Chapter 6

Linear Least Squares Problems

This chapter is concerned with methods for solving the algebraic problem

$$\min_{\mathbf{x}} \|\mathbf{b} - A\mathbf{x}\|_2,$$

where the dimensions of the real matrix and vectors are

$$A \in \mathcal{R}^{m\times n},\ \mathbf{x} \in \mathcal{R}^n,\ \mathbf{b} \in \mathcal{R}^m,\ m \geq n.$$

We assume that A has full column rank. Notice that in the overdetermined case, $m > n$, there is typically no $\mathbf{x}$ satisfying $A\mathbf{x} = \mathbf{b}$ exactly, even in the absence of roundoff error.

The least squares problem arises often in many diverse application fields, especially where **data fitting** is required. Instances arise in machine learning, computer vision, and computer graphics applications, to name but a few. In computer vision people may want to match local invariant features of cluttered images under arbitrary rotations, scalings, change of brightness and contrast, and so on. Such actions are parametrized and the best values for the parameters are then sought to match the data. In *computer graphics* a parametrization of a surface mesh in three dimensions may be sought that yields a *morphing* of one animal into another, say. In such applications the question of just how to parametrize, or how to generate an efficient predictive model A in the present terminology, is often the crux of the matter and is far from trivial. But once the method of parametrization is determined, what follows is an instance of the problem considered here. Other applications seek to find an approximating function $v(t,\mathbf{x})$ depending on a continuous variable t that fits data pairs (t_i, b_i), $i = 1,\ldots,m$, as in Example 4.16. We will apply our solution techniques to such instances in Section 6.1.

In Section 6.1 we formulate an $n \times n$ system of linear equations called the **normal equations** that yield the solution for the stated minimization problem. Solving the normal equations is a fast and straightforward approach. However, the resulting algorithm is not as stable as can be in general. **Orthogonal transformations** provide a more stable way to proceed, and this is described in Section 6.2. Algorithms to actually carry out the necessary **QR decomposition** are described in Section 6.3.

6.1 Least squares and the normal equations

In this section we first solve the least squares minimization problem, obtaining the *normal equations*. Then we show examples related to *data fitting*.

The space spanned by the columns of our $m \times n$ matrix A (i.e., all vectors $\mathbf{z}$ of the form $\mathbf{z} = A\mathbf{y}$) is generally of dimension at most n, and we further assume that the dimension equals n, so that A has full column rank. In other words, we assume that its columns are linearly independent. Notice also that in the overdetermined case, $m > n$, $\mathbf{b}$ generally does not lie in the range space of A.

Deriving the normal equations

Let us rewrite the problem that we aim to solve as

$$\min_{\mathbf{x}} \frac{1}{2}\|\mathbf{b} - A\mathbf{x}\|^2.$$

We have squared the normed expression, thus getting rid of the square root sign,[25] and multiplied it by $1/2$: this will not change the minimizer, in the sense that the same solution coefficients x_j will be obtained. Notice that we have dropped the subscript 2 in the norm notation: there is no other norm here to get confused by. Finally, we define the residual vector as usual by

$$\mathbf{r} = \mathbf{b} - A\mathbf{x}.$$

Writing these matrix and vectors explicitly we have

$$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \quad A = \begin{pmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & & \vdots \\ \vdots & \dots & \vdots \\ a_{m,1} & \dots & a_{m,n} \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1 \\ \vdots \\ \vdots \\ b_m \end{pmatrix}, \quad \mathbf{r} = \begin{pmatrix} r_1 \\ \vdots \\ \vdots \\ r_m \end{pmatrix}.$$

Note that the matrix A is $m \times n$, with $m > n$ and perhaps $m \gg n$, so it is "long and skinny," and correspondingly we do not expect $\mathbf{r}$ to vanish at the optimum. See Figure 6.1.

We have a minimization problem for a smooth scalar function in several variables, given by

$$\min_{\mathbf{x}} \psi(\mathbf{x}), \quad \text{where} \quad \psi(\mathbf{x}) = \frac{1}{2}\|\mathbf{r}\|^2.$$

The *necessary conditions* for a minimum are obtained by setting the derivatives of ψ with respect to each unknown x_k to zero, yielding

$$\frac{\partial}{\partial x_k}\psi(\mathbf{x}) = 0, \quad k = 1, \dots, n.$$

Since

$$\psi(\mathbf{x}) = \frac{1}{2}\|\mathbf{r}\|^2 = \frac{1}{2}\sum_{i=1}^{m}\left(b_i - \sum_{j=1}^{n} a_{i,j}x_j\right)^2,$$

[25] If you are not sure you remember everything you need to remember about norms, Section 4.2 may come to your aid.

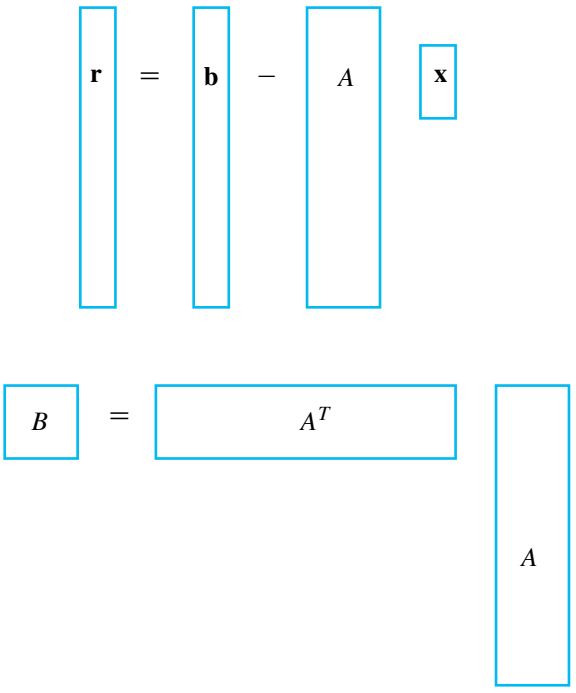

Figure 6.1. *Matrices and vectors and their dimensions in ℓ_2 data fitting.*

the conditions for a minimum yield

$$\frac{\partial}{\partial x_k}\psi(\mathbf{x}) = \sum_{i=1}^{m}\left[\left(b_i - \sum_{j=1}^{n} a_{i,j}x_j\right)(-a_{i,k})\right] = 0$$

for $k = 1,2,\ldots,n$. The latter expression can be rewritten as

$$\sum_{i=1}^{m} a_{i,k}\sum_{j=1}^{n} a_{i,j}x_j = \sum_{i=1}^{m} a_{i,k}b_i \qquad \text{for } k = 1,\ldots,n.$$

In matrix-vector form this expression looks much simpler; it reads

$$A^TA\mathbf{x} = A^T\mathbf{b}.$$

This system of n linear equations in n unknowns is called the *normal equations*. Note that $B = A^TA$ can be much smaller in size than A; see Figure 6.1. The matrix B is symmetric positive definite given that A has full column rank; see Section 4.3.

Least squares solution uniqueness

Is the solution of the normal equations really a minimizer (and not, say, a maximizer) of the least squares norm? and if yes, is it the *global* minimum? The Least Squares Theorem given on the next page says it is. The answer is affirmative because the matrix B is positive definite.

To see this (completing the proof of the theorem), note first that our objective function is a quadratic in n variables. Indeed, we can write

$$\psi(\mathbf{x}) = \frac{1}{2}(\mathbf{b}-A\mathbf{x})^T(\mathbf{b}-A\mathbf{x}) = \frac{1}{2}\mathbf{x}^TB\mathbf{x} - \mathbf{b}^TA\mathbf{x} + \frac{1}{2}\|\mathbf{b}\|^2.$$

Theorem: Least Squares.
The least squares problem

$$\min_{\mathbf{x}} \|A\mathbf{x}-\mathbf{b}\|_2,$$

where A has full column rank, has a unique solution that satisfies the normal equations

$$\left(A^TA\right)\mathbf{x} = A^T\mathbf{b}.$$

Expanding this quadratic function in a Taylor series (see page 259) around the alleged minimizer amounts to the usual Taylor expansion in each variable: for any nonzero vector increment $\triangle\mathbf{x}$ we write

$$\psi(\mathbf{x}+\triangle\mathbf{x}) = \psi(\mathbf{x}) + \sum_{j=1}^{n} \triangle x_j \underbrace{\frac{\partial\psi}{\partial x_j}(\mathbf{x})}_{=0} + \frac{1}{2}\sum_{j=1}^{n}\sum_{k=1}^{n} \triangle x_j \triangle x_k \underbrace{\frac{\partial^2\psi}{\partial x_j \partial x_k}(\mathbf{x})}_{=B_{j,k}}.$$

Hence, in vector notation we obtain

$$\psi(\mathbf{x}+\triangle\mathbf{x}) = \psi(\mathbf{x}) + (\triangle\mathbf{x})^T B\triangle\mathbf{x} > \psi(\mathbf{x}),$$

completing the proof. ♦

Solving via the normal equations

It is important to realize that $\mathbf{x}$, which has been our vector argument above, is now specified as the solution for the normal equations, which is indeed the solution of the least squares minimization problem.

