of the section. Nevertheless, the flop count gives us a useful first indication of an algorithm's operation time, and we shall count flops as a matter of course.

Exercise 1.1.8 Begin to familiarize yourself with MATLAB. Log on to a machine that has MATLAB installed, and start MATLAB. From MATLAB's command line type `A = randn(3,4)` to generate a 3×4 matrix with random entries. To learn more about the `randn` command, type `help randn`. Now type `x = randn(4,1)` to get a vector (a 4×1 matrix) of random numbers. To multiply A by x and store the result in a new vector b, type `b = A*x`.

To get MATLAB to save a transcript of your session, type `diary on`. This will cause a file named *diary*, containing a record of your MATLAB session, to be saved. Later on you can edit this file, print it out, turn it in to your instructor, or whatever. To learn more about the `diary` command, type `help diary`.

Other useful commands are `help` and `help help`. In addition to getting help from the command line, you can open MATLAB's help browser and search or browse for a large number of topics. Explore MATLAB to find out what other features it has. There are demonstrations and help for beginners that you might find useful. MATLAB also has a built in editor and a debugger. □

Exercise 1.1.9 Consider the following simple MATLAB program.

```
n = 500;
for jay = 1:4
  if jay > 1
    oldtime = time;
  end
  A = randn(n);
  x = randn(n,1);
  t = cputime;
  for rep = 1:100 % compute the product 100 times
    b = A*x;
  end
  matrixsize = n
  time = cputime - t
  if jay > 1
    ratio = time/oldtime
  end
  n = 2*n;
end
```

The syntax is simple enough that you can readily figure out what the program does. The commands `randn` and `b = A*x` are familiar from the previous exercise. We perform each matrix-vector multiplication 100 times in order to build up a significant amount of computing time. The function `cputime` tells how much computer (central processing unit) time the current MATLAB session has used. This program times the execution of 100 matrix-vector multiplications for square matrices A of dimension 500, 1000, 2000, and 4000.

Enter this program into a file called matvectime.m. Actually, you can call it whatever you please, but you must use the .m extension. (MATLAB programs are called m-files.) Now start MATLAB and type `matvectime` (without the .m) to execute the program. Depending on how fast your computer is, you may like to change the size of the matrix or the number of times the `jay` loop or the `rep` loop is executed.

MATLAB normally prints the output of all of its operations to the screen. You can suppress printing by terminating the operation with a semicolon. Looking at the program, we see that `A`, `x`, `t`, and `b` will not be printed, but `matrixsize`, `time`, and `ratio` will.

Look at the values of `ratio`. Are they close to what you would expect based on the flop count? □

Exercise 1.1.10 Write a MATLAB program that performs matrix-vector multiplication two different ways: (a) using the built-in MATLAB command `b = A*x`, and (b) using loops, as follows.

```
b = zeros(n,1);
for j = 1:n
  for i = 1:n
    b(i) = b(i) + A(i,j)*x(j);
  end
end
```

Time the two different methods on matrices of various sizes. Which method is faster? (You may want to use some of the code from Exercise 1.1.9.) □

Exercise 1.1.11 Write a Fortran or C program that performs matrix-vector multiplication using loops. How does its speed compare with that of MATLAB? □

Multiplying a Matrix by a Matrix

If A is an $n \times m$ matrix, and X is $m \times p$, we can form the product $B = AX$, which is $n \times p$. The (i, j) entry of B is

$$b_{ij} = \sum_{k=1}^{m} a_{ik} x_{kj}. \tag{1.1.12}$$

In words, the (i, j) entry of B is the dot product of the ith row of A with the jth column of X. If $p = 1$, this operation reduces to matrix-vector multiplication. If $p > 1$, the matrix-matrix multiply amounts to p matrix-vector multiplies: the jth column of B is just A times the jth column of X.

