CHAPTER 8

Implementation Issues

In the previous chapter, we rewrote the simplex method using matrix notation. This is the first step toward our aim of describing the simplex method as one would implement it as a computer program. In this chapter, we shall continue in this direction by addressing some important implementation issues.

The most time-consuming steps in the simplex method are the computations

$$\Delta x_{\mathcal{B}} = B^{-1}Ne_j \qquad \text{and} \qquad \Delta z_{\mathcal{N}} = -(B^{-1}N)^T e_i,$$

and the difficulty in these steps arises from the B^{-1}. Of course, we don't ever actually compute the inverse of the basis matrix. Instead, we calculate, say, $\Delta x_{\mathcal{B}}$ by solving the following system of equations:

$$B\Delta x_{\mathcal{B}} = a_j, \tag{8.1}$$

where

$$a_j = Ne_j$$

is the column of N associated with nonbasic variable x_j.

Similarly, the calculation of $\Delta z_{\mathcal{N}}$ is also broken into two steps:

$$\begin{aligned} B^T v &= e_i, \\ \Delta z_{\mathcal{N}} &= -N^T v. \end{aligned} \tag{8.2}$$

Here, the first step is the solution of a large system of equations, this time involving B^T instead of B, and the second step is the comparatively trivial task of multiplying a vector on the left by the matrix $-N^T$.

Solving the systems of equations (8.1) and (8.2) is where most of the complexity of a simplex iteration lies. We discuss solving such systems in the first two sections. In the second section, we look at the effect of sparsity on these systems. The next few sections explain how to reuse and/or update the computations of one iteration in subsequent iterations. In the final sections, we address a few other issues that affect the efficiency of an implementation.

1. Solving Systems of Equations: LU-Factorization

In this section, we discuss solving systems of equations of the form

$$Bx = b,$$

where B is an invertible $m \times m$ matrix and b is an arbitrary m-vector. (Analysis of the transpose $B^T x = b$ is left to Exercise 8.4.) Our first thought is to use Gaussian

R.J. Vanderbei, *Linear Programming*, International Series in Operations Research & Management Science 196, DOI 10.1007/978-1-4614-7630-6_8,

elimination. This idea is correct, but to explain how Gaussian elimination is actually implemented, we need to take a fresh look at how it works. To explain, let us consider an example:

$$B = \begin{bmatrix} 2 & & 4 & & -2 \\ 3 & 1 & & 1 & \\ -1 & & -1 & & -2 \\ & -1 & & & -6 \\ & & 1 & & 4 \end{bmatrix}.$$

(Note that, to emphasize the importance of sparsity, zero entries are simply left blank.) In Gaussian elimination, one begins by subtracting appropriate multiples of the first row from each subsequent row to get zeros in the first column below the diagonal. For our specific example, we subtract $3/2$ times the first row from the second row and we subtract $-1/2$ times the first row from the third row. The result is

$$\begin{bmatrix} 2 & & 4 & & -2 \\ & 1 & -6 & 1 & 3 \\ & & 1 & & -3 \\ & -1 & & & -6 \\ & & 1 & & 4 \end{bmatrix}.$$

Shortly, we will want to remember the values of the nonzero elements in the first column. Therefore, let us agree to do the row operations that are required to eliminate nonzeros, but when we write down the result of the elimination, we will leave the nonzeros there. With this convention, the result of the elimination of the first column can be written as

$$\left[\begin{array}{ccccc} 2 & & 4 & & -2 \\ 3 & 1 & -6 & 1 & 3 \\ -1 & & 1 & & -3 \\ & -1 & & & -6 \\ & & 1 & & 4 \end{array}\right].$$

Note that we have drawn a line to separate the eliminated top/left parts of the matrix from the uneliminated lower-right part.

Next, we eliminate the nonzeros below the second diagonal (there's only one) by subtracting an appropriate multiple of the second row from each subsequent row. Again, we write the answer without zeroing out the eliminated elements:

$$\left[\begin{array}{ccccc} 2 & & 4 & & -2 \\ 3 & 1 & -6 & 1 & 3 \\ -1 & & 1 & & -3 \\ & -1 & -6 & 1 & -3 \\ & & 1 & & 4 \end{array}\right].$$

After eliminating the third column, we get

$$\left[\begin{array}{rrrrr} 2 & & 4 & & -2 \\ 3 & 1 & -6 & 1 & 3 \\ -1 & & 1 & & -3 \\ & -1 & -6 & 1 & -21 \\ & & 1 & & 7 \end{array}\right].$$

Now, the remaining uneliminated part is already an upper triangular matrix, and hence no more elimination is required.

At this point, you are probably wondering how this strangely produced matrix is related to the original matrix B. The answer is both simple and elegant. First, take the final matrix and split it into three matrices: the matrix consisting of all elements on or below the diagonal, the matrix consisting of just the diagonal elements, and the matrix consisting of all elements on or above the diagonal. It is amazing but true that B is simply the product of the resulting lower triangular matrix times the inverse of the diagonal matrix times the upper triangular matrix:

$$B = \begin{bmatrix} 2 & & & & \\ 3 & 1 & & & \\ -1 & & 1 & & \\ & -1 & -6 & 1 & \\ & & 1 & & 7 \end{bmatrix} \begin{bmatrix} 2 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 7 \end{bmatrix}^{-1} \begin{bmatrix} 2 & & 4 & & -2 \\ & 1 & -6 & 1 & 3 \\ & & 1 & & -3 \\ & & & 1 & -21 \\ & & & & 7 \end{bmatrix}.$$

(If you don't believe it, multiply them and see.) Normally, the product of the lower triangular matrix and the diagonal matrix is denoted by L,

$$L = \begin{bmatrix} 2 & & & & \\ 3 & 1 & & & \\ -1 & & 1 & & \\ & -1 & -6 & 1 & \\ & & 1 & & 7 \end{bmatrix} \begin{bmatrix} 2 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 7 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & & & & \\ \frac{3}{2} & 1 & & & \\ -\frac{1}{2} & & 1 & & \\ & -1 & -6 & 1 & \\ & & 1 & & 1 \end{bmatrix},$$

and the upper triangular matrix is denote by U:

$$U = \begin{bmatrix} 2 & & 4 & & -2 \\ & 1 & -6 & 1 & 3 \\ & & 1 & & -3 \\ & & & 1 & -21 \\ & & & & 7 \end{bmatrix}.$$

The resulting representation,

$$B = LU,$$

is called an *LU-factorization* of B. Finding an LU-factorization is equivalent to Gaussian elimination in the sense that multiplying B on the left by L^{-1} has the effect of applying row operations to B to put it into upper-triangular form U.