Furthermore, if A has full column rank, then $B = A^TA$ is symmetric positive definite. The least squares problem has therefore been reduced, at least in principle, to that of solving an $n \times n$ system of linear equations of the form described in Section 5.5.

The beauty does not end here, geometrically speaking. Notice that for the corresponding residual at optimum we have

$$A^T(\mathbf{b}-A\mathbf{x}) = A^T\mathbf{r} = \mathbf{0}.$$

Hence we seek a solution satisfying that *the residual is orthogonal to the column space of A*. Since a picture is better than a thousand words (or at least a thousand bytes), Figure 6.2 is provided to illustrate the projection.

Our solution is given by $\mathbf{x} = (A^TA)^{-1}A^T\mathbf{b}$. The matrix multiplying $\mathbf{b}$ is important enough to have a name and special notation: it is called the **pseudo-inverse** of A, denoted by

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

The solution via the normal equations using direct solvers amounts to the four steps laid out in the algorithm given on the facing page.

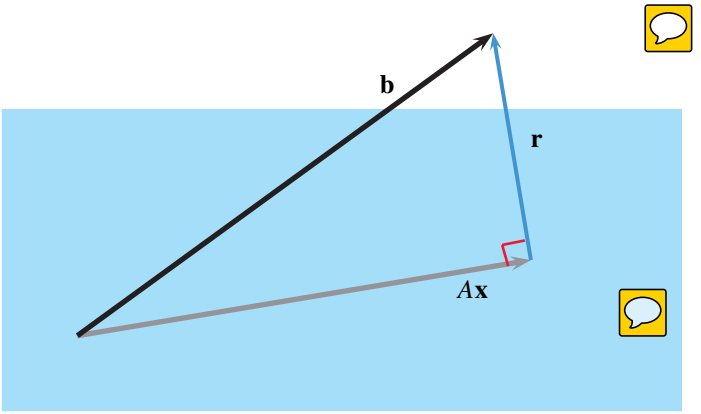

Figure 6.2. *Discrete least squares approximation.*

Algorithm: Least Squares via Normal Equations.

1. Form $B = A^T A$ and $\mathbf{y} = A^T \mathbf{b}$.
2. Compute the Cholesky Factor, i.e., the lower triangular matrix G satisfying $B = GG^T$.
3. Solve the lower triangular system $G\mathbf{z} = \mathbf{y}$ for $\mathbf{z}$.
4. Solve the upper triangular system $G^T\mathbf{x} = \mathbf{z}$ for $\mathbf{x}$.

Example 6.1. Let's see some numbers. Choose $m = 5$, $n = 3$, and specify

$$A = \begin{pmatrix} 1 & 0 & 1 \\ 2 & 3 & 5 \\ 5 & 3 & -2 \\ 3 & 5 & 4 \\ -1 & 6 & 3 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 4 \\ -2 \\ 5 \\ -2 \\ 1 \end{pmatrix}.$$

We wish to solve the least squares problem $\min_{\mathbf{x}} \|\mathbf{b} - A\mathbf{x}\|$.

Following the normal equations route we calculate

$$B = A^T A = \begin{pmatrix} 40 & 30 & 10 \\ 30 & 79 & 47 \\ 10 & 47 & 55 \end{pmatrix}, \quad \mathbf{y} = A^T \mathbf{b} = \begin{pmatrix} 18 \\ 5 \\ -21 \end{pmatrix}.$$

Now solving the square system $B\mathbf{x} = \mathbf{y}$ yields the final result $\mathbf{x} = (.3472, .3990, -.7859)^T$, correct to the number of digits shown.

Here are more numbers. The optimal residual (rounded) is

$$\mathbf{r} = \mathbf{b} - A\mathbf{x} = (4.4387, .0381, .495, -1.893, 1.311)^T.$$

This vector is orthogonal to each column of A. Finally, the pseudo-inverse of A (rounded) is

$$A^\dagger = B^{-1}A^T = \begin{pmatrix} .049 & .0681 & .1049 & .0531 & -.1308 \\ -.0491 & -.0704 & .0633 & .0114 & .1607 \\ .0512 & .1387 & -.1095 & .0533 & -.059 \end{pmatrix}.$$

As with the inverse of a square nonsingular matrix, we do not recommend explicitly calculating such a matrix, and will not do so again in this book.

Please see if you can write a seven-line MATLAB script verifying all these calculations. ∎

The overall computational work for the first step of the algorithm is approximately mn^2 floating point operations (flops); for step 2 it is $n^3/3 + \mathcal{O}(n^2)$ flops, while steps 3 and 4 cost $\mathcal{O}(n^2)$ flops. This is another case where although operation counts are in general unsatisfactory for measuring true performance, they do deliver the essential result that the main cost here, especially when $m \gg n$, is in *forming* the matrix $B = A^T A$.

Data fitting

Generally, data fitting problems arise as follows. We have *observed data* $\mathbf{b}$ and a model function that for any candidate model $\mathbf{x}$ provides *predicted data*. The task is to find $\mathbf{x}$ such that the predicted data match the observed data to the extent possible, by minimizing their difference in the least squares sense. In the linear case which we study here, the predicted data are given by $A\mathbf{x}$. In this context the assumption that A has full column rank does not impose a serious restriction: it just implies that there is no redundancy in the representation of the predicted data, so that for any vector $\hat{\mathbf{x}} \in \mathcal{R}^n$ there is no other vector $\tilde{\mathbf{x}} \in \mathcal{R}^n$ such that $A\tilde{\mathbf{x}} = A\hat{\mathbf{x}}$.

Example 6.2 (linear regression). Consider fitting a given data set of m pairs (t_i, b_i) by a straight line. Thus, we want to find the coefficients x_1 and x_2 of

$$v(t) = x_1 + x_2 t,$$

such that $v(t_i) \approx b_i$, $i = 1, \ldots, m$. So $n = 2$ here, and

$$A = \begin{pmatrix} 1 & t_1 \\ 1 & t_2 \\ \vdots & \vdots \\ 1 & t_m \end{pmatrix}.$$

The components of the normal equations are

$$B_{1,1} = \sum_{i=1}^{m} 1 = m, \quad B_{1,2} = \sum_{i=1}^{m} 1 \cdot t_i = \sum_{i=1}^{m} t_i,$$

$$B_{2,1} = B_{1,2}, \quad B_{2,2} = \sum_{i=1}^{m} t_i \cdot t_i = \sum_{i=1}^{m} t_i^2,$$

$$y_1 = \sum_{i=1}^{m} b_i, \quad y_2 = \sum_{i=1}^{m} t_i b_i.$$

This leads to a system of just two equations given by

$$\begin{aligned} m x_1 + \left(\sum_{i=1}^{m} t_i\right) x_2 &= \sum_{i=1}^{m} b_i, \\ \left(\sum_{i=1}^{m} t_i\right) x_1 + \left(\sum_{i=1}^{m} t_i^2\right) x_2 &= \sum_{i=1}^{m} t_i b_i. \end{aligned}$$

The solution is written explicitly as the famous formula

$$\begin{aligned} x_1 &= \frac{\sum_{i=1}^m t_i^2 \sum_{i=1}^m b_i \; - \; \sum_{i=1}^m t_i b_i \sum_{i=1}^m t_i}{m \sum_{i=1}^m t_i^2 \; - \; \left(\sum_{i=1}^m t_i\right)^2}, \\ x_2 &= \frac{m \sum_{i=1}^m t_i b_i \; - \; \sum_{i=1}^m t_i \sum_{i=1}^m b_i}{m \sum_{i=1}^m t_i^2 \; - \; \left(\sum_{i=1}^m t_i\right)^2}. \end{aligned}$$

For readers who like to see examples with numbers, consider the data

i	1	2	3
t_i	0.0	1.0	2.0
b_i	0.1	0.9	2.0

Then $m = 3$, $\sum_{i=1}^m t_i = 0.0 + 1.0 + 2.0 = 3.0$, $\sum_{i=1}^m b_i = 0.1 + 0.9 + 2.0 = 3.0$, $\sum_{i=1}^m t_i^2 = 5.0$, $\sum_{i=1}^m t_i b_i = 0.9 + 4.0 = 4.9$. This yields (please verify) the solution

$$x_1 = 0.05, \;\; x_2 = 0.95.$$

One purpose of this example is to convince you of the beauty of matrix-vector notation and computer programs as an alternative to the above exposition.