A computer program to multiply A by X might look something like this:

$$\begin{array}{l} B \leftarrow 0 \\ \text{for } i = 1, \ldots, n \\ \quad \left[\begin{array}{l} \text{for } j = 1, \ldots, p \\ \quad \left[\begin{array}{l} \text{for } k = 1, \ldots, m \\ \quad \left[\; b_{ij} \leftarrow b_{ij} + a_{ik} x_{kj} \right. \end{array} \right. \end{array} \right. \end{array} \tag{1.1.13}$$

The decision to put the i-loop outside the j-loop was perfectly arbitrary. In fact, the order in which the updates $b_{ij} \leftarrow b_{ij} + a_{ik}x_{kj}$ are made is irrelevant, so the three loops can be nested in any way. Thus there are six basic variants of the matrix-matrix multiplication algorithm. These are all equivalent in principle. In practice, some versions may run faster than others on a given computer because of the order in which the data are accessed.

It is a simple matter to count the flops in matrix-matrix multiplication. Since there are two flops in the innermost loop of (1.1.13), the total flop count is

$$\sum_{i=1}^{n} \sum_{j=1}^{p} \sum_{k=1}^{m} 2 = 2nmp.$$

In the important case when all of the matrices are square of dimension $n \times n$, the flop count is $2n^3$. Thus square matrix multiplication is an $O(n^3)$ operation. This function grows rather quickly with n: each time n is doubled, the flop count is multiplied by eight. (However, this is not the whole story. See the remarks on fast matrix multiplication at the end of this section.)

Exercise 1.1.14 Modify the MATLAB code from Exercise 1.1.9 by changing the matrix-vector multiply `b = A*x` to a matrix-matrix multiply `B = A*X`, where X is $n \times n$. You may also want to decrease the initial matrix dimension n and the number of times the `rep` loop is executed. Run the code and check the ratios. Are they close to what you would expect them to be, based on the flop count? □

Block Matrices and Block Matrix Operations

The idea of partitioning matrices into blocks is simple but powerful. It is a useful tool for proving theorems, constructing algorithms, and developing faster variants of algorithms. We will use block matrices again and again throughout the book.

Consider the matrix product $AX = B$, where the matrices have dimensions $n \times m$, $m \times p$, and $n \times p$, respectively. Suppose we partition A into blocks:

$$A = \begin{array}{c} \\ n_1 \\ n_2 \end{array} \begin{array}{c} \begin{array}{cc} m_1 & m_2 \end{array} \\ \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \end{array} \qquad \begin{cases} n_1 + n_2 = n \\ m_1 + m_2 = m. \end{cases} \tag{1.1.15}$$

The labels n_1, n_2, m_1, and m_2 indicate that the block A_{ij} has dimensions $n_i \times m_j$.

We can partition X similarly.

$$X = \begin{array}{c} \\ m_1 \\ m_2 \end{array} \begin{array}{c} \begin{array}{cc} p_1 & p_2 \end{array} \\ \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \end{array} \qquad \begin{cases} m_1 + m_2 = m \\ p_1 + p_2 = p. \end{cases} \tag{1.1.16}$$

The numbers m_1 and m_2 are the same as in (1.1.15). Thus, for example, the number of rows in X_{12} is the same as the number of columns in A_{11} and A_{21}. Continuing in the same spirit, we partition B as follows:

$$B = \begin{array}{c} \\ n_1 \\ n_2 \end{array} \begin{array}{c} \begin{array}{cc} p_1 & p_2 \end{array} \\ \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \end{array} \qquad \begin{cases} n_1 + n_2 = n \\ p_1 + p_2 = p. \end{cases} \tag{1.1.17}$$

The row partition of B is the same as that of A, and the column partition is the same as that of X. The product $AX = B$ can now be written as

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}. \tag{1.1.18}$$

We know that B is related to A and X by the equations (1.1.12), but how are the blocks of B related to the blocks of A and X? We would hope to be able to multiply the blocks as if they were numbers. For example, we hope that $A_{11}X_{11}+A_{12}X_{21} = B_{11}$. Theorem 1.1.19 states that this is indeed the case.