The value of an LU-factorization is that it can be used to solve systems of equations. For example, suppose that we wish to solve equation (8.1), where B is as above and

$$a_j = \begin{bmatrix} 7 \\ -2 \\ 0 \\ 3 \\ 0 \end{bmatrix}. \tag{8.3}$$

First, we substitute LU for B so that the system becomes

$$LU\Delta x_{\mathcal{B}} = a_j.$$

Now, if we let $y = U\Delta x_{\mathcal{B}}$, then we can solve

$$Ly = b$$

for y, and once y is known, we can solve

$$U\Delta x_{\mathcal{B}} = y$$

for $\Delta x_{\mathcal{B}}$. Because L is lower triangular, solving $Ly = b$ is easy. Indeed, writing the system out,

$$\begin{bmatrix} 1 & & & & \\ \frac{3}{2} & 1 & & & \\ -\frac{1}{2} & & 1 & & \\ & -1 & -6 & 1 & \\ & & 1 & & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 7 \\ -2 \\ 0 \\ 3 \\ 0 \end{bmatrix},$$

we notice immediately that $y_1 = 7$. Then, given y_1, it becomes clear from the second equation that $y_2 = -2 - (3/2)y_1 = -25/2$. Continuing in this way, we find that

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 7 \\ -\frac{25}{2} \\ \frac{7}{2} \\ \frac{23}{2} \\ -\frac{7}{2} \end{bmatrix}.$$

The process of successively solving for the elements of the vector y starting with the first and proceeding to the last is called *forward substitution*.

Of course, solving $U\Delta x_{\mathcal{B}} = y$ is easy too, since U is upper triangular. The system to solve is given by

$$\begin{bmatrix} 2 & 4 & & -2 \\ & 1 & -6 & 1 & 3 \\ & & 1 & & -3 \\ & & & 1 & -21 \\ & & & & 7 \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \\ \Delta x_4 \\ \Delta x_5 \end{bmatrix} = \begin{bmatrix} 7 \\ -\frac{25}{2} \\ \frac{7}{2} \\ \frac{23}{2} \\ -\frac{7}{2} \end{bmatrix}$$

(note that, to keep notations simple, we are assuming that the basic indices are 1 through 5 so that $\Delta x_{\mathcal{B}} = (\Delta x_1, \Delta x_2, \Delta x_3, \Delta x_4, \Delta x_5)$). This time we start with the last equation and see that $\Delta x_5 = -1/2$. Then the second to last equation tells us that $\Delta x_4 = 23/2 + 21(\Delta x_5) = 1$. After working our way to the first equation, we have

$$\Delta x_{\mathcal{B}} = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \\ \Delta x_4 \\ \Delta x_5 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 2 \\ 1 \\ -\frac{1}{2} \end{bmatrix}.$$

This process of working from the last element of $\Delta x_{\mathcal{B}}$ back to the first is called *backward substitution*.

2. Exploiting Sparsity

In the previous section, we took a specific matrix B and constructed an LU factorization of it. However, with that example we were lucky in that every diagonal element was nonzero at the moment it was used to eliminate the nonzeros below it. Had we encountered a zero diagonal element, we would have been forced to rearrange the columns and/or the rows of the matrix to put a nonzero element in this position. For a random matrix (whatever that means), the odds of encountering a zero are nil, but a basis matrix can be expected to have plenty of zeros in it, since, for example, it is likely to contain columns associated with slack variables, which are all zero except for one 1. A matrix that contains zeros is called a *sparse matrix*.

When a sparse matrix has lots of zeros, two things happen. First, the chances of being required to make row and/or column permutations is high. Second, additional computational efficiency can be obtained by making further row and/or column permutations with the aim of keeping L and/or U as sparse as possible.

The problem of finding the "best" permutation is, in itself, harder than the linear programming problem that we ultimately wish to solve. But there are simple heuristics that help to preserve sparsity in L and U. We shall focus on just one such heuristic, called the *minimum-degree* ordering heuristic, which is describe as follows:

> *Before eliminating the nonzeros below a diagonal "pivot" element, scan all uneliminated rows and select the sparsest row, i.e., that row having the fewest nonzeros in its uneliminated part (ties can be broken arbitrarily). Swap this row with the pivot row. Then scan the uneliminated nonzeros in this row and select that one whose column has the fewest nonzeros in its uneliminated part. Swap this column with the pivot column so that this nonzero becomes the pivot element. (Of course, provisions should be made to reject such a pivot element if its value is close to zero.)*

As a matter of terminology, the number of nonzeros in the uneliminated part of a row/column is called the *degree* of the row/column. Hence, the name of the heuristic.

Let's apply the minimum-degree heuristic to the LU-factorization of the matrix B studied in the previous section. To keep track of the row and column permutations, we will indicate original row indices on the left and original column indices across the top. Hence, we start with:

$$B = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \left[\begin{array}{ccccc} 2 & & 4 & & -2 \\ 3 & 1 & & 1 & \\ -1 & & -1 & & -2 \\ & -1 & & & -6 \\ & & 1 & & 4 \end{array}\right] \end{array}.$$

To begin, row 4 has the fewest nonzeros, and within row 4, the -1 in column 2 belongs to the column with the fewest nonzeros. Hence, we swap rows 1 and 4 and we swap columns 1 and 2 to rewrite B as

$$B = \begin{array}{c} \\ 4 \\ 2 \\ 3 \\ 1 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 1 & 3 & 4 & 5 \end{array} \\ \left[\begin{array}{ccccc} -1 & & & & -6 \\ 1 & 3 & & 1 & \\ & -1 & -1 & & -2 \\ & 2 & 4 & & -2 \\ & & 1 & & 4 \end{array}\right] \end{array}.$$

Now, we eliminate the nonzeros under the first diagonal element (and, as before, we leave the eliminated nonzeros as they were). The result is

$$\begin{array}{c} \\ 4 \\ 2 \\ 3 \\ 1 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 1 & 3 & 4 & 5 \end{array} \\ \left[\begin{array}{ccccc} -1 & & & & -6 \\ 1 & 3 & & 1 & -6 \\ & -1 & -1 & & -2 \\ & 2 & 4 & & -2 \\ & & 1 & & 4 \end{array}\right] \end{array}.$$

Before doing the elimination associated with the second diagonal element, we note that row 5 is the row with minimum degree, and within row 5, the element 1 in column 3 has minimum column degree. Hence, we swap rows 2 and 5 and we swap columns 1 and 3 to get

$$\begin{array}{c} \\ 4 \\ 5 \\ 3 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 3 & 1 & 4 & 5 \end{array} \\ \left[\begin{array}{ccccc} -1 & & & & -6 \\ & 1 & & & 4 \\ & -1 & -1 & & -2 \\ & 4 & 2 & & -2 \\ 1 & & 3 & 1 & -6 \end{array}\right] \end{array}.$$

Now we eliminate the nonzeros under the second diagonal element to get

$$\begin{array}{c} \\ 4 \\ 5 \\ 3 \\ 1 \\ 2 \end{array}\begin{array}{c} \begin{array}{ccccc} 2 & 3 & 1 & 4 & 5 \end{array} \\ \left[\begin{array}{ccc|cc} -1 & & & & -6 \\ & 1 & & & 4 \\ & -1 & -1 & & 2 \\ & 4 & 2 & & -18 \\ 1 & & 3 & 1 & -6 \end{array}\right] \end{array}.$$