A more common regression curve would of course be constructed for many more data points. See Figure 6.3 for a plot of a regression curve obtained with these formulas where $m = 25$. Here $t_i = i < t_{i+1}$ denotes years since 1980, and b_i is the average change in temperature off the base of 1 in a certain secret location. Although it does not always hold that $b_i < b_{i+1}$, the straight line fit has a positive slope, which indicates a clear warming trend. ∎

While in a typical course on statistics for the social sciences the regression formulas of Example 8.1 appear as if by magic, here they are a simple by-product of a general treatment—albeit, without the statistical significance.

Polynomial data fitting

Extending the linear regression formulas to a higher degree polynomial fit, $v(t) \equiv p_{n-1}(t) = x_1 + x_2 t + \cdots + x_n t^{n-1}$, is straightforward too. Writing for each data point $v(t_i) = x_1 + x_2 t_i + \cdots + x_n t_i^{n-1} = (1, t_i, \ldots, t_i^{n-1})\,\mathbf{x}$, the matrix A is the extension of the previously encountered **Vander-**

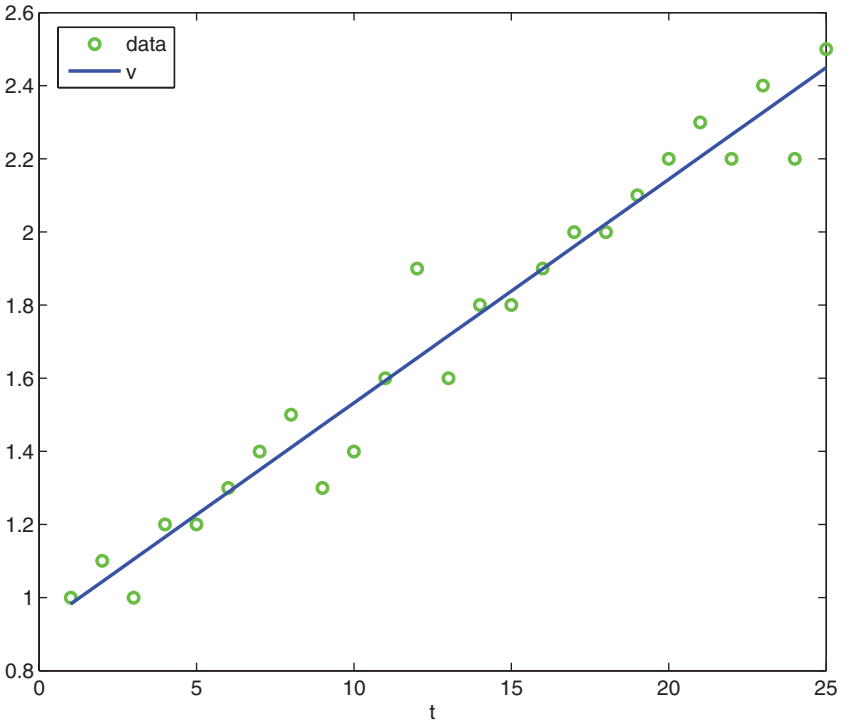

Figure 6.3. *Linear regression curve (in blue) through green data. Here,* $m = 25$, $n = 2$.

monde matrix given by

$$A = \begin{pmatrix} 1 & t_1 & {t_1}^2 & \cdots & {t_1}^{n-1} \\ 1 & t_2 & {t_2}^2 & \cdots & {t_2}^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & t_{m-1} & {t_{m-1}}^2 & \cdots & {t_{m-1}}^{n-1} \\ 1 & t_m & {t_m}^2 & \cdots & {t_m}^{n-1} \end{pmatrix}.$$

Note that the structure of the matrix A depends on our choice of the basis functions used to describe polynomials. More on this is provided in Chapter 10. For now let us just say that the simple basis used here, called *monomial basis*, is good only for polynomials of a really low degree. But then these are the ones we may want in the present context anyway.

Here is a MATLAB function for best approximation by low order polynomials, using the normal equations:

```
function coefs = lsfit (t, b, n)
%
% function coefs = lsfit (t, b, n)
%
```

```
% Construct coefficients of the polynomial of
% degree at most n-1 that best fits data (t,b)

t = t(:); b = b(:); % make sure t and b are column vectors
m = length(t);

% long and skinny A
A = ones(m,n);
for j=1:n-1
  A(:,j+1) = A(:,j).*t;
end

% normal equations and solution
B = A'*A;  y = A'*b;
coefs = B \ y;
```


Example 6.3. Sample the function $f(t) = \cos(2\pi t)$ at 21 equidistant points on the interval $[0,1]$ and construct best fits by polynomials of degree at most $n-1$ for each $n = 1,2,3,4,5$.

Following is an appropriate MATLAB script:

```
% data
m = 21;
tt = 0:1/(m-1):1;
bb = cos(2*pi*tt);

% find polynomial coefficients
for n=1:5
  coefs{n} = lsfit(tt,bb,n);
end

% Evaluate and plot
t = 0:.01:1;
z = ones(5,101);
for n=1:5
  z(n,:) = z(n,:) * coefs{n}(n);
  for j=n-1:-1:1
      z(n,:) = z(n,:).*t + coefs{n}(j);
  end
end
plot(t,z,tt,bb,'ro')
xlabel('t')
ylabel('p_{n-1}')
```

The resulting approximants $p_{n-1}(t)$ are plotted in Figure 6.4. Note that here, due to symmetry, $p_1(t) \equiv p_0(t)$ and $p_3(t) \equiv p_2(t)$. So, the degree "at most $n-1$" turns out to be equal to $n-2$ rather than to $n-1$ for odd values of $n-1$. ∎

Data fitting vs. interpolation

Reflecting on Examples 6.3 and 4.16 (page 85), there is a seeming paradox hidden in our arguments, namely, that we attempt to minimize the residual for a fixed n, $n < m$, but refuse to simply increase n until $n = m$. Indeed, this would drive the residual to zero, the resulting scheme being a polynomial

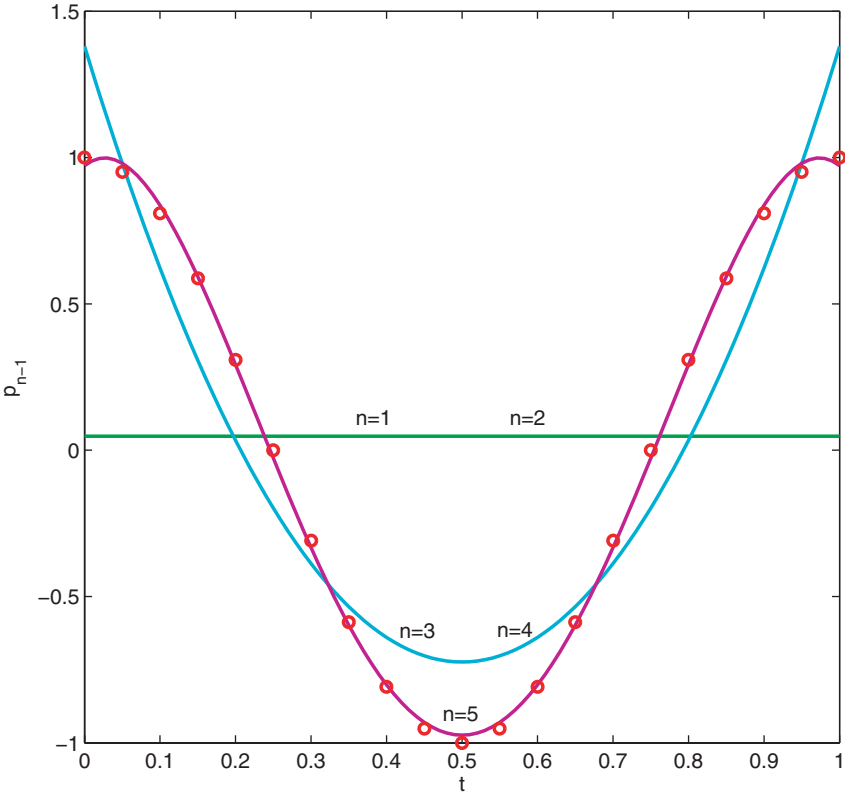

Figure 6.4. *The first* 5 *best polynomial approximations to* $f(t) = \cos(2\pi t)$ *sampled at* $0 : .05 : 1$. *The data values appear as red circles. Clearly,* p_4 *fits the data better than* p_2, *which in turn is a better approximation than* p_0. *Note* $p_{2j+1} = p_{2j}$.

that interpolates the data. Why not just interpolate?! (Interpolation techniques are discussed at length in Chapter 10.)