Theorem 1.1.19 *Let A, X, and B be partitioned as in (1.1.15), (1.1.16), and (1.1.17), respectively. Then $AX = B$ if and only if*

$$A_{i1}X_{1j} + A_{i2}X_{2j} = B_{ij}, \qquad i, j = 1, 2.$$

You can easily convince yourself that Theorem 1.1.19 is true. It follows more or less immediately from the definition of matrix multiplication. We will skip the tedious but routine exercise of writing out a detailed proof. You might find the following exercise useful.

Exercise 1.1.20 Consider matrices A, X, and B, partitioned as indicated.

$$A = \left[\begin{array}{c|cc} 1 & 3 & 2 \\ 2 & 1 & 1 \\ \hline -1 & 0 & 1 \end{array}\right] \qquad X = \left[\begin{array}{c|cc} 1 & 0 & 1 \\ \hline 2 & 1 & 1 \\ -1 & 2 & 0 \end{array}\right] \qquad B = \left[\begin{array}{c|cc} 5 & 7 & 4 \\ 3 & 3 & 3 \\ \hline -2 & 2 & -1 \end{array}\right]$$

Thus, for example, $A_{12} = \begin{bmatrix} 3 & 2 \\ 1 & 1 \end{bmatrix}$ and $A_{21} = \begin{bmatrix} -1 \end{bmatrix}$. Show that $AX = B$ and $A_{i1}X_{1j} + A_{i2}X_{2j} = B_{ij}$ for $i, j = 1, 2$. □

Exercise 1.2.4 Prove that if A^{-1} exists, then there can be no nonzero y for which $Ay = 0$. □

Exercise 1.2.5 Prove that if A^{-1} exists, then $\det(A) \neq 0$. □

We now move on to some examples.

Electrical Circuits

Example 1.2.6 Consider the electrical circuit shown in Figure 1.1. Suppose the circuit is in an equilibrium state; all of the voltages and currents are constant. The four unknown nodal voltages $x_1, \ldots, x_4$ can be determined as follows. At each of the four nodes, the sum of the currents away from the node must be zero (Kirchhoff's current law). This gives us an equation for each node. In each of these equations the currents can be expressed in terms of voltages using Ohm's law, which states that the voltage drop (in volts) is equal to the current (in amperes) times the resistance (in ohms). For example, suppose the current from node 3 to node 4 through the $5\,\Omega$ resistor is I. Then by Ohm's law, $x_3 - x_4 = 5I$, so $I = .2(x_3 - x_4)$. Treating the other two currents flowing from node 3 in the same way, and applying Kirchhoff's current law, we get the equation

$$.2(x_3 - x_4) + 1(x_3 - x_1) + .5(x_3 - 6) = 0$$

or

$$-x_1 + 1.7x_3 - .2x_4 = 3.$$

Applying the same procedure to nodes 1, 2, and 4, we obtain a system of four linear equations in four unknowns:

$$\begin{array}{rcrcrcrcl} 2x_1 & - & x_2 & - & x_3 & & & = & 0 \\ -\ x_1 & + & 1.5x_2 & & & - & .5x_4 & = & 0 \\ -\ x_1 & & & + & 1.7x_3 & - & .2x_4 & = & 3 \\ & - & .5x_2 & - & .2x_3 & + & 1.7x_4 & = & 0 \end{array},$$

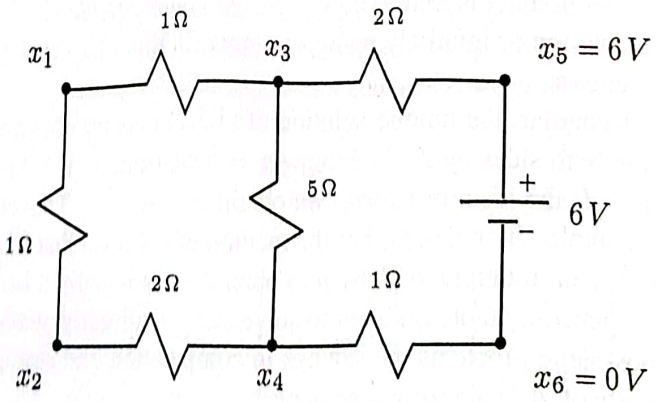

Figure 1.1 Solve for the nodal voltages.