For the third stage of elimination, note that row 3 is a minimum-degree row and that, among the nonzero elements of that row, the -1 is in a minimum-degree column. Hence, for this stage no permutations are needed. The result of the elimination is

$$\begin{array}{c} \\ 4 \\ 5 \\ 3 \\ 1 \\ 2 \end{array}\begin{array}{c} \begin{array}{ccccc} 2 & 3 & 1 & 4 & 5 \end{array} \\ \left[\begin{array}{ccc|cc} -1 & & & & -6 \\ & 1 & & & 4 \\ & -1 & -1 & & 2 \\ & 4 & 2 & & -14 \\ 1 & & 3 & 1 & \end{array}\right] \end{array}.$$

For the next stage of the elimination, both of the remaining two rows have the same degree, and hence we don't need to swap rows. But we do need to swap columns 5 and 4 to put the -14 into the diagonal position. The result of the swap is

$$\begin{array}{c} \\ 4 \\ 5 \\ 3 \\ 1 \\ 2 \end{array}\begin{array}{c} \begin{array}{ccccc} 2 & 3 & 1 & 5 & 4 \end{array} \\ \left[\begin{array}{ccc|cc} -1 & & & -6 & \\ & 1 & & 4 & \\ & -1 & -1 & 2 & \\ & 4 & 2 & -14 & \\ 1 & & 3 & & 1 \end{array}\right] \end{array}.$$

At this point, we notice that the remaining 2×2 uneliminated part of the matrix is already upper triangular (in fact, diagonal), and hence no more elimination is needed.

With the elimination completed, we can extract the matrices L and U in the usual way:

$$L = \begin{array}{c} \\ 4 \\ 5 \\ 3 \\ 1 \\ 2 \end{array}\begin{array}{c} \begin{array}{ccccc} & & & & \end{array} \\ \left[\begin{array}{ccccc} -1 & & & & \\ & 1 & & & \\ & -1 & -1 & & \\ & 4 & 2 & -14 & \\ 1 & & 3 & & 1 \end{array}\right] \end{array} \begin{array}{c} \\ \\ \\ \\ \\ \end{array}\begin{array}{c} \begin{array}{ccccc} & & & & \end{array} \\ \left[\begin{array}{ccccc} -1 & & & & \\ & 1 & & & \\ & & -1 & & \\ & & & -\frac{1}{14} & \\ & & & & 1 \end{array}\right] \end{array}$$

$$= \begin{array}{c} \\ 4 \\ 5 \\ 3 \\ 1 \\ 2 \end{array}\begin{array}{c} \begin{array}{ccccc} & & & & \end{array} \\ \left[\begin{array}{ccccc} 1 & & & & \\ & 1 & & & \\ & -1 & 1 & & \\ & 4 & -2 & 1 & \\ -1 & & -3 & & 1 \end{array}\right] \end{array},$$

and

$$U = \begin{array}{c} \begin{array}{ccccc} 2 & 3 & 1 & 5 & 4 \end{array} \\ \left[\begin{array}{ccccc} -1 & & & -6 & \\ & 1 & & 4 & \\ & & -1 & 2 & \\ & & & -14 & \\ & & & & 1 \end{array}\right] \end{array}.$$

(Note that the columns of L and the rows of U do not have any "original" indices associated with them, and so no permutation is indicated across the top of L or down the left side of U.)

This LU-factorization has five off-diagonal nonzeros in L and three off-diagonal nonzeros in U for a total of eight off-diagonal nonzeros. In contrast, the LU factorization from the previous section had a total of 12 off-diagonal nonzeros. Hence, the minimum-degree ordering heuristic paid off for this example by reducing the number of nonzeros by 33 %. While such a reduction may not seem like a big deal for small matrices such as our 5×5 example, for large matrices the difference can be dramatic.

The fact that we have permuted the rows and columns to get this factorization has only a small impact on how one uses the factorization to solve systems of equations. To illustrate, let us solve the same system that we considered before: $B\Delta x_{\mathcal{B}} = a_j$, where a_j is given by (8.3). The first step in solving this system is to permute the rows of a_j so that they agree with the rows of L and then to use forward substitution to solve the system $Ly = a_j$. Writing it out, the system looks like this:

$$\begin{array}{c} 4 \\ 5 \\ 3 \\ 1 \\ 2 \end{array} \left[\begin{array}{ccccc} 1 & & & & \\ & 1 & & & \\ & -1 & 1 & & \\ & 4 & -2 & 1 & \\ -1 & & -3 & & 1 \end{array}\right] \left[\begin{array}{c} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{array}\right] = \begin{array}{c} 4 \\ 5 \\ 3 \\ 1 \\ 2 \end{array} \left[\begin{array}{c} 3 \\ 0 \\ 0 \\ 7 \\ -2 \end{array}\right]$$

The result of the forward substitution is that

$$\left[\begin{array}{c} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{array}\right] = \left[\begin{array}{c} 3 \\ 0 \\ 0 \\ 7 \\ 1 \end{array}\right]. \tag{8.4}$$

The next step is to solve the system $U\Delta x_{\mathcal{B}} = y$. Writing this system out, we get

$$\begin{array}{c} \begin{array}{ccccc} 2 & 3 & 1 & 5 & 4 \end{array} \\ \left[\begin{array}{ccccc} -1 & & & -6 & \\ & 1 & & 4 & \\ & & -1 & 2 & \\ & & & -14 & \\ & & & & 1 \end{array}\right] \end{array} \begin{array}{c} 2 \\ 3 \\ 1 \\ 5 \\ 4 \end{array} \left[\begin{array}{c} \Delta x_2 \\ \Delta x_3 \\ \Delta x_1 \\ \Delta x_5 \\ \Delta x_4 \end{array}\right] = \left[\begin{array}{c} 3 \\ 0 \\ 0 \\ 7 \\ 1 \end{array}\right].$$

Using backward substitution, we see that

$$\begin{array}{c} 2 \\ 3 \\ 1 \\ 5 \\ 4 \end{array} \begin{bmatrix} \Delta x_2 \\ \Delta x_3 \\ \Delta x_1 \\ \Delta x_5 \\ \Delta x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ -1 \\ -\frac{1}{2} \\ 1 \end{bmatrix}.$$

Finally, we rewrite the solution listing the elements of $\Delta x_{\mathcal{B}}$ in their original order:

$$\Delta x_{\mathcal{B}} = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \\ \Delta x_4 \\ \Delta x_5 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 2 \\ 1 \\ -\frac{1}{2} \end{bmatrix}.$$

Of course, the answer obtained here agrees with the one obtained at the end of the previous section.