The simple answer is that choosing n is part of our *modeling* efforts, and the ensuing least squares minimization problem is part of the *solution process* with n already fixed. But there are reasons for choosing n small in the first place. One is hinted at in Example 6.2, namely, that we are trying to find the **trend** in the data on a long time scale.

An additional reason for not interpolating the data values is that they may contain measurement errors. Moreover, we may want a model function that depends only on a few parameters x_j for ease of manipulation, although we determine them based on all the given data. For a poignant example we refer the reader to Exercise 3.

The solution of the least squares data fitting problem through solving the normal equations has the advantage of being straightforward and efficient. A linear least squares solver is implemented in the MATLAB backslash operator as well as in the MATLAB command `polyfit`. Replacing the last three lines in our function `lsfit` by the line

```
coefs = A \ b;
```

would implement for the same purpose, albeit more enigmatically, an algorithm which in terms of roundoff error accumulation is at least as good. See Section 6.2. The routine `polyfit` is even

easier to use, as it does not require forming the matrix and the right-hand-side vector; the input consists of the data points and the degree of the required polynomial.

Data fitting in other norms

Before moving on let us also mention data fitting in other norms.

1. Using ℓ_1 we consider
$$\min_{\mathbf{x}} \|\mathbf{b} - A\mathbf{x}\|_1.$$
Here we would be looking at finding coefficients $\mathbf{x}$ such that the sum of absolute values of the deviations is minimized. This norm is particularly useful if we need to automatically get rid of an undue influence of *outliers* in the data, which are data values that conspicuously deviate from the rest due to measurement error.

2. Using ℓ_∞ we consider
$$\min_{\mathbf{x}} \|\mathbf{b} - A\mathbf{x}\|_\infty.$$
This is a *min-max* problem of finding the minimum over $x_1, x_2, \ldots, x_n$ of the maximum data deviation. This norm is useful if the worst-case error in the approximation is important and must be kept in check.

Both ℓ_1 and ℓ_∞ best approximations lead to **linear programming** problems, briefly discussed in the more advanced Section 9.3. They are significantly more complex than the problem that we are faced with using least squares yet are very important in many modern areas of application.

Note that if $n = m$, then we have one and the same solution regardless of the norm used, because with all three norms the minimum is obtained with $\mathbf{x} = A^{-1}\mathbf{b}$, which yields a zero residual. But when $n < m$ these different norms usually yield significantly different best approximations.

The least squares approximation (which is the approximation associated with the ℓ_2-norm) is not only the simplest to calculate, it also has a variety of beautiful mathematical properties. We continue to concentrate on it in the remainder of this chapter.

Specific exercises for this section: Exercises 1–4.

6.2 Orthogonal transformations and QR

The main drawback of the normal equations approach for solving linear least squares problems is accuracy in the presence of large condition numbers. Information may be lost when forming A^TA when $\kappa(A^TA)$ is very large. In this section we derive better algorithms in this respect.

But first let us investigate the source of difficulty.

Example 6.4. This example demonstrates the possible inaccuracy that arises when the normal equations are formed. Given

$$A = \begin{pmatrix} 1 & 1 & 1 \\ \varepsilon & 0 & 0 \\ 0 & \varepsilon & 0 \\ 0 & 0 & \varepsilon \end{pmatrix},$$

we have

$$A^T A = \begin{pmatrix} 1+\varepsilon^2 & 1 & 1 \\ 1 & 1+\varepsilon^2 & 1 \\ 1 & 1 & 1+\varepsilon^2 \end{pmatrix}.$$

Now, if $\eta < \varepsilon \leq \sqrt{\eta}$, where η is the rounding unit, then the evaluated $A^T A$ is numerically singular even though A has numerically full column rank. ∎

Troubling as this little example may appear to be, it is yet unclear who is to blame: is the *problem* to be solved simply more ill-conditioned, or is it the *algorithm* that insists on forming $A^T A$ that is responsible for the effective singularity? To see things better, we now consider a perturbation analysis as in Section 5.8, sticking to the 2-norm for vectors and matrices alike.

Condition number by SVD

Let us assume that $\mathbf{b}$ is in the range space of A, so for the exact least squares solution $\mathbf{x}$ we have

$$\mathbf{b} = A\mathbf{x}, \quad \mathbf{r} = \mathbf{0}.$$

Note that we will not use this to define a new algorithm (that would be cheating); rather, we seek an error estimate that is good also when $\|\mathbf{r}\|$ is small, and we will only use the bound

$$\|\mathbf{b}\| \leq \|A\|\|\mathbf{x}\|.$$

Next, for a given approximate solution $\hat{\mathbf{x}}$ we can calculate $\hat{\mathbf{b}} = A\hat{\mathbf{x}}$ and

$$\hat{\mathbf{r}} = \mathbf{b} - A\hat{\mathbf{x}} = \mathbf{b} - \hat{\mathbf{b}} = A(\mathbf{x} - \hat{\mathbf{x}}).$$

Thus, $\|\mathbf{x} - \hat{\mathbf{x}}\| \leq \|A^\dagger\|\|\hat{\mathbf{r}}\|$ and, corresponding to the bound (5.1) on page 128, we get

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \kappa(A)\frac{\|\hat{\mathbf{r}}\|}{\|\mathbf{b}\|}, \quad \kappa(A) = \|A\|\|A^\dagger\|.$$

To obtain an expression for the condition number $\kappa(A) = \kappa_2(A)$, recall from Section 4.4 (page 81) that A enjoys having an SVD, so there are orthogonal matrices U and V such that

$$A = U\Sigma V^T.$$

In the present case (with $\text{rank}(A) = n \leq m$) we can write

$$\Sigma = \begin{pmatrix} S \\ 0 \end{pmatrix},$$

with S a square diagonal matrix having the singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0$ on its main diagonal.

Now, recall that a square matrix Q is *orthogonal* if its columns are orthonormal, i.e., $Q^T Q = I$. It is straightforward to verify that for any vector $\mathbf{y}$ we have in the ℓ_2-norm $\|Q\mathbf{y}\| = \|\mathbf{y}\|$, because

$$\|Q\mathbf{y}\|^2 = (Q\mathbf{y})^T(Q\mathbf{y}) = \mathbf{y}^T Q^T Q\mathbf{y} = \mathbf{y}^T\mathbf{y} = \|\mathbf{y}\|^2.$$

(If the last couple of sentences look terribly familiar to you, it may be because we have mentioned this in Section 4.3.) The important point here is that multiplication by an orthogonal matrix does not change the norm. Thus, we simply have

$$\|A\| = \|\Sigma\| = \sigma_1.$$

Furthermore, $A^TA = V\Sigma^T U^T U\Sigma V^T = VS^2V^T$, so for the pseudo-inverse we get

$$A^\dagger = (A^TA)^{-1}A^T = VS^{-2}V^TV\Sigma^TU^T = V(S^{-1}\ 0)U^T.$$

This yields $\|A^\dagger\| = \|(S^{-1}\ 0)\| = \frac{1}{\sigma_n}$, and hence the *condition number* is given by

$$\kappa(A) = \kappa_2(A) = \frac{\sigma_1}{\sigma_n}. \tag{6.1}$$

The error bound (5.1) is therefore directly generalized to the least squares solution for overdetermined systems, with the condition number given by (6.1) in terms of the largest over the smallest singular values.

What can go wrong with the normal equations algorithm

Turning to the algorithm that requires solving a linear system with $B = A^TA$, the condition number of this symmetric positive definite matrix is the ratio of its largest to smallest eigenvalues (see page 132), which in turn is given by

$$\kappa_2(B) = \frac{\lambda_1}{\lambda_n} = \frac{\sigma_1^2}{\sigma_n^2} = \kappa_2(A)^2.$$

Thus, the relative error in the solution using the normal equations is bounded by $\kappa(A)^2$, which is square what it could be using a more stable algorithm. In practice, this is a problem when $\kappa(A)$ is "large but not incredibly large," as in Example 6.4.

Next we seek algorithms that allow for the error to be bounded in terms of $\kappa(A)$ and not its square.

The QR decomposition

We next develop solution methods via the QR *orthogonal decomposition*. An SVD approach also yields an algorithm that is discussed in Section 8.2.

Let us emphasize again the essential point that for any orthogonal matrix P and vector $\mathbf{w}$ of appropriate sizes we can write

$$\|\mathbf{b} - A\mathbf{w}\| = \|P\mathbf{b} - PA\mathbf{w}\|.$$

We can therefore design an **orthogonal transformation** such that PA would have a more yielding form than the given matrix A. For the least squares problem it could be beneficial to transform our given A into an upper triangular form in this way.

