.shtml`

[4] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, Third Edition, SIAM, Philadelphia, 1999.
`http://www.netlib.org/lapack`

[5] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[6] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Third Edition, The Johns Hopkins University Press, Baltimore, 1996.

[7] G. W. Stewart, *Introduction to Matrix Computations*, Academic Press, New York, 1973.

[8] G. W. Stewart, *Matrix Algorithms: Basic Decompositions*, SIAM, Philadelphia, 1998.

[9] L. N. Trefethen and D. Bau, III, *Numerical Linear Algebra*, SIAM, Philadelphia, 1997.