Chapter 2

Linear Algebra: A Constructive Approach

In Section 1.4 we sketched a geometric interpretation of the simplex method. In this chapter, we describe the basis of an algebraic interpretation that allows it to be implemented on a computer. The fundamental building block for the simplex method from linear algebra is the *Jordan exchange*. In this chapter, we describe the Jordan exchange and its implementation in MATLAB. We use it in a constructive derivation of several key results in linear algebra concerning linear independence and the solution of linear systems of equations. In the latter part of the chapter, we discuss the LU factorization, another linear algebra tool that is important in implementations of the simplex method.

In this chapter and the rest of the book, we assume basic familiarity with MATLAB. There are many books and web sites that will get you started in MATLAB; we recommend the MATLAB Primer by Sigmon & Davis (2004).

We first describe the Jordan exchange, a fundamental building block of linear algebra and the simplex algorithm for linear programming.

2.1 Jordan Exchange

Consider the following simple linear equation in the one-dimensional variables x and y:

$$y = ax.$$

The form of the equation indicates that x is the *independent* variable and y is the *dependent* variable: Given a value of x, the equation tells us how to determine the corresponding value of y. Thus we can think of the dependent variable as a function of the independent variable; that is $y(x) := ax$. If we assume that $a \neq 0$, we can reverse the roles of y and x as follows:

$$x = \tilde{a}y, \qquad \text{where } \tilde{a} = \frac{1}{a}.$$

Note that now we have a function $x(y)$ in which x is determined as a function of y. This exchange in roles between dependent and independent gives a very simple procedure for solving either the *equation* $ax - b = 0$ or the *inequality* $ax - b \geq 0$, using the following

simple equivalences:

$$\begin{array}{lllllll} ax - b = 0 & \iff & ax = y, & y = b & \iff & x = \tilde{a}y, & y = b \\ ax - b \geq 0 & \iff & ax = y, & y \geq b & \iff & x = \tilde{a}y, & y \geq b. \end{array} \tag{2.1}$$

In particular, the second formula gives an explicit characterization of the values of x that satisfy the inequality $ax - b \geq 0$, in terms of an independent variable y for which $y \geq b$.

The *Jordan exchange* is a generalization of the process above. It deals with the case in which $x \in \mathbf{R}^n$ is a *vector* of independent variables and $y \in \mathbf{R}^m$ is a *vector* of dependent variables, and we wish to exchange one of the independent variables with one of the dependent variables. The Jordan exchange plays a crucial role in solving systems of equations, linear inequalities, and linear programs. In addition, it can be used to derive fundamental results of linear algebra and linear programming.

We now demonstrate a Jordan exchange on the system $y = Ax$ by exchanging the roles of a component y_r of y and a component x_s of x. First, we write this system equation-wise as

$$y_i = A_{i1}x_1 + A_{i2}x_2 + \cdots + A_{in}x_n, \qquad i = 1, 2, \ldots, m, \tag{2.2}$$

where the independent variables are $x_1, x_2, \ldots, x_n$ and the dependent variables are $y_1, y_2, \ldots, y_m$, and the A_{ij}'s are the coefficients. We can think of the y_i's as (linear) functions of x_j's, that is

$$y_i(x) := A_{i1}x_1 + A_{i2}x_2 + \cdots + A_{in}x_n, \qquad i = 1, 2, \ldots, m, \tag{2.3}$$

or, more succinctly, $y(x) := Ax$. This system can also be represented in the following *tableau* form:

$$\begin{array}{c|cccccc} & x_1 & \cdots & x_s & \cdots & x_n \\ \hline y_1 = & A_{11} & \cdots & A_{1s} & \cdots & A_{1n} \\ \vdots & \vdots & & \vdots & & \vdots \\ y_r = & A_{r1} & \cdots & A_{rs} & \cdots & A_{rn} \\ \vdots & \vdots & & \vdots & & \vdots \\ y_m = & A_{m1} & \cdots & A_{ms} & \cdots & A_{mn} \end{array} .$$

Note that the tableau is nothing more than a compact representation of the system of equations (2.2) or the functions determining the dependent variables from the independent variables (2.3). All the operations that we perform on the tableau are just simple algebraic operations on the system of equations, rewritten to conform with the tableau representation.

We now describe the *Jordan exchange* or *pivot operation* with regard to the tableau representation. The dependent variable y_r will become independent, while x_s changes from being independent to being dependent. The process is carried out by the following three steps.

(a) Solve the rth equation

$$y_r = A_{r1}x_1 + \cdots + A_{rs}x_s + \cdots + A_{rn}x_n$$

for x_s in terms of $x_1, x_2, \ldots, x_{s-1}, y_r, x_{s+1}, \ldots, x_n$. Note that this is possible if and only if $A_{rs} \neq 0$. (A_{rs} is known as the *pivot element*, or simply the *pivot*.)

(b) Substitute for x_s in all the remaining equations.