Fortunately, this is possible using the celebrated **QR decomposition**. For a matrix A of size $m \times n$ with full column rank, there exists an orthogonal matrix Q of size $m \times m$ and an $m \times n$ matrix $\binom{R}{0}$, where R is an upper triangular $n \times n$ matrix, such that

$$A = Q\begin{pmatrix} R \\ 0 \end{pmatrix}.$$

Exercise 6 shows that R is nonsingular if and only if A has full column rank.

The details on how to construct such a decomposition are given in Section 6.3; see in particular the description of *Householder reflections*. Assuming the decomposition is available let us now see how to utilize it for solving our problem.

It is possible to multiply a residual vector by Q^T, still retaining the same least squares problem. This yields

$$\|\mathbf{b}-A\mathbf{x}\| = \left\|\mathbf{b}-Q\begin{pmatrix}R\\0\end{pmatrix}\mathbf{x}\right\| = \left\|Q^T\mathbf{b}-\begin{pmatrix}R\\0\end{pmatrix}\mathbf{x}\right\|.$$

So we can now minimize the right-hand term. But at this point it is clear how to do it! Partitioning the vector $Q^T\mathbf{b}$ to its first n components $\mathbf{c}$ and its last $m-n$ components $\mathbf{d}$, so

$$Q^T\mathbf{b} = \begin{pmatrix}\mathbf{c}\\\mathbf{d}\end{pmatrix},$$

we see that

$$\|\mathbf{r}\|^2 = \|\mathbf{b}-A\mathbf{x}\|^2 = \|\mathbf{c}-R\mathbf{x}\|^2 + \|\mathbf{d}\|^2.$$

We have no control over $\|\mathbf{d}\|^2$. But the first term can be set to its minimal value of zero by solving the upper triangular system $R\mathbf{x} = \mathbf{c}$. This yields the solution $\mathbf{x}$, and we also get $\|\mathbf{r}\| = \|\mathbf{d}\|$.

Example 6.5. Let

$$A = \begin{pmatrix}1 & 0\\1 & 1\\1 & 2\end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix}0.1\\0.9\\2.0\end{pmatrix}.$$

The data are the same as in Example 6.2. Note that this example can be happily solved by a simple method; but we use it to demonstrate a simple instance of QR decomposition.

The MATLAB command `[Q,T] = qr(A)` produces an orthogonal $m \times m$ matrix Q and an upper triangular $m \times n$ matrix T such that $A = QT$. Applying it for the above data we get

$$Q = \begin{pmatrix}-.5774 & .7071 & .4082\\-.5774 & 0 & -.8165\\-.5774 & -.7071 & .4082\end{pmatrix}, \quad T = \begin{pmatrix}-1.7321 & -1.7321\\0 & -1.4142\\0 & 0\end{pmatrix}.$$

Corresponding to our notation we see that

$$R = \begin{pmatrix}-1.7321 & -1.7321\\0 & -1.4142\end{pmatrix}.$$

Multiplying A and $\mathbf{b}$ by Q^T we have the problem

$$\min_{\mathbf{x}} \left\| \begin{pmatrix}-1.7321\\-1.3435\\0.1225\end{pmatrix} - \begin{pmatrix}-1.7321 & -1.7321\\0 & -1.4142\\0 & 0\end{pmatrix}\mathbf{x} \right\|.$$

Hence $\mathbf{c} = (-1.7321, -1.3435)^T$, $\mathbf{d} = .1225$. Solving $R\mathbf{x} = \mathbf{c}$ yields the solution $\mathbf{x} = (.05, .95)^T$, while the norm of the residual is $\|\mathbf{r}\| = \|\mathbf{d}\| = .1225$. ∎

Note that the QR decomposition obtained in Example 6.5 is not unique: $-Q$ and $-R$ provide perfectly valid QR factors, too.

Note: Standard methods for solving the linear least squares problem include the following:

1. Normal equations: fast, simple, intuitive, but less robust in ill-conditioned situations.
2. QR decomposition: this is the "standard" approach implemented in general-purpose software. It is more computationally expensive than the normal equations approach if $m \gg n$ but is more robust.
3. SVD: used mostly when A is rank deficient or nearly rank deficient (in which case the QR approach may not be sufficiently robust). The SVD-based approach is very robust but is significantly more expensive in general and cannot be adapted to deal efficiently with sparse matrices. We will discuss this further in Section 8.2.

Economy size QR decomposition

An *economy size* QR decomposition also exists, with Q now being $m \times n$ with orthonormal columns and R remaining an $n \times n$ upper triangular matrix. Here we may write

$$A = QR.$$

Note that Q has the dimensions of A; in particular, it is rectangular and thus has no inverse if $m > n$.

Such a decomposition may be derived because if we look at the original QR decomposition, we see that the last $m-n$ columns of the $m \times m$ matrix Q actually multiply rows of zeros in the right-hand factor (those zero rows that appear right below the upper triangular matrix R).

Another way to get the same least squares algorithm is by looking directly at the normal equations and applying the decomposition. To see this, it is notationally convenient to use the economy size QR, obtaining

$$\begin{aligned}\mathbf{x} = A^\dagger \mathbf{b} &= (A^T A)^{-1} A^T \mathbf{b} = (R^T Q^T Q R)^{-1} R^T Q^T \mathbf{b} \\ &= (R^T R)^{-1} R^T Q^T \mathbf{b} = R^{-1} Q^T \mathbf{b}.\end{aligned}$$

See Exercise 5.

The least squares solution via the QR decomposition thus comprises the three steps specified in the algorithm given on the next page.

The dominant cost factor in this algorithm is the computation of the QR decomposition. Ways to compute it will be discussed soon, in Section 6.3.

Algorithm efficiency

If done efficiently, the cost of forming the QR decomposition is approximately $2mn^2 - 2n^3/3$ flops: it is the same as for the normal equations method if $m \approx n$ and is twice as expensive if $m \gg n$. The QR decomposition method takes some advantage of sparsity and bandedness, similarly to the LU decomposition if $n = m$, but twice as slow. It is more robust than the normal equations method: while for the latter the relative error is proportional to $[\kappa(A)]^2$ and breaks down already if $\kappa(A) \approx \frac{1}{\sqrt{\eta}}$, a robust version of QR yields a relative error proportional to $\kappa(A)$, so it breaks down only if $\kappa(A) \approx \frac{1}{\eta}$, with η the rounding unit fondly remembered from Chapter 2.

Algorithm: Least Squares via the QR Decomposition (full and economy versions).

1. Decompose
   (a) $A = QR$, with R upper triangular and Q $m \times n$ with orthonormal columns; or
   (b) in the full version, $A = Q \begin{pmatrix} R \\ 0 \end{pmatrix}$, with Q $m \times m$ orthogonal.
2. Compute
   (a) $\mathbf{c} = Q^T \mathbf{b}$; or
   (b) in the full version, $\begin{pmatrix} \mathbf{c} \\ \mathbf{d} \end{pmatrix} = Q^T \mathbf{b}$.
3. Solve the upper triangular system $R\mathbf{x} = \mathbf{c}$.

Example 6.6. A lot of special effort has gone into optimizing the backslash operation in MATLAB. It is therefore of interest to see how the different options for solving overdetermined systems play out in terms of computing time.

Thus, we fill a matrix A and a vector $\mathbf{b}$ with random numbers and measure CPU times for solving the overdetermined system using the QR and normal equations options. We assume that no singularity or severe ill-conditioning arise, which is reasonable: as important as singular matrices are, they are not encountered at random. Here is the code:

```
for n = 300:100:1000
  % fill a rectangular matrix A and a vector b with random numbers
  % hoping that A'*A is nonsingular
  m = n+1; % or m= 3*n+1, or something else
  A = randn(m,n); b = randn(m,1);

  % solve and find execution times; first, Matlab way using QR
  t0 = cputime;
  xqr = A \ b;
  temp = cputime;
  tqr(n/100-2) = temp - t0;

  % next use normal equations
  t0 = temp;
  B = A'*A; y = A'*b;
  xne = B \ y;
  temp = cputime;
  tne(n/100-2) = temp - t0;
end

ratio = tqr./tne;
plot(300:100:1000,ratio)
```

Note that the backslash operator appears in both methods. But its meaning is different. In particular, the system for B involves a square matrix.