Even with good fill-in minimizing heuristics such as minimum-degree, the LU-factorization remains a significant computational bottleneck. To see why, consider for the moment dense matrices. If we were to write a subroutine to carry out an LU-factorization, we would find that the main body of the routine would have a big triply nested loop:

```
for each column index j {
    for each remaining row index i {
        for each remaining column index k {
            update the (i,k) entry in accordance with
            the aim to make the (i,j) entry be zero
        }
    }
}
```

Since each of these loops involves approximately m steps, the LU-factorization routine requires about m^3 operations and hence is called an order m^3 algorithm. Similar considerations tell us that the forward and backward substitutions are both order m^2 algorithms. This means that forward and backward substitution can be done much faster than LU-factorization. Indeed, if $m = 5{,}000$, then factorization takes a couple of 1,000 times longer than a forward or backward substitution. Of course, this argument is for dense matrices. But for sparse matrices a similar, if less dramatic, effect is seen. Typically, for sparse matrices, one expects that factorization will take from 10 to 100 times longer than substitution. Therefore, it is important to perform as few LU-factorizations as possible. This is the subject of the next section.

3. Reusing a Factorization

In the previous two sections, we showed how to use an LU-factorization of B to solve the system of equations

$$B\Delta x_{\mathcal{B}} = a_j$$

for the primal step direction $\Delta x_{\mathcal{B}}$. Since the basis matrix doesn't change much from one iteration of the simplex method to the next (columns get replaced by new ones one at a time), we ask whether the LU-factorization of B from the current iteration might somehow be used again to solve the systems of equations that arise in the next iteration (or even the next several iterations).

Let B denote the current basis (for which a factorization has already been computed) and let $\tilde{B}$ denote the basis of the next iteration. Then $\tilde{B}$ is simply B with the column that holds the column vector a_i associated with the leaving variable x_i replaced by a new column vector a_j associated with the entering variable x_j. This verbal description can be converted into a formula:

$$\tilde{B} = B + (a_j - a_i)e_i^T. \tag{8.5}$$

Here, as before, e_i denotes the vector that is all zeros except for a one in the position associated with index i—to be definite, let us say that this position is the pth position in the vector. To see why this formula is correct, it is helpful to realize that a column vector, say a, times e_i^T produces a matrix that is all zero except for the pth column, which contains the column vector a.

Since the basis B is invertible, (8.5) can be rewritten as

$$\tilde{B} = B\left(I + B^{-1}(a_j - a_i)e_i^T\right).$$

Denote the matrix in parentheses by E. Recall that $a_j = Ne_j$, since it is the column vector from A associated with the entering variable x_j. Hence,

$$B^{-1}a_j = B^{-1}Ne_j = \Delta x_{\mathcal{B}},$$

which is a vector we need to compute in the current iteration anyway. Also,

$$B^{-1}a_i = e_i,$$

since a_i is the column of B associated with the leaving variable x_i. Therefore, we can write E more simply as

$$E = I + (\Delta x_{\mathcal{B}} - e_i)e_i^T.$$

Now, if E has a simple inverse, then we can use it together with the LU-factorization of B to provide an efficient means of solving systems of equations involving $\tilde{B}$. The following proposition shows that E does indeed have a simple inverse.

PROPOSITION 8.1. *Given two column vectors u and v for which $1 + v^T u \neq 0$,*

$$(I + uv^T)^{-1} = I - \frac{uv^T}{1 + v^T u}.$$

PROOF. The proof is trivial. We simply multiply the matrix by its supposed inverse and check that we get the identity:

$$\begin{aligned}
\left(I + uv^T\right)\left(I - \frac{uv^T}{1 + v^T u}\right) &= I + uv^T - \frac{uv^T}{1 + v^T u} - \frac{uv^T uv^T}{1 + v^T u} \\
&= I + uv^T\left(1 - \frac{1}{1 + v^T u} - \frac{v^T u}{1 + v^T u}\right) \\
&= I,
\end{aligned}$$

where the last equality follows from the observation that the parenthesized expression vanishes. □

The identity in Proposition 8.1 may seem mysterious, but in fact it has a simple derivation based on the explicit formula for the sum of a geometric series:

$$\sum_{j=0}^{\infty} \xi^j = \frac{1}{1-\xi}, \qquad \text{for } |\xi| < 1.$$

This is an identity for real numbers, but it also holds for matrices:

$$\sum_{j=0}^{\infty} X^j = (I - X)^{-1},$$

provided that the absolute value of each of the eigenvalues of X is less than one (we don't prove this here, since it's just for motivation). Assuming, for the moment, that the absolute values of the eigenvalues of uv^T are less than one (actually, all but one of them are zero), we can expand $(I + uv^T)^{-1}$ in a geometric series, reassociate products, and collapse the resulting geometric series to get

$$\begin{aligned}
\left(I + uv^T\right)^{-1} &= I - uv^T + (uv^T)(uv^T) - (uv^T)(uv^T)(uv^T) + \cdots \\
&= I - uv^T + u(v^Tu)v^T - u(v^Tu)(v^Tu)v^T + \cdots \\
&= I - u\left(1 - v^Tu + (v^Tu)^2 - \cdots\right)v^T \\
&= I - u\frac{1}{1+v^Tu}v^T \\
&= I - \frac{uv^T}{1+v^Tu},
\end{aligned}$$

where the last equality follows from the fact that $1/(1 + v^Tu)$ is a scalar and therefore can be pulled out of the vector/matrix calculation.

Applying Proposition 8.1 to matrix E, we see that

$$\begin{aligned}
E^{-1} &= I - \frac{(\Delta x_{\mathcal{B}} - e_i)e_i^T}{1 + e_i^T(\Delta x_{\mathcal{B}} - e_i)} \\
&= I - \frac{(\Delta x_{\mathcal{B}} - e_i)e_i^T}{\Delta x_i} \\
&= \left[\begin{array}{ccc|c|ccc}
1 & & & -\frac{\Delta x_{j_1}}{\Delta x_i} & & & \\
& \ddots & & \vdots & & & \\
& & 1 & -\frac{\Delta x_{j_{p-1}}}{\Delta x_i} & & & \\
\hline
& & & \frac{1}{\Delta x_i} & & & \\
\hline
& & & -\frac{\Delta x_{j_{p+1}}}{\Delta x_i} & 1 & & \\
& & & \vdots & & \ddots & \\
& & & -\frac{\Delta x_{j_m}}{\Delta x_i} & & & 1
\end{array}\right].
\end{aligned}$$

Now, let's look at the systems of equations that need to be solved in the next iteration. Using tildes to denote items associated with the next iteration, we see that we need to solve

$$\tilde{B}\Delta\tilde{x}_{\mathcal{B}} = \tilde{a}_j \qquad \text{and} \qquad \tilde{B}^T\tilde{v} = \tilde{e}_i$$

(actually, we should probably put the tilde on the j instead of the a_j and on the i instead of the e_i, but doing so seems less aesthetically appealing, even though it's more correct). Recalling that $\tilde{B} = BE$, we see that the first system is equivalent to

$$BE\Delta\tilde{x}_{\mathcal{B}} = \tilde{a}_j,$$

which can be solved in two stages:

$$\begin{aligned} Bu &= \tilde{a}_j, \\ E\Delta\tilde{x}_{\mathcal{B}} &= u. \end{aligned}$$

Of course, the second system (involving E) is trivial, since we have an explicit formula for the inverse of E:

$$\begin{aligned} \Delta\tilde{x}_{\mathcal{B}} &= E^{-1}u \\ &= u - \frac{u_i}{\Delta x_i}(\Delta x_{\mathcal{B}} - e_i) \end{aligned}$$

(where, in keeping with our tradition, we have used u_i to denote the element of u associated with the basic variable x_i—that is, u_i is the pth entry of u).

