would you say captures the general trend of the data better? Obviously, this is a subjective question, and its answer depends on both the nature of the given data (e.g., the uncertainty of the data values) and the purpose of the fit. Explain your assumptions in answering.

3.2. A common problem in surveying is to determine the altitudes of a series of points with respect to some reference point. The measurements are subject to error, so more observations are taken than are strictly necessary to determine the altitudes, and the resulting overdetermined system is solved in the least squares sense to smooth out errors. Suppose that there are four points whose altitudes x_1, x_2, x_3, x_4 are to be determined. In addition to direct measurements of each x_i with respect to the reference point, measurements are also taken of each point with respect to all of the others. The resulting measurements are:

$$\begin{aligned} x_1 &= 2.95, & x_2 &= 1.74, \\ x_3 &= -1.45, & x_4 &= 1.32, \\ x_1 - x_2 &= 1.23, & x_1 - x_3 &= 4.45, \\ x_1 - x_4 &= 1.61, & x_2 - x_3 &= 3.21, \\ x_2 - x_4 &= 0.45, & x_3 - x_4 &= -2.75. \end{aligned}$$

Set up the corresponding least squares system $Ax \cong b$ and use a library routine, or one of your own design, to solve it for the best values of the altitudes. How do your computed values compare with the direct measurements?

3.3. (a) For a series of matrices A of order n, record the execution times for a library routine to compute the LU factorization of A. Using a linear least squares routine, or one of your own design, fit a cubic polynomial to the execution times as a function of n. To obtain reliable results, use a fairly wide range of values for n, say, in increments of 100 from 100 up to several hundred, depending on the speed and available memory of the computer you use. You may obtain more accurate timings by averaging several runs for a given matrix size. The resulting cubic polynomial could be used to predict the execution time for other values of n not tried, such as very large values for n. What is the predicted execution time for a matrix of order 10,000?

(b) Try to determine the basic execution rate (in floating-point operations per second, or *flops*) for your computer by timing a known computation, such as matrix multiplication. You can then use this information to determine the complexity of LU factorization, based on the polynomial fit to the execution times. After converting to floating-point operations, how does the dominant term compare with the theoretically expected value of $\frac{4}{3}n^3$ (counting both additions and multiplications)? Try to explain any discrepancy. If you use a system that provides operation counts automatically, such as `MATLAB` or some supercomputers, try this same experiment fitting the operation counts directly.

3.4. (a) Solve the following least squares problem using any method you like:

$$\begin{bmatrix} 0.16 & 0.10 \\ 0.17 & 0.11 \\ 2.02 & 1.29 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \cong \begin{bmatrix} 0.26 \\ 0.28 \\ 3.31 \end{bmatrix}.$$

(b) Now solve the same least squares problem again, but this time use the slightly perturbed right-hand side

$$b = \begin{bmatrix} 0.27 \\ 0.25 \\ 3.33 \end{bmatrix}.$$

(c) Compare your results from parts a and b. Can you explain this difference?

3.5. A planet follows an elliptical orbit, which can be represented in a Cartesian (x, y) coordinate system by the equation

$$ay^2 + bxy + cx + dy + e = x^2.$$

(a) Use a library routine, or one of your own design, for linear least squares to determine the orbital parameters a, b, c, d, e, given the following observations of the planet's position:

x	1.02	0.95	0.87	0.77	0.67
y	0.39	0.32	0.27	0.22	0.18
x	0.56	0.44	0.30	0.16	0.01
y	0.15	0.13	0.12	0.13	0.15

In addition to printing the values for the orbital parameters, plot the resulting orbit and the given data points in the (x, y) plane.

(b) This least squares problem is nearly rank-deficient. To see what effect this has on the solution, perturb the input data slightly by adding

to each coordinate of each data point a random number uniformly distributed on the interval $[-0.005, 0.005]$ (see Section 13.5) and solve the least squares problem with the perturbed data. Compare the new values for the parameters with those previously computed. What effect does this difference have on the plot of the orbit? Can you explain this behavior?

(c) Solve the same least squares problem again, for both the original and the perturbed data, this time using a library routine (or one of your own design) specifically designed to deal with rank deficiency (by using column pivoting, for example). Such a routine usually includes as an input parameter a tolerance to be used in determining the numerical rank of the matrix. Experiment with various values for the tolerance, say, 10^{-k}, $k = 1, \ldots, 5$. What is the resulting rank of the matrix for each value of the tolerance? Compare the behavior of the two solutions (for the original and the perturbed data) with each other as the tolerance and the resulting rank change. How well do the resulting orbits fit the data points as the tolerance and rank vary? Which solution would you regard as better: one that fits the data more closely, or one that is less sensitive to small perturbations in the data? Why?

(d) Use a library routine to compute the singular value decomposition of the 10×5 least squares matrix.

(e) Use the singular value decomposition to compute the solution to the least squares problem. With the singular values in order of decreasing magnitude, compute the solutions using the first k singular values, $k = 1, \ldots, 5$. For each of the five solutions obtained, print the values for the orbital parameters and also plot the resulting orbits along with the given data points in the (x, y) plane.

(f) Perturb the input data slightly by adding to each coordinate of each data point a random number uniformly distributed on the interval $[-0.005, 0.005]$ (see Section 13.5). Compute the singular value decomposition of the new least squares matrix, and solve the least squares problem with the perturbed data as in part e. Compare the new values for the parameters with those previously computed for each value of k. What effect does this difference have on the plot of the orbits? Can you explain this behavior? Which solution would you regard as better: one that fits the data more closely, or one that is less sensitive to small perturbations in the data? Why?

(g) For simplicity, we have used ordinary least squares in this problem, but in fact all of the data are equally subject to observational errors (indeed, x appears on both sides of the equation), which makes the applicability of ordinary least squares questionable. Reformulate this problem as a total least squares problem and solve the latter using the singular value decomposition as described in Section 3.6.1.

3.6. Write a routine for computing the pseudoinverse of an arbitrary $m \times n$ matrix. You may call a library routine to compute the singular value decomposition, then use its output to compute the pseudoinverse (see Section 3.6.1). Consider the use of a tolerance for declaring relatively small singular values to be zero. Test your routine on both singular and nonsingular matrices. In the latter case, of course, your results should agree with those of standard matrix inversion. What happens when the matrix is nonsingular, but severely ill-conditioned (e.g., a Hilbert matrix)?

3.7. Write a routine for solving an arbitrary, possibly rank-deficient, linear least squares problem $\boldsymbol{Ax} \cong \boldsymbol{b}$ using the singular value decomposition. You may call a library routine to compute the SVD, then use its output to compute the least squares solution (see Section 3.6.1). The input to your routine should include the matrix $\boldsymbol{A}$, right-hand-side vector $\boldsymbol{b}$, and a tolerance for determining the numerical rank of $\boldsymbol{A}$. Test your routine on some of the linear least squares problems in the other computer problems for this chapter.

3.8. To demonstrate how results from the normal equations method and QR factorization can differ numerically, we need a least squares problem that is ill-conditioned and also has a small residual. We can generate such a problem as follows. We will fit a polynomial of degree $n-1$,

$$p_{n-1}(t) = x_1 + x_2 t + x_3 t^2 + \cdots + x_n t^{n-1},$$

to m data points (t_i, y_i), $m > n$. We choose $t_i = (i-1)/(m-1)$, $i = 1, \ldots, m$, so that the data points are equally spaced on the interval $[0, 1]$. We will generate the corresponding values y_i by first choosing values for the x_j, say, $x_j = 1$,