From Figure 6.5 we see that as n gets large enough the normal equations method is about twice more efficient when $m \approx n$ and about 4 times more efficient when $m \approx 3n$. These results do depend on the computing environment and should not be taken as more than a local, rough indication. Regarding accuracy of the results, even for $n = 1000$ the maximum difference between the two obtained solutions was still very small in this experiment. But then again, random matrices do not give rise to particularly ill-conditioned matrices, and when the going gets tough in terms of conditioning, the differences may be much more significant. ∎

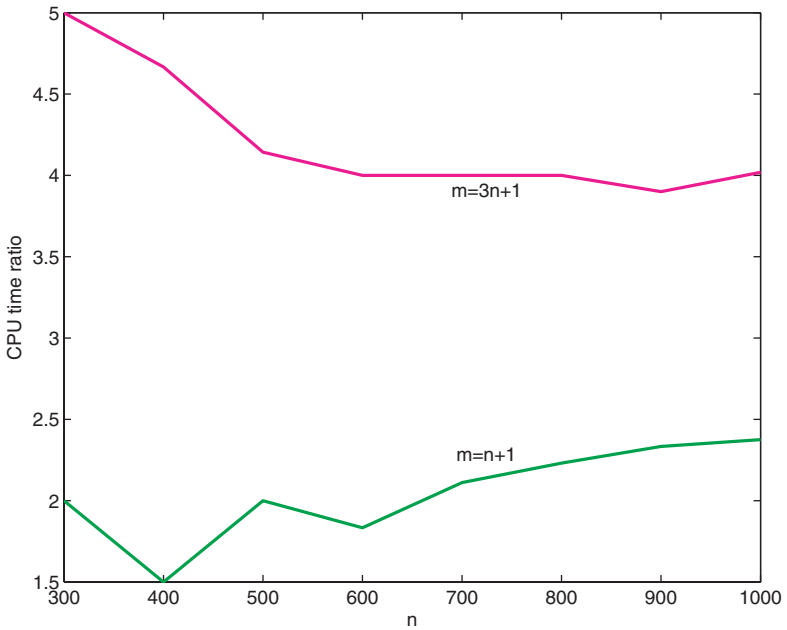

Figure 6.5. *Ratio of execution times using QR vs. normal equations. The number of rows for each n is $3n + 1$ for the upper curve and $n + 1$ for the lower one.*

Note: That the MATLAB designers decided to have QR as the default least squares solver is a tribute to the cause of robustness.

Specific exercises for this section: Exercises 5–6.

6.3 Householder transformations and Gram–Schmidt orthogonalization

In the previous section we have discussed the QR decomposition and explained its merits. But how does one actually compute it? That is the subject of the present section.

There are two intuitive ways of going about this: either by constructing R bit by bit until we obtain an orthogonal matrix Q, or by applying a sequence of orthogonal transformations that bring the given matrix A into an upper triangular matrix R. Throughout this section we continue to use the shorthand notation $\|\mathbf{v}\|$ for $\|\mathbf{v}\|_2$.

Note: A particularly robust QR decomposition method for the least squares problem is through **Householder reflections** (also known as *Householder transformations*). Below we first describe other, conceptually important methods that may be easier to grasp, but programs and a numerical example are only provided for this method of choice. This section is more technical than the previous ones in the present chapter.

Gram–Schmidt orthogonalization

The first of these two approaches is probably the more intuitive one, although not necessarily the better way to go computationally, and it yields the *Gram–Schmidt* process. Let us illustrate it for a simple example and then generalize.

Example 6.7. Consider a 3×2 instance. We write

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix} = \begin{pmatrix} q_{11} & q_{12} \\ q_{21} & q_{22} \\ q_{31} & q_{32} \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} \\ 0 & r_{22} \end{pmatrix}.$$

For notational convenience, denote inner products by

$$\langle \mathbf{z}, \mathbf{y} \rangle \equiv \mathbf{z}^T \mathbf{y}.$$

Writing the above column by column we have

$$(\mathbf{a}_1,\ \mathbf{a}_2) = (r_{11}\mathbf{q}_1,\ r_{12}\mathbf{q}_1 + r_{22}\mathbf{q}_2).$$

Requiring orthonormal columns yields the three conditions $\|\mathbf{q}_1\| = 1$, $\|\mathbf{q}_2\| = 1$, and $\langle \mathbf{q}_1, \mathbf{q}_2 \rangle = 0$. We thus proceed as follows: for the first column, use $\|\mathbf{q}_1\| = 1$ to obtain

$$r_{11} = \|\mathbf{a}_1\|\ ;\ \ \mathbf{q}_1 = \mathbf{a}_1 / r_{11}.$$

For the second column we have $\mathbf{a}_2 = r_{12}\mathbf{q}_1 + r_{22}\mathbf{q}_2$, and applying an inner product with $\mathbf{q}_1$ yields

$$r_{12} = \langle \mathbf{a}_2, \mathbf{q}_1 \rangle.$$

Next, observe that once r_{12} is known we can compute $\tilde{\mathbf{q}}_2 = \mathbf{a}_2 - r_{12}\mathbf{q}_1$ and then set $r_{22} = \|\tilde{\mathbf{q}}_2\|$ and $\mathbf{q}_2 = \tilde{\mathbf{q}}_2 / r_{22}$. This completes the procedure; we now have the matrices Q and R in the economy size version. ∎

Example 6.7 should convince you that it is possible to obtain a QR decomposition by an orthogonalization process that constructs the required factors column by column. By directly extending the above procedure we obtain the Gram–Schmidt orthogonalization procedure, which computes $r_{ij} = \langle \mathbf{a}_j, \mathbf{q}_i \rangle$ such that

$$\mathbf{a}_j = \sum_{i=1}^{j} r_{ij}\mathbf{q}_i.$$

For any given j, once $\mathbf{q}_1$ through $\mathbf{q}_{j-1}$ are known, $\mathbf{q}_j$ (and hence r_{jj}) can be computed in a straightforward manner.

It is assumed that the columns of A are linearly independent, or in other words that A has full column rank. If there is linear dependence, say, $\mathbf{a}_j$ is linearly dependent on $\mathbf{a}_1$ through $\mathbf{a}_{j-1}$, then at the jth stage of the algorithm we will get (ignoring roundoff errors) $r_{jj} = 0$. Thus, robust codes have a condition for gracefully exiting if r_{jj} is zero or approximately zero.

Modified Gram–Schmidt

The procedure we just described is called *classical Gram–Schmidt*. For all its endearing simplicity, it is numerically unstable if the columns of A are nearly linearly dependent.

Stability can be improved by a simple fix: instead of using $\mathbf{a}_1$ through $\mathbf{a}_{j-1}$ for constructing $\mathbf{q}_j$, we employ the already computed $\mathbf{q}_1$ through $\mathbf{q}_{j-1}$, which by construction are orthogonal to one another and thus less prone to damaging effects of roundoff errors. The resulting algorithm, given on this page, is identical to the classical one except for the fourth line and is called the *modified Gram–Schmidt algorithm*. It is the preferred version in all numerical codes.

Algorithm: Modified Gram–Schmidt Orthogonalization.
Input: matrix A of size $m \times n$.

```
for j = 1 : n
  q_j = a_j
  for i = 1 : j − 1
    r_ij = ⟨q_j, q_i⟩
    q_j = q_j − r_ij q_i
  end
  r_jj = ‖q_j‖
  q_j = q_j / r_jj
end
```

Orthogonal transformations

The Gram–Schmidt procedure just described is a process of orthogonalization: we use the elements of a triangular matrix as coefficients while turning the given A into an orthogonal matrix Q. A natural alternative is to use orthogonal *transformations* to turn A into an upper triangular matrix R. In many ways, the concept is reminiscent of Gaussian elimination, at least in terms of the "zeroing out" aspect of it.

A slow but occasionally useful way of accomplishing this goal is by a surgical, pointed algorithm that takes aim at one entry of A at a time. Such are **Givens rotations**. They are transformations that successively rotate vectors in a manner that turns 2×2 submatrices into upper triangular 2×2 ones. Givens rotations are worth considering in cases where the original matrix has a special structure. A famous instance here are the Hessenberg matrices (page 139); see Exercise 5.20. Such matrices arise in abundance in advanced techniques for solving linear systems and eigenvalue problems, and Givens rotations thus form an integral part of many of the relevant algorithms. They are easy to parallelize and are useful in a variety of other situations as well.

Householder reflections

The technique employing *Householder transformations* is the most suitable for general-purpose QR decomposition and the most reminiscent of LU decomposition.

Turning a column into a unit vector, orthogonally

Suppose we are given a vector $\mathbf{z}$ and are looking for an orthogonal transformation that zeros out all its entries except the first one. How can we find such a transformation?