The system involving $\tilde{B}^T$ is handled in the same manner. Indeed, first we rewrite it as

$$E^T B^T\tilde{v} = \tilde{e}_i$$

and then observe that it too can be solved in two steps:

$$\begin{aligned} E^T u &= \tilde{e}_i, \\ B^T\tilde{v} &= u. \end{aligned}$$

This time, the first step is the trivial one[1]:

$$\begin{aligned} u &= E^{-T}\tilde{e}_i \\ &= \tilde{e}_i - e_i\frac{(\Delta x_{\mathcal{B}} - e_i)^T\tilde{e}_i}{\Delta x_i}. \end{aligned}$$

Note that the fraction in the preceding equation is a scalar, and so this final expression for u shows that it is a vector with at most two nonzeros—that is, the result is utterly trivial even if the formula looks a little bit cumbersome.

We end this section by returning briefly to our example. Suppose that $\tilde{B}$ is B with column 3 replaced by the vector a_j given in (8.3). Suppose that

$$\tilde{a}_j = \begin{bmatrix} 5 & 0 & 0 & 0 & -1 \end{bmatrix}^T.$$

To solve $\tilde{B}\Delta\tilde{x}_{\mathcal{B}} = \tilde{a}_j$, we first solve $Bu = \tilde{a}_j$ using our LU-factorization of B. The result of the forward and backward substitutions is

[1]Occasionally we use the superscript $-T$ for the transpose of the inverse of a matrix. Hence, $E^{-T} = (E^{-1})^T$.

$$u = \begin{bmatrix} 0 & 3 & 1 & -3 & -\frac{1}{2} \end{bmatrix}^T.$$

Next, we solve for $\Delta\tilde{x}_{\mathcal{B}} = E^{-1}u$:

$$\Delta\tilde{x}_{\mathcal{B}} = u - \frac{u_3}{\Delta x_3}(\Delta x_{\mathcal{B}} - e_3) = \begin{bmatrix} 0 \\ 3 \\ 1 \\ -3 \\ -\frac{1}{2} \end{bmatrix} - \frac{1}{2}\left(\begin{bmatrix} -1 \\ 0 \\ 2 \\ 1 \\ -\frac{1}{2} \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} \frac{1}{2} \\ 3 \\ \frac{1}{2} \\ -\frac{7}{2} \\ -\frac{1}{4} \end{bmatrix}.$$

Of course, it is easy to check that we have gotten the correct answer: simply multiply $\tilde{B}$ times $\Delta\tilde{x}_{\mathcal{B}}$ and check that it equals $\tilde{a}_j$. It does.

4. Performance Tradeoffs

The idea of writing the next basis as a product of the current basis times an easily invertible matrix can be extended over several iterations. For example, if we look k iterations out, we can write

$$B_k = B_0 E_0 E_1 \cdots E_{k-1}.$$

If we have an LU-factorization of B_0 and we have saved enough information to reconstruct each E_j, then we can use this product to solve systems of equations involving B_k.

Note that in order to reconstruct E_j, all we need to save is the primal step direction vector $\Delta x^j_{\mathcal{B}}$ (and an integer telling which column it goes in). In actual implementations, these vectors are stored in lists. For historical reasons, this list is called an *eta-file* (and the matrices E_j are called *eta matrices*). Given the LU-factorization of B_0 and the eta-file, it is an easy matter to solve systems of equations involving either B or B^T. However, as k gets large, the amount of work required to go through the entire eta-file begins to dominate the amount of work that would be required to simply form a new LU-factorization of the current basis. Hence, the best strategy is to use an eta-file but with periodic refactorization of the basis (and accompanied purging of the eta-file).

The question then becomes: how often should one recompute a factorization of the current basis? To answer this question, suppose that we know that it takes F arithmetic operations to form an LU-factorization (of a typical basis for the problem at hand), S operations to do one forward/backward substitution, and E operations to multiply by the inverse of one eta-matrix. Then the number of operations for the initial iteration of the simplex method is $F + 2S$ (since we need to do an LU-factorization and two forward/backward substitutions—one for the system involving the basis and the other for the system involving its transpose). Then, in the next iteration, we need to do two forward/backward substitutions and two eta-inverse calculations. Each subsequent iteration is the same as the previous, except that there are two extra eta-inverse calculations. Hence, the average number of arithmetic operations per iteration if we refactorize after every K iterations is

$$\begin{aligned} T(K) &= \frac{1}{K}\left((F+2S)+2(S+E)+2(S+2E)\right. \\ &\qquad \left.+\cdots+2(S+(K-1)E)\right) \\ &= \frac{1}{K}F+2S+(K-1)E. \end{aligned}$$

Treating K as a real variable for the moment, we can differentiate this expression with respect to K, set the derivative equal to zero, and solve for K to get an estimate for the optimal choice of K:

$$K=\sqrt{\frac{F}{E}}.$$

As should be clear from our earlier discussions, E is of order m and, if the basis matrix is dense, F is of order m^3. Hence, for dense matrices, our estimates would indicate that refactorizations should take place every m iterations or so. However, for sparse matrices, F will be substantially less that m^3—more like a constant times m^2—which would indicate that refactorizations should occur on the order of every $\sqrt{m}$ iterations. In practice, one typically allows the value of K to be a user-settable parameter whose default value is set to something like 100.

5. Updating a Factorization

There is an important alternative to the eta-matrix method for reusing an LU-factorization, which we shall describe in this section and the next. As always, it is easiest to work with an example, so let's continue with the same example we've been using throughout this chapter.

Recall that the matrix $\tilde{B}$ is simply B with its third column replaced by the vector a_j given in (8.3):

$$\tilde{B}=\begin{array}{c} \\ 1\\2\\3\\4\\5 \end{array}\!\!\begin{array}{c} \begin{array}{ccccc}1&2&3&4&5\end{array}\\ \left[\begin{array}{ccccc} 2 & & \boxed{7} & & -2\\ 3 & 1 & -2 & 1 & \\ -1 & & & & -2\\ & -1 & 3 & & -6\\ & & & & 4 \end{array}\right] \end{array} = \begin{array}{c} \\ 4\\5\\3\\1\\2 \end{array}\!\!\begin{array}{c} \begin{array}{ccccc}2&3&1&5&4\end{array}\\ \left[\begin{array}{ccccc} -1 & \boxed{3} & & -6 & \\ & & & 4 & \\ & & -1 & -2 & \\ & & 7 & 2 & -2\\ 1 & -2 & 3 & & 1 \end{array}\right] \end{array}.$$

(Note that we've highlighted the new column by putting a box around it.)