which can be written as a single matrix equation

$$\begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 1.5 & 0 & -.5 \\ -1 & 0 & 1.7 & -.2 \\ 0 & -.5 & -.2 & 1.7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 3 \\ 0 \end{bmatrix}.$$

Though we will not prove it here, the coefficient matrix is nonsingular, so the system has exactly one solution. Solving it using MATLAB, we find that

$$x = \begin{bmatrix} 3.0638 \\ 2.4255 \\ 3.7021 \\ 1.1489 \end{bmatrix}.$$

Thus, for example, the voltage at node 3 is 3.7021 volts. These are not the exact answers; they are rounded off to four decimal places. Given the nodal voltages, we can easily calculate the current through any of the resistors by Ohm's law. For example, the current flowing from node 3 to node 4 is $.2(x_3 - x_4) = 0.5106$ amperes. □

Exercise 1.2.7 Verify the correctness of the equations in Example 1.2.6. Use MATLAB (or some other means) to compute the solution. If you are unfamiliar with MATLAB, you can find out how to enter matrices by searching for the topic "entering matrices" in MATLAB's help browser or by exploring MATLAB's "demos" or "getting started". Once you have entered the matrix `A` and the vector `b`, you can type `x = A\b` to solve for x. A transcript of your whole session (which you can later edit, print out, and turn in to your instructor) can be made by using the command `diary on`. For more information about the `diary` command type `help diary` at the command line or search for "diary" in the help browser. □

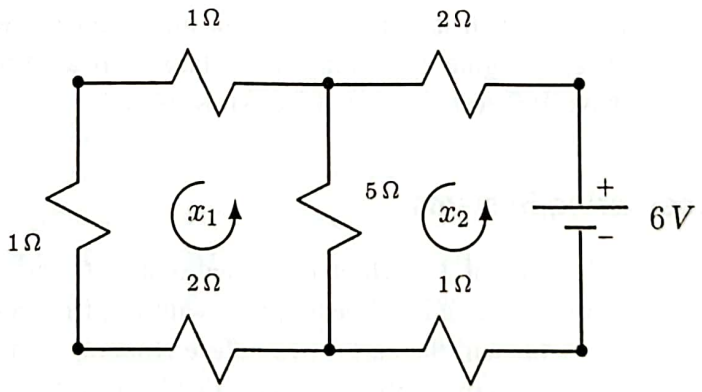

Figure 1.2 Solve for the loop currents.