Consider the matrix

$$P = I - 2\mathbf{u}\mathbf{u}^T,$$

where $\mathbf{u}$ is a given unit vector, $\|\mathbf{u}\| = 1$, and $I = I_m$ is the $m \times m$ identity matrix. Clearly, $P\mathbf{u} = -\mathbf{u}$, so we can say that P is a *reflector*. We could proceed as follows: for the given vector $\mathbf{z}$, find a vector $\mathbf{u}$ such that P, defined as above, has the property that $P\mathbf{z}$ transforms $\mathbf{z}$ into the vector $(\alpha, 0, \ldots, 0)^T = \alpha\mathbf{e}_1$. Denoting $\beta = 2\mathbf{u}^T\mathbf{z}$, we want

$$P\mathbf{z} = \mathbf{z} - 2\mathbf{u}\mathbf{u}^T\mathbf{z} = \mathbf{z} - \beta\mathbf{u} = \alpha\mathbf{e}_1.$$

From this it follows that $\mathbf{u}$ is a vector in the direction $\mathbf{z} - \alpha\mathbf{e}_1$. Now, since P is an orthogonal transformation (Exercise 9) we have that $\|\mathbf{z}\| = \|P\mathbf{z}\| = |\alpha|$; hence it follows that $\mathbf{u}$ is a unit vector in the direction of $\mathbf{z} \pm \|\mathbf{z}\|\mathbf{e}_1$. We finally select the sign in the latter expression in accordance with that of z_1 (the first element of $\mathbf{z}$), because this reduces the possibility of cancellation error.

Extending to QR decomposition

The procedure outlined above can be applied and extended in an analogous way to the derivation of LU decomposition in Section 5.2. To compute the QR decomposition we proceed as follows. Start by applying a reflection, call it $P^{(1)}$, that turns the first column of A into a multiple of $\mathbf{e}_1$ as described above, using the first column of A for $\mathbf{z}$, so the matrix $P^{(1)}A$ now has all zeros in its first column except the $(1,1)$ entry. Next, apply a similar transformation based on the second column of $P^{(1)}A$, zeroing it below the diagonal $(2,2)$ entry, as when using $M^{(2)}$ in the formation of the LU decomposition. After completing this stage, the matrix $P^{(2)}P^{(1)}A$ has all zero entries below the $(1,1)$ and $(2,2)$ elements.

In a general step of the algorithm, $P^{(k)}$ is a concatenation of the $(k-1) \times (k-1)$ identity matrix with an $(m-k+1) \times (m-k+1)$ reflector which is responsible for zeroing out all entries below the (k,k) entry of $P^{(k-1)}P^{(k-2)} \cdots P^{(1)}A$. See Figure 6.6. Carrying this through until the end (n steps in total when $n < m$), we obtain the desired QR decomposition. The fact that the product of orthogonal matrices is also an orthogonal matrix comes in handy.

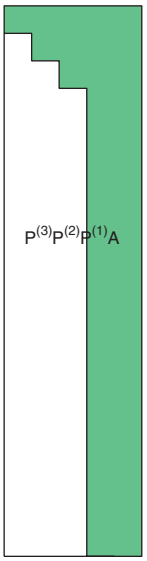

Figure 6.6. *Householder reflection. Depicted is a* 20×5 *matrix* A *after* 3 *reflections.*

The Householder reflection is not only elegant, it is also very robust. It has better numerical stability properties than modified Gram–Schmidt, and its storage requirements are modest: there is no need to store the $P^{(k)}$ matrices, since each of them is defined by a single vector, say, $\mathbf{u}_k$. Similar to the LU decomposition, when Householder reflections are used the vectors $\mathbf{u}_k$ can overwrite A column by column. So, this QR decomposition, although not in the economy-size version, is actually rather economical! The following script demonstrates how it can all be done:

```
function [A,p] = house(A)
%
% function [A,p] = house(A)
%
% perform QR decomposition using Householder reflections
% Transformations are of the form P_k = I - 2u_k(u_k^T), so
% store effecting vector u_k in p(k) + A(k+1:m,k). Assume m > n.

[m,n]=size(A); p = zeros(1,n);
for k = 1:n
  % define u of length = m-k+1
  z = A(k:m,k);
  e1 = [1; zeros(m-k,1)];
  u = z+sign(z(1))*norm(z)*e1; u = u/norm(u);
  % update nonzero part of A by I-2uu^T
  A(k:m,k:n) = A(k:m,k:n)-2*u*(u'*A(k:m,k:n));
  % store u
  p(k) = u(1);
  A(k+1:m,k) = u(2:m-k+1);
end
```

Solving least squares problems with `house`

Now, suppose we wish to compute the solution $\mathbf{x}$ of the least squares problem $\min_{\mathbf{x}} \|\mathbf{b} - A\mathbf{x}\|$. For that, we will need to compute $Q^T\mathbf{b}$. If the QR decomposition of A is already given, having executed the above script, all we need to do is the following:

```
function [x,nr] = lsol(b,AA,p)
%
% function [x,r] = lsol(b,AA,p)
%
% Given the output AA and p of house, containing Q and R of matrix A,
% and given a right-hand-side vector b,  solve min || b - Ax ||.
% Return also the norm of the residual, nr = ||r|| = ||b - Ax||.

y = b(:); [m,n] = size(AA);
% transform b
for k=1:n
  u = [p(k);AA(k+1:m,k)];
  y(k:m) = y(k:m) - 2*u *(u'*y(k:m));
end
% form upper triangular R and solve
R = triu(AA(1:n,:));
x = R \ y(1:n); nr = norm(y(n+1:m));
```

Notice that these programs are remarkably short, especially given the length of our description leading to them. This reflects our acquired stage of knowing what we're doing!

Example 6.8. We show for a small matrix how Householder transformations can be used to construct the QR decomposition and solve a least squares problem. For this, consider the data of Example 6.1.

The first column of A is $\mathbf{a}_1 = (1,2,5,3-1)^T$, and the sign of its first element is positive. Thus, compute $\mathbf{a}_1 + \|\mathbf{a}_1\|(1,0,0,0,0)^T = (7.3246,2,5,3,-1)^T$ and normalize it, obtaining the reflection vector $\mathbf{u} = (0.7610, 0.2078, 0.5195, 0.3117, -0.1039)^T$. This reflection vector defines the transformation

$$P^{(1)} = I_5 - 2\mathbf{u}\mathbf{u}^T = \begin{pmatrix} -0.1581 & -0.3162 & -0.7906 & -0.4743 & 0.1581 \\ -0.3162 & 0.9137 & -0.2159 & -0.1295 & 0.0432 \\ -0.7906 & -0.2159 & 0.4603 & -0.3238 & 0.1079 \\ -0.4743 & -0.1295 & -0.3238 & 0.8057 & 0.0648 \\ 0.1581 & 0.0432 & 0.1079 & 0.0648 & 0.9784 \end{pmatrix}$$

and

$$P^{(1)}A = \begin{pmatrix} -6.3246 & -4.7434 & -1.5811 \\ 0 & 1.7048 & 4.2952 \\ 0 & -0.2380 & -3.7620 \\ 0 & 3.0572 & 2.9428 \\ 0 & 6.6476 & 3.3524 \end{pmatrix}; \qquad P^{(1)}\mathbf{b} = \begin{pmatrix} -2.8460 \\ -3.8693 \\ 0.3266 \\ -4.8040 \\ 1.9347 \end{pmatrix}.$$

Note, again, that $P^{(1)}$ need not be constructed explicitly; all we need to store in this step is the vector $\mathbf{u}$. Next, we work on zeroing out the elements of $P^{(1)}A$ below the (2,2) entry. Thus, we define $\mathbf{z} = (1.7048, -0.2380, 3.0572, 6.6476)^T$, and our reflector is $\mathbf{u} = \mathbf{z} + \text{sign}(\mathbf{z}(1))\|\mathbf{z}\|\mathbf{e}_1 = (0.7832, -0.0202, 0.2597, 0.5646)^T$. Notice that now $\mathbf{z}$ and $\mathbf{e}_1$ are only four elements in length.