(c) Write the new system in a new tableau form as follows:

$$\begin{array}{cc|ccccccc} & & x_1 & \cdots & x_{s-1} & y_r & x_{s+1} & \cdots & x_n \\ y_1 & = & B_{11} & \cdots & & B_{1s} & & \cdots & B_{1n} \\ \vdots & & \vdots & & & \vdots & & & \vdots \\ y_{r-1} & = & & & & & & & \\ x_s & = & B_{r1} & \cdots & & B_{rs} & & \cdots & B_{rn} \\ y_{r+1} & = & & & & & & & \\ \vdots & & \vdots & & & \vdots & & & \vdots \\ y_m & = & B_{m1} & \cdots & & B_{ms} & & \cdots & B_{mn} \end{array}$$

To determine the elements B_{ij} in terms of the elements A_{ij}, let us carry out the algebra specified by the Jordan exchange. As will be our custom in this book, we will describe and produce corresponding MATLAB m-files for the important algebraic operations that we perform. Solution of the rth equation

$$y_r = \sum_{\substack{j=1 \\ j\neq s}}^{n} A_{rj}x_j + A_{rs}x_s$$

for x_s gives

$$x_s = \frac{1}{A_{rs}}y_r + \sum_{\substack{j=1 \\ j\neq s}}^{n} \frac{-A_{rj}}{A_{rs}}x_j = B_{rs}y_r + \sum_{\substack{j=1 \\ j\neq s}}^{n} B_{rj}x_j, \tag{2.4}$$

where

$$B_{rs} = \frac{1}{A_{rs}}, \qquad B_{rj} = \frac{-A_{rj}}{A_{rs}} \qquad \forall j \neq s. \tag{2.5}$$

These formulae define the rth row $B_{r\cdot}$ of the transformed tableau. We can express them in MATLAB by first defining `J` to represent the columns of the tableau excluding the sth column, that is

```
≫ J = [1:s-1,s+1:n];
```

and then writing

```
≫ B(r,s) = 1.0/A(r,s); B(r,J) = -A(r,J)/A(r,s);
```

This "vector index" facility is an important feature of MATLAB which enables terse coding of expressions such as those given in (2.5).

We can now proceed with part (b) of the Jordan exchange. Substituting of the expression for x_s from (2.4) in the ith equation of the tableau ($i \neq r$), we have

$$\begin{aligned} y_i &= \sum_{\substack{j=1 \\ j\neq s}}^{n} A_{ij}x_j + A_{is}\left(\frac{1}{A_{rs}}y_r + \sum_{\substack{j=1 \\ j\neq s}}^{n} \frac{-A_{rj}}{A_{rs}}x_j\right) \\ &= \sum_{\substack{j=1 \\ j\neq s}}^{n} B_{ij}x_j + B_{is}y_r, \end{aligned}$$

where

$$B_{is} = \frac{A_{is}}{A_{rs}}, \qquad B_{ij} = \left(A_{ij} - \frac{A_{is}}{A_{rs}}A_{rj}\right) = \left(A_{ij} - B_{is}A_{rj}\right) \qquad \forall i \neq r,\ j \neq s. \quad (2.6)$$

These formulae define rows $B_{i\cdot}$, $i = 1, 2, \ldots, m, i \neq r$ of the transformed tableau. We can also write these equations succinctly in MATLAB notation by defining `J` as above and defining `I` to represent all the rows of the tableau except the rth row, that is

```
≫ I = [1:r-1,r+1:m];
```

and writing

```
≫ B(I,s) = A(I,s)/A(r,s);
≫ B(I,J) = A(I,J) - B(I,s)*A(r,J);
```

The complete description of one step of the Jordan exchange with pivot A_{rs} is coded in `jx.m`—the function `jx` in MATLAB.

Note that we have introduced the "function" facility of MATLAB. Any function can be defined in a file of the same name as the function, but with suffix `.m`, just as the function `jx` is stored in `jx.m`. It can then be invoked from within MATLAB—either from the command window or from within other functions—by simply typing the function name together with its arguments. The following example shows a call to the function `jx`.

Example 2.1.1. Solve the system of equations for x_1, x_2 in terms of y_1, y_2:

$$\begin{aligned} y_1 &= 2x_1 + x_2 \\ y_2 &= 3x_1 + x_2. \end{aligned}$$

Working from the MATLAB command window, one first loads the data file containing the matrix `A` of Example 2.1.1, then invokes `jx` twice to perform the to required Jordan exchanges:

```
≫ load ex2.1.1
≫ B = jx(A,1,1)
≫ B = jx(B,2,2)
```

MATLAB FILE jx.m: Jordan exchange

```
function B = jx(A,r,s)
% syntax: B = jx(A,r,s)
% input: matrix A, integers r,s
% perform a jordan exchange with pivot A(r,s)

[m,n] = size(A); B = zeros(m,n);
I = [1:r-1,r+1:m]; J = [1:s-1,s+1:n];

% update pivot row
B(r,s) = 1.0/A(r,s); B(r,J) = -A(r,J)/A(r,s);

% update pivot column
B(I,s) = A(I,s)/A(r,s);

% update remainder of tableau
B(I,J) = A(I,J)-B(I,s)*A(r,J);
return;
```

Note that we overwrite B at each step to hold the cumulative effects of the sequence of exchanges.