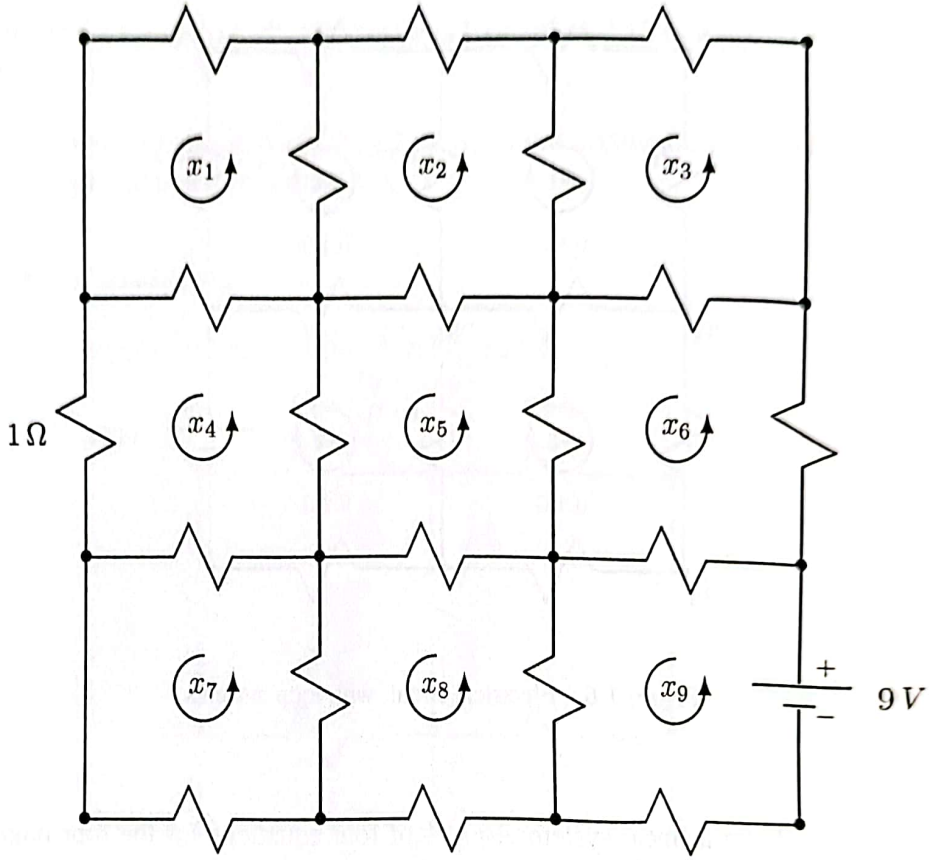

Figure 1.7 Calculate the loop currents.

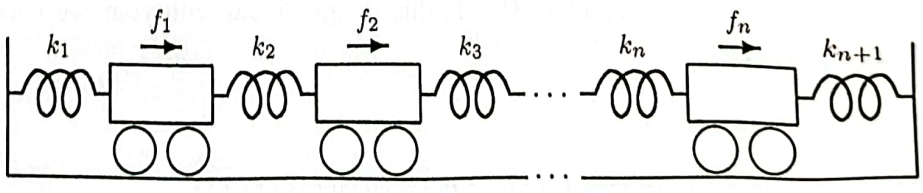

Figure 1.8 System of n masses

Exercise 1.2.20 Consider a system of n carts connected by springs, as shown in Figure 1.8. The ith spring has a stiffness of k_i newtons/meter. Suppose that the carts are subjected to steady forces of $f_1, f_2, \ldots, f_n$ newtons, respectively, causing displacements of $x_1, x_2, \ldots, x_n$ meters, respectively.

(a) Write down a system of n linear equations $Ax = b$ that could be solved for $x_1, \ldots, x_n$. Notice that if n is at all large, the vast majority of the entries of A will be zeros. Matrices with this property are called *sparse*. Since all of the nonzeros are confined to a narrow band around the main diagonal, A is also called *banded*. In particular, the nonzeros are confined to three diagonals, so A is *tridiagonal*.

(b) Compute the solution in the case $n = 20$, $k_i = 1$ newton/meter for all i, and $f_i = 0$ except for $f_5 = 1$ newton and $f_{16} = -1$ newton. (Type `help toeplitz` to learn an easy way to enter the coefficient matrix in MATLAB.)

□

Exercise 1.2.21 Recall the definition of the derivative from elementary calculus:

$$u'(x) = \lim_{h \to 0} \frac{u(x+h) - u(x)}{h}.$$

(a) Show that if h is a sufficiently small positive number, then both

$$\frac{u(x+h) - u(x)}{h} \quad \text{and} \quad \frac{u(x) - u(x-h)}{h}$$

are good approximations of $u'(x)$.

(b) Take the average of the two estimates from part (a) to obtain the estimate

$$u'(x) \approx \frac{u(x+h) - u(x-h)}{2h}.$$

Draw a picture (the graph of u and a few straight lines) that shows that this estimate is likely to be a better estimate of $u'(x)$ than either of the estimates from part (a) are.

(c) Apply the estimate from part (b) to $u''(x)$ with h replaced by $h/2$ to obtain

$$u''(x) \approx \frac{u'(x+h/2) - u'(x-h/2)}{h}.$$

Now approximate $u'(x+h/2)$ and $u'(x-h/2)$ using the estimate from part (b), again with h replaced by $h/2$, to obtain

$$u''(x) \approx \frac{u(x+h) - 2u(x) + u(x-h)}{h^2}.$$

□