Since $L^{-1}B=U$ and $\tilde{B}$ differs from B in only one column, it follows that $L^{-1}\tilde{B}$ coincides with U except for the column that got changed. And, since this column got replaced by a_j, it follows that this column of $L^{-1}\tilde{B}$ contains $L^{-1}a_j$, which we've already computed and found to be given by (8.4). Hence,

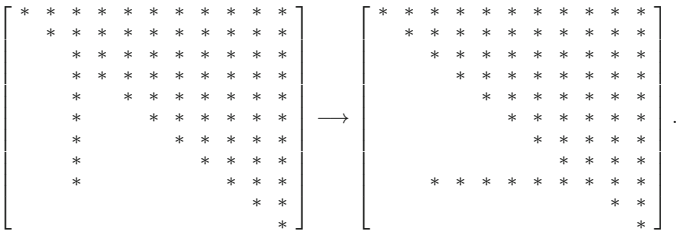
$$L^{-1}\tilde{B}=\begin{array}{c} \\ 1\\2\\3\\4\\5 \end{array}\!\!\begin{array}{c} \begin{array}{ccccc}2&3&1&5&4\end{array}\\ \left[\begin{array}{ccccc} -1 & \boxed{3} & & -6 & \\ & & & 4 & \\ & & -1 & 2 & \\ & & 7 & -14 & \\ & & 1 & & 1 \end{array}\right] \end{array}. \tag{8.6}$$

As we saw before, the columns of L have no "original" indices to relate back to, and so one can simply take them to be numbered in natural order. The same is then true for the rows of L^{-1} and hence for the rows of $L^{-1}\tilde{B}$. That is why the rows shown above are numbered as they are. We've shown these numbers explicitly, since they are about to get permuted.

The boxed column in (8.6) is called a *spike*, since it has nonzeros below the diagonal. The 4×4 submatrix constructed from rows 2 through 5 and columns 3, 1, 5, and 4 is called the *bump*. To get this matrix back into upper-triangular form, one could do row operations to eliminate the nonzeros in the spike that lie below the diagonal. But such row operations could create fill-in anywhere in the bump. Such fill-in would be more than one would like to encounter. However, consider what happens if the spike column is moved to the rightmost column of the bump, shifting the other columns left one position in the process, and if the top row of the bump (i.e., row 2) is moved to the bottom of the bump, shifting the other bump rows up by one. The result of these permutations is

$$L^{-1}\tilde{B} = \begin{array}{c} \\ 1 \\ 3 \\ 4 \\ 5 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 1 & 5 & 4 & 3 \end{array} \\ \left[\begin{array}{ccccc} -1 & & -6 & & 3 \\ & \boxed{\begin{array}{cc} -1 & 2 \\ & -14 \end{array}} & & & \boxed{\begin{array}{c} \\ 7 \end{array}} \\ & & & 1 & 1 \\ & & 4 & & \end{array}\right] \end{array}.$$

(For future reference, we've boxed the bump.) In general, the effect of this permutation is that the column spike gets replaced by a row spike along the bottom row of the bump:



Now any fill-in produced by row operations is confined to the spike row. In our example, there is only one nonzero in the spike row, and to eliminate it we need to add $2/7$ times row 4 to it. This row operation can be represented algebraically as multiplication on the left by the matrix

$$\tilde{E} = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & \frac{2}{7} & & 1 \end{bmatrix}.$$

That is,

$$\tilde{E}L^{-1}\tilde{B} = \begin{array}{c} \\ 1 \\ 3 \\ 4 \\ 5 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 1 & 5 & 4 & 3 \end{array} \\ \begin{bmatrix} -1 & & -6 & & 3 \\ & -1 & 2 & & \\ & & -14 & & 7 \\ & & & 1 & 1 \\ & & & & 2 \end{bmatrix} \end{array}.$$

If we denote the new upper triangular matrix by $\tilde{U}$, then, solving for $\tilde{B}$, we get the following factorization of $\tilde{B}$:

$$\tilde{B} = L\tilde{E}^{-1}\tilde{U}.$$

We can use this new factorization to solve systems of equations. For example, to solve

$$\tilde{B}\Delta\tilde{x}_{\mathcal{B}} = \tilde{a}_j,$$

we first solve

$$Ly = \tilde{a}_j \tag{8.7}$$

for y. Then, given y, we compute

$$z = \tilde{E}y,$$

and finally we solve

$$\tilde{U}\Delta\tilde{x}_{\mathcal{B}} = z$$

for $\Delta\tilde{x}_{\mathcal{B}}$. It is clear from the following chain of equalities that these three steps compute $\Delta\tilde{x}_{\mathcal{B}}$:

$$\Delta\tilde{x}_{\mathcal{B}} = \tilde{U}^{-1}z = \tilde{U}^{-1}\tilde{E}y = \tilde{U}^{-1}\tilde{E}L^{-1}\tilde{a}_j = \tilde{B}^{-1}\tilde{a}_j.$$

For our example, we use forward substitution to solve (8.7) for y. The result is

$$y = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{bmatrix} 0 \\ -1 \\ -1 \\ 7 \\ -3 \end{bmatrix} = \begin{array}{c} \\ 1 \\ 3 \\ 4 \\ 5 \\ 2 \end{array} \begin{bmatrix} 0 \\ -1 \\ 7 \\ -3 \\ -1 \end{bmatrix}.$$

Next, we apply the row operations (in our case, there is only one) to get z:

$$z = \tilde{E}y = \begin{array}{c} \\ 1 \\ 3 \\ 4 \\ 5 \\ 2 \end{array} \begin{bmatrix} 0 \\ -1 \\ 7 \\ -3 \\ 1 \end{bmatrix}.$$

Finally, backward substitution using $\tilde{U}$ is performed to compute $\Delta\tilde{x}_{\mathcal{B}}$:

$$\begin{array}{c} \\ 1 \\ 3 \\ 4 \\ 5 \\ 2 \end{array}\!\!\begin{array}{c} \begin{array}{ccccc} 2 & 1 & 5 & 4 & 3 \end{array} \\ \left[\begin{array}{ccccc} -1 & & -6 & & 3 \\ & -1 & 2 & & \\ & & -14 & & 7 \\ & & & 1 & 1 \\ & & & & 2 \end{array}\right] \end{array}\!\!\begin{array}{c} \begin{array}{c} 2 \\ 1 \\ 5 \\ 4 \\ 3 \end{array} \\ \left[\begin{array}{c} \\ \\ ? \\ \\ \\ \end{array}\right] \end{array} = \begin{array}{c} \begin{array}{c} 1 \end{array} \\ \left[\begin{array}{c} 0 \\ -1 \\ 7 \\ -3 \\ 1 \end{array}\right] \end{array} \begin{array}{c} \\ 3 \\ 4 \\ 5 \\ 2 \end{array}.$$

The result of the backward substitution is

$$\Delta\tilde{x}_{\mathcal{B}} = \begin{array}{c} \begin{array}{c} 2 \\ 1 \\ 5 \\ 4 \\ 3 \end{array} \\ \left[\begin{array}{c} 3 \\ \frac{1}{2} \\ -\frac{1}{4} \\ -\frac{7}{2} \\ \frac{1}{2} \end{array}\right] \end{array} = \begin{array}{c} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \\ \left[\begin{array}{c} \frac{1}{2} \\ 3 \\ \frac{1}{2} \\ -\frac{7}{2} \\ -\frac{1}{4} \end{array}\right] \end{array},$$

which agrees with the solution we got before using eta-matrices.