The matrix $P^{(2)}$ is a concatenation of the 1×1 identity matrix with $I_4 - 2\mathbf{u}\mathbf{u}^T$, and now the second column has been taken care of. Finally, we work on the third column. Please verify that, setting $\mathbf{z} = (-3.5155, -0.2234, -3.5322)^T$ and $\mathbf{u} = (-0.9232, -0.0243, -0.3835)^T$. After completing this step we have $A = Q\binom{R}{0}$ where

$$Q = (P^{(3)}P^{(2)}P^{(1)})^T = \begin{pmatrix} -0.1581 & 0.0998 & 0.2555 & -0.3087 & 0.8969 \\ -0.3162 & -0.1996 & 0.6919 & -0.4755 & -0.3942 \\ -0.7906 & 0.0998 & -0.5464 & -0.2454 & -0.0793 \\ -0.4743 & -0.3659 & 0.2661 & 0.7427 & 0.1369 \\ 0.1581 & -0.8980 & -0.2945 & -0.2585 & 0.1227 \end{pmatrix}$$

and

$$R = \begin{pmatrix} -6.3246 & -4.7434 & -1.5811 \\ 0 & -7.5166 & -5.2550 \\ 0 & 0 & 4.9885 \end{pmatrix}.$$

For the right-hand-side vector given in Example 6.1, $Q^T\mathbf{b} = (-2.8460, 1.1308, -3.9205, -3.2545, 3.8287)^T$. The solution of the least squares problem is the product of R^{-1} with the first $n = 3$ elements of $Q^T\mathbf{b}$, yielding again $\mathbf{x} = (0.3472, 0.3990, -0.7859)^T$. ∎

Specific exercises for this section: Exercises 7–9.

6.4 Exercises

0. **Review questions**

   (a) What are the normal equations? What orthogonality property of least squares solutions do they reflect?

   (b) Let A be an $m \times n$ matrix, with $m > n$. Under what conditions on the matrix A is the matrix A^TA:

      i. symmetric?

      ii. nonsingular?

      iii. positive definite?

   (c) Suppose a given rectangular matrix A is sparse but has one dense row. What is the sparsity pattern of A^TA? Explain the meaning of this for solving least squares problems using normal equations.

   (d) Define the pseudo-inverse of a rectangular matrix A that has more rows than columns, and explain how it connects to a matrix inverse and what is "pseudo" about it.

   (e) What is the floating point operation count for solving the least squares problem using the normal equations?

   (f) Can the regression curve of Example 6.2 be obtained using the program `lsfit`? If yes, then how?

   (g) Why do data fitting problems typically yield overdetermined systems and not systems of the form treated in Chapter 5?

   (h) In what circumstances is the QR decomposition expected to yield better least squares fitting results than the normal equations?

   (i) What is the difference between QR decomposition and the economy size QR decomposition?

   (j) What is the difference between the QR decomposition produced by modified Gram–Schmidt and that produced by Householder transformations?

   (k) What is the floating point operation count for solving the least squares problem using the QR decomposition?

1. The following data were obtained in connection with the values of a certain fictitious material property:

t	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
b	0.9	1.01	1.05	0.97	0.98	0.95	0.01	−0.1	0.02	−0.1	0.0

It was then hypothesized that the underlying material property is a piecewise constant function with one break point (i.e., two constant pieces).

(a) Plot the data points and decide where (approximately) the break point should be.

(b) Find the piecewise constant function which best fits the data by least squares, and plot it, too.

2. (a) Evaluate the function

$$f(t) = .05\sin(1000t) + .5\cos(\pi t) - .4\sin(10t)$$

at the 101 points given by 0:.01:1. Plot the resulting broken line interpolant.

(b) In order to study the slow scale trend of this function, we wish to find a low degree polynomial (degree at most 6) that best approximates f in the least squares norm at the above 101 data points. By studying the figure from part (a) find out the smallest n that would offer a good fit in this sense. (Try to do this without further computing.)

(c) Find the best approximating polynomial v of degree n and plot it together with f. What are your observations?

3. Let us synthesize data in the following way. We start with the cubic polynomial

$$q(t) = -11 + \frac{55}{3}t - \frac{17}{2}t^2 + \frac{7}{6}t^3.$$

Thus, $n = 4$. This is sampled at 33 equidistant points between 0.9 and 4.1. Then we add to these values 30% noise using the random number generator `randn` in MATLAB, to obtain "the data" which the approximations that you will construct "see." From here on, no knowledge of $q(t)$, its properties, or its values anywhere is assumed: we pretend we don't know where the data came from!

Your programming task is to pass through these data three approximations:

(a) An interpolating polynomial of degree 32. This can be done using the MATLAB function `polyfit`. You don't need to know how such interpolation works for this exercise, although details are given in Chapter 10.

(b) An interpolating cubic spline using the MATLAB function `spline`. The corresponding method is described in Section 11.3, but again you don't need to rush and study that right now.

(c) A cubic polynomial which best fits the data in the ℓ_2 sense, obtained by our function `lsfit`.

Plot the data and the obtained approximations as in Figures 6.3 and 6.4. Which of these approximations make sense? Discuss.

4. Often in practice, an approximation of the form

$$u(t) = \gamma_1 e^{\gamma_2 t}$$

is sought for a data fitting problem, where γ_1 and γ_2 are constants.

Assume given data $(t_1, z_1), (t_2, z_2), \ldots, (t_m, z_m)$, where $z_i > 0$, $i = 1, 2, \ldots, m$, and $m > 0$.

(a) Explain in one brief sentence why the techniques introduced in the present chapter cannot be directly applied to find this $u(t)$.

(b) Considering instead

$$v(t) = \ln u(t) = (\ln \gamma_1) + \gamma_2 t,$$

it makes sense to define $b_i = \ln z_i$, $i = 1,2,\ldots,m$, and then find coefficients x_1 and x_2 such that $v(t) = x_1 + x_2 t$ is the best least squares fit for the data

$$(t_1, b_1), (t_2, b_2), \ldots, (t_m, b_m).$$

Using this method, find $u(t)$ for the data

i	1	2	3
t_i	0.0	1.0	2.0
z_i	$e^{0.1}$	$e^{0.9}$	e^2

5. (a) Why can't one directly extend the LU decomposition to a long and skinny matrix in order to solve least squares problems?

(b) When writing $\mathbf{x} = A^{\dagger}\mathbf{b} = \cdots = R^{-1}Q^T\mathbf{b}$ we have somehow moved from the conditioning $[\kappa(A)]^2$ to the conditioning $\kappa(A)$ through mathematical equalities. Where is the improvement step hidden? Explain.

6. (a) Let Q be an orthogonal $m \times m$ matrix and R an $n \times n$ upper triangular matrix, $m > n$, such that

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix}.$$

Show that the diagonal elements of R all satisfy $r_{ii} \neq 0$, $i = 1,\ldots,n$, if and only if A has full column rank.

(b) Next, let Q be $m \times n$ with orthonormal columns (so $Q^T Q = I$, but Q does not have an inverse) such that

$$A = QR.$$

Prove the same claim as in part (a) for this economy size decomposition.

7. Find the QR factorization of the general 2×2 matrix

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}.$$

8. (a) Explain what may happen during the course of the Gram–Schmidt process if the matrix A is rank deficient.

(b) Show that classical Gram–Schmidt and the modified version are mathematically equivalent.

(c) Construct a 3×2 example, using a decimal floating point system with a 2-digit fraction, in which modified Gram–Schmidt proves to be more numerically stable than the classical version.

9. (a) For a given real vector $\mathbf{u}$ satisfying $\|\mathbf{u}\|_2 = 1$, show that the matrix $P = I - 2\mathbf{u}\mathbf{u}^T$ is orthogonal.

(b) Suppose A is a complex-valued matrix. Construct a complex analogue of Householder transformations, with the reflector given by $P = I - 2\mathbf{u}\mathbf{u}^*$, where $*$ denotes a complex conjugate transpose and $\mathbf{u}^*\mathbf{u} = 1$. (The matrix P is now *unitary*, meaning that $P^*P = I$.)

6.5 Additional notes

The material of this chapter elegantly extends the approaches of Chapter 5 to handle over-determined systems, and unlike Chapter 8 it does not require any serious change in our state of mind in the sense that exact direct algorithms are considered. Yet its range of application is ubiquitous. No wonder it is popular and central among fans of numerical linear algebra and users alike. A variety of issues not discussed or just skimmed over here are exposed and dealt with in books on numerical linear algebra; see, e.g., Demmel [21], Watkins [74], or the reference book of Golub and van Loan [30].

As mentioned in Section 6.1 the problems of data fitting in ℓ_1 and in ℓ_∞ can be posed as special instances of *linear programming*. There are many undergraduate textbooks on the latter topic. We mention instead the higher level but concise expositions in the true classics Luenberger [51] and Fletcher [25] and the more modern Nocedal and Wright [57]. In Section 9.3 we quickly address linear programming as well.

The same books are also excellent references for treatments of *nonlinear least squares* problems. Such problems are treated in Section 9.2. A very simple example is provided in Exercise 4. Nonlinear data fitting problems arise frequently when solving inverse problems. The books by Tikhonov and Arsenin [67] and Engl, Hanke, and Neubauer [24] are relevant, though admittedly not always very accessible.