6. Shrinking the Bump

There is an important enhancement to the factorization updating technique described in the previous section. After permuting rows and columns converting the spike column into a spike row, we can exploit the fact that the spike row is often very sparse (coming as it does from what was originally the top row of the bump) and do further row and column permutations to reduce the size of the bump. To see what we mean, let's look at our example. First, we note that the leftmost element of the spike row is zero (and hence that the left column of the bump is a singleton column). Therefore, we can simply declare that this column and the corresponding top row do not belong to the bump. That is, we can immediately reduce the size of the bump by one:

$$\begin{array}{c} \\ 1 \\ 3 \\ 4 \\ 5 \\ 2 \end{array}\!\!\begin{array}{c} \begin{array}{ccccc} 2 & 1 & 5 & 4 & 3 \end{array} \\ \left[\begin{array}{ccccc} -1 & & -6 & & 3 \\ & -1 & 2 & & \\ & & \boxed{\begin{array}{cc} -14 & 7 \\ & 1 \\ 4 & \end{array}} & & \\ & & & & \end{array}\right] \end{array}.$$

This idea can be extended to any column that has a single nonzero in the bump. For example, column 4 is a singleton column too. The trick now is to move this column to the leftmost column of the bump, pushing the intermediate columns to the right, and to apply the same permutation to the rows. After permuting the rows and columns like this, the bump can be reduced in size again:

$$\begin{array}{c} \\ 1 \\ 3 \\ 5 \\ 4 \\ 2 \end{array}\!\!\begin{array}{c} \begin{array}{ccccc} 2 & 1 & 4 & 5 & 3 \end{array} \\ \left[\begin{array}{ccccc} -1 & & & -6 & 3 \\ & -1 & & 2 & \\ & & 1 & & 1 \\ & & & \boxed{\begin{array}{cc} -14 & 7 \\ 4 & \end{array}} & \\ & & & & \end{array}\right] \end{array}.$$

Furthermore, this reduction in the bump causes a new singleton column to appear (since a singleton need only be a singleton within the bump), namely, column 3. Hence, we permute once again. This time just the column gets permuted, since the singleton is already in the correct row. The bump gets reduced in size once again, now to a 1×1 bump, which is not really a bump at all:

$$\begin{array}{c} \\ 1 \\ 3 \\ 5 \\ 4 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 1 & 4 & 3 & 5 \end{array} \\ \left[\begin{array}{ccccc} -1 & & & 3 & -6 \\ & -1 & & & 2 \\ & & 1 & 1 & \\ & & & 7 & -14 \\ & & & & \boxed{4} \end{array}\right] \end{array}.$$

Note that we have restored upper triangularity using only permutations; no row operations were needed. While this doesn't always happen, it is a common and certainly welcome event.

Our example, being fairly small, doesn't exhibit all the possible bump-reducing permutations. In addition to looking for singleton columns, one can also look for singleton rows. Each singleton row can be moved to the bottom of the bump. At the same time, the associated column is moved to the right-hand column of the bump. After this permutation, the right-hand column and the bottom row can be removed from the bump.

Before closing this section, we reiterate a few important points. First, as the bump gets smaller, the chances of finding further singletons increases. Also, with the exception of the lower-right diagonal element of the bump, all other diagonal elements are guaranteed to be nonzero, since the matrix U from which $\tilde{U}$ is derived has this property. Therefore, most bump reductions apply the same permutation to the rows as to the columns. Finally, we have illustrated how to update the factorization once, but this technique can, of course, be applied over and over. Eventually, however, it becomes more efficient to refactorize the basis from scratch.

7. Partial Pricing

In many real-world problems, the number of constraints m is small compared with the number of variables n. Looking over the steps of the primal simplex method, we see that the only steps involving n-vectors are Step 2, in which we pick a nonbasic variable to be the entering variable,

$$\text{pick } j \in \{j \in \mathcal{N} : z_j^* < 0\};$$

Step 6, in which we compute the step direction for the dual variables,

$$\Delta z_{\mathcal{N}} = -(B^{-1}N)^T e_i;$$

and Step 8, in which we update the dual variables,

$$z_{\mathcal{N}}^* \leftarrow z_{\mathcal{N}}^* - s\Delta z_{\mathcal{N}}.$$

Scanning all the nonbasic indices in Step 2 requires looking at n candidates. When n is huge, this step is likely to be a bottleneck step for the algorithm. However, there is no requirement that all indices be scanned. We could simply scan from

the beginning and stop at the first index j for which z_j^* is negative (as in Bland's rule). However, in practice, it is felt that picking an index j corresponding to a very negative z_j^* produces an algorithm that is likely to reach optimality faster. Therefore, the following scheme, referred to as *partial pricing* is often employed. Initially, scan only a fraction of the indices (say $n/3$), and set aside a handful of good ones (say, the 40 or so having the most negative z_j^*). Then use only these 40 in Steps 2, 6, and 8 for subsequent iterations until less than a certain fraction (say, $1/2$) of them remain eligible. At this point, use (6.8) to compute the current values of a new batch of $n/3$ nonbasic dual variables, and go back to the beginning of this partial pricing process by setting aside the best 40. In this way, most of the iterations look like they only have 40 nonbasic variables. Only occasionally does the grim reality of the full huge number of nonbasic variables surface.

Looking at the dual simplex method (Figure 6.1), we see that we aren't so lucky. In it, vectors of length n arise in the max-ratio test:

$$t = \left(\max_{j \in \mathcal{N}} \frac{\Delta z_j}{z_j^*} \right)^{-1}$$

$$\text{pick } j \in \operatorname{argmax}_{j \in \mathcal{N}} \frac{\Delta z_j}{z_j^*}.$$

Here, the entire collection of nonbasic indices must be checked; otherwise, dual feasibility will be lost and the algorithm will fail. Therefore, in cases where n is huge relative to m and partial pricing is used, it is important not to use the dual simplex method as a Phase I procedure. Instead, one should use the technique of adding artificial variables as we did in Chapter 2 to force an initial feasible solution.

8. Steepest Edge

In Chapter 4, we saw that one of the drawbacks of the largest-coefficient rule is its sensitivity to the scale in which variables are quantified. In this section, we shall discuss a pivot rule that partially remedies this problem. Recall that each step of the simplex method is a step along an edge of the feasible region from one vertex to an adjacent vertex. The largest coefficient rule picks the variable that gives the largest rate of increase of the objective function. However, this rate of increase is measured in the "space of nonbasic variables" (we view the basic variables simply as dependent variables). Also, this space changes from one iteration to the next. Hence, in a certain respect, it would seem wiser to measure the rate of increase in the larger space consisting of all the variables, both basic and nonbasic. When the rate of increase is gauged in this larger space, the pivot rule is called the *steepest-edge* rule. It is the subject of this section.

Fix a nonbasic index $j \in \mathcal{N}$. We wish to consider whether x_j should be the entering variable. If it were, the step direction vector would be

$$\Delta x = \begin{bmatrix} \Delta x_{\mathcal{B}} \\ \Delta x_{\mathcal{N}} \end{bmatrix} = \begin{bmatrix} -B^{-1}Ne_j \\ e_j \end{bmatrix}.$$

This vector points out along the edge corresponding to the pivot that would result by letting x_j enter the basis. As we know, the objective function is

$$f(x) = c^T x = c_{\mathcal{B}}^T x_{\mathcal{B}} + c_{\mathcal{N}}^T x_{\mathcal{N}}.$$

The derivative of $f(x)$ in the direction of Δx is given by

$$\frac{\partial f}{\partial \Delta x} = c^T \frac{\Delta x}{\|\Delta x\|} = \frac{c_{\mathcal{B}}^T \Delta x_{\mathcal{B}} + c_{\mathcal{N}}^T \Delta x_{\mathcal{N}}}{\|\Delta x\|}.$$

The numerator is easy (and familiar):

$$\begin{aligned} c_{\mathcal{B}}^T \Delta x_{\mathcal{B}} + c_{\mathcal{N}}^T \Delta x_{\mathcal{N}} &= c_j - c_{\mathcal{B}}^T B^{-1} N e_j \\ &= \left(c_{\mathcal{N}} - (B^{-1}N)^T c_{\mathcal{B}}\right)_j \\ &= -z_j^*. \end{aligned}$$

The denominator is more troublesome:

$$\|\Delta x\|^2 = \|\Delta x_{\mathcal{B}}\|^2 + 1 = \|B^{-1} N e_j\|^2 + 1.$$

To calculate $B^{-1}Ne_j$ for every $j \in \mathcal{N}$ is exactly the same as computing the matrix $B^{-1}N$, which (as we've discussed before) is time consuming and therefore a computation we wish to avoid. But it turns out that we can compute $B^{-1}Ne_j$ for every $j \in \mathcal{N}$ once at the start (when B is essentially, if not identically, an identity matrix) and then update the norms of these vectors using a simple formula, which we shall now derive.

Let

$$\nu_k = \|B^{-1} N e_k\|^2, \qquad k \in \mathcal{N}.$$

Suppose that we know these numbers, we use them to perform one step of the simplex method, and we are now at the beginning of the next iteration. As usual, let us denote quantities in this next iteration by putting tildes on them. For example, $\tilde{B}$ denotes the new basis matrix. As we've seen before, $\tilde{B}$ is related to B by the equation $\tilde{B} = BE$, where

$$E^{-1} = I - \frac{(\Delta x_{\mathcal{B}} - e_i) e_i^T}{\Delta x_i}.$$

Now, let's compute the new ν values:

$$\begin{aligned} \tilde{\nu}_k &= a_k^T \tilde{B}^{-T} \tilde{B}^{-1} a_k \\ &= a_k^T B^{-T} E^{-T} E^{-1} B^{-1} a_k \\ &= a_k^T B^{-T} \left(I - \frac{e_i (\Delta x_{\mathcal{B}} - e_i)^T}{\Delta x_i}\right) \left(I - \frac{(\Delta x_{\mathcal{B}} - e_i) e_i^T}{\Delta x_i}\right) B^{-1} a_k. \end{aligned} \tag{8.8}$$

Recall from (8.2) that we must compute

$$v = B^{-T} e_i$$

in the course of the old iteration. If, in addition, we compute

$$w = B^{-T} \Delta x_{\mathcal{B}}$$

then, expanding out the product in (8.8) and expressing the resulting terms using v and w, we get the following formula for quickly updating the old ν's to the new ν's:

$$\tilde{\nu}_k = \nu_k - 2\frac{a_k^T v(w-v)^T a_k}{\Delta x_i} + (a_k^T v)^2 \frac{\|\Delta x_{\mathcal{B}} - e_i\|^2}{(\Delta x_i)^2}.$$

Recent computational studies using this update formula have shown that the steepest-edge rule for choosing the entering variable is competitive against, if not superior to, other pivot rules.

Exercises

8.1 (a) Without permuting rows or columns, compute the LU-factorization of

$$B = \begin{bmatrix} 2 & 5 & & 6 & \\ 1 & 1 & 3 & 9 & 6 \\ & & 2 & 6 & 4 \\ & & & 4 & 1 \\ & & -1 & -3 & -1 \end{bmatrix}. \tag{8.9}$$

(b) Solve the system $B\Delta x_{\mathcal{B}} = a_j$ where

$$a_j = \begin{bmatrix} 0 \\ 2 \\ 1 \\ 3 \\ 0 \end{bmatrix}.$$

(c) Suppose that $\tilde{B}$ is B with its second column replaced by a_j. Solve the system $\tilde{B}\Delta\tilde{x}_{\mathcal{B}} = \tilde{a}_j$ where

$$\tilde{a}_j = \begin{bmatrix} 1 \\ 0 \\ -1 \\ 0 \\ 0 \end{bmatrix}$$

using the eta-matrix method.

(d) Solve the system $\tilde{B}\Delta\tilde{x}_{\mathcal{B}} = \tilde{a}_j$ again, this time using the factorization updating method.

8.2 Use the minimum-degree ordering heuristic to find an LU-factorization of the matrix B given by (8.9).

8.3 A *permutation matrix* is a matrix of zeros and ones for which each row has one 1 and each column has one 1.

(a) Let B be an $m \times m$ matrix, and let P be a permutation matrix. Show that PB is a matrix obtained by permuting the rows of B and that BP is a matrix obtained by permuting the columns of B. Are the two permutations the same?

(b) Show that every permutation of the rows of a matrix B corresponds to multiplying B on the left by a permutation matrix.
(c) Show that for any permutation matrix P,

$$P^{-1} = P^T.$$

8.4 Explain how to use the factorization $B = LU$ to solve

$$B^T x = b.$$

Notes

Techniques for exploiting sparsity in matrix factorization have their roots in the paper by Markowitz (1957). A few standard references on matrix factorization are the books of Duff et al. (1986), Golub and VanLoan (1989), and Gill et al. (1991). The eta-matrix technique given in Section 8.3 for using an old basis to solve systems of equations involving the current basis was first described by Dantzig and Orchard-Hayes (1954). The factorization updating technique described in Section 8.5 is the method given by Forrest and Tomlin (1972). The bump reduction techniques of Section 8.6 were first introduced by Saunders (1973) and Reid (1982). The steepest-edge pivoting rule is due to Goldfarb and Reid (1977). A similar rule, known as *Devex*, was given by Harris (1973).