We now introduce a MATLAB structure that stores a complete tableau—the row and column labels corresponding to the dependent and independent variables, along with the matrix of coefficients that defines the relationship between these quantities. The `totbl` command can be used to construct a tableau as follows:

```
≫ load ex2.1.1
≫ T = totbl(A);
```

		x_1	x_2
y_1	=	2	1
y_2	=	3	1

The row labels (dependent variables) are assigned the default values y_1 and y_2, and the column labels (independent variables) the default values x_1 and x_2; other forms of the `totbl` command, discussed later, will allow the user to define their own labels. The command `tbl` can be used to print out the tableau along with its associated labels.

To perform Jordan exchanges on the tableau representation (rather than on just the matrix), we use the labeled Jordan exchange function `ljx` in place of `jx`, as follows:

```
≫ T = ljx(T,1,1);
```

		y_1	x_2
x_1	=	0.5	−0.5
y_2	=	1.5	−0.5

```
≫ T = ljx(T,2,2);
```

$$\begin{array}{ccc} & y_1 & y_2 \\ x_1 & -1 & 1 \\ x_2 & 3 & -2 \end{array}$$

In addition to making the algebraic changes to the matrix, `ljx` swaps the row and column labels as required by the exchange, and prints the modified tableau using `tbl`. The trailing semicolon should not be omitted after the call to `ljx` since it results in the printing of additional unnecessary information about the structure `T`. ∎

The following simple theorem provides a formal justification of the Jordan exchange formulae (2.5) and (2.6), as well as their extension to multiple pivots in succession. The result will enable us to use the Jordan exchange to give some constructive derivations of key results in linear algebra and linear programming.

Theorem 2.1.1. *Consider the linear function y defined by $y(x) := Ax$, where $A \in \mathbf{R}^{m\times n}$. After k pivots (with appropriate reordering of rows and columns) denote the initial and kth tableaus as follows:*

$$\begin{array}{ccc} & x_{\mathrm{J}_1} & x_{\mathrm{J}_2} \\ y_{\mathrm{I}_1} & A_{\mathrm{I}_1\mathrm{J}_1} & A_{\mathrm{I}_1\mathrm{J}_2} \\ y_{\mathrm{I}_2} & A_{\mathrm{I}_2\mathrm{J}_1} & A_{\mathrm{I}_2\mathrm{J}_2} \end{array} \qquad \begin{array}{ccc} & y_{\mathrm{I}_1} & x_{\mathrm{J}_2} \\ x_{\mathrm{J}_1} & B_{\mathrm{I}_1\mathrm{J}_1} & B_{\mathrm{I}_1\mathrm{J}_2} \\ y_{\mathrm{I}_2} & B_{\mathrm{I}_2\mathrm{J}_1} & B_{\mathrm{I}_2\mathrm{J}_2} \end{array}$$

Here $\mathrm{I}_1, \mathrm{I}_2$ is a partition of $\{1, 2, \ldots, m\}$ and $\mathrm{J}_1, \mathrm{J}_2$ is a partition of $\{1, 2, \ldots, n\}$, with I_1 and J_1 containing the same number of elements. Then for all values of $x \in \mathbf{R}^n$

$$\begin{aligned} x_{\mathrm{J}_1} &= B_{\mathrm{I}_1\mathrm{J}_1} y_{\mathrm{I}_1}(x) + B_{\mathrm{I}_1\mathrm{J}_2} x_{\mathrm{J}_2} \\ y_{\mathrm{I}_2}(x) &= B_{\mathrm{I}_2\mathrm{J}_1} y_{\mathrm{I}_1}(x) + B_{\mathrm{I}_2\mathrm{J}_2} x_{\mathrm{J}_2}. \end{aligned}$$

That is, the original linear functions y satisfy the new linear relationships given by the kth tableau.

Proof. We show the result for one pivot. The result for k pivots follows by induction.

For a pivot on the (r, s) element, we have $\mathrm{I}_1 = \{r\}, \mathrm{I}_2 = \{1, \ldots, r-1, r+1, \ldots, m\}$, $\mathrm{J}_1 = \{s\}$ and $\mathrm{J}_2 = \{1, \ldots, s-1, s+1, \ldots, n\}$. Then for all x, we have

$$\begin{aligned} x_{\mathrm{J}_1} - B_{\mathrm{I}_1\mathrm{J}_1} y_{\mathrm{I}_1}(x) - B_{\mathrm{I}_1\mathrm{J}_2} x_{\mathrm{J}_2} &= x_s - B_{rs} y_r(x) - \sum_{\substack{j=1 \\ j\neq s}}^{n} B_{rj} x_j \\ &= x_s - \frac{1}{A_{rs}} \Big(\sum_{j=1}^{n} A_{rj} x_j\Big) - \sum_{\substack{j=1 \\ j\neq s}}^{n} \frac{-A_{rj}}{A_{rs}} x_j \\ &= 0 \end{aligned}$$

and

$$\begin{aligned}
&y_{\mathrm{I}_2}(x) - B_{\mathrm{I}_2\mathrm{J}_1} y_{\mathrm{I}_1}(x) - B_{\mathrm{I}_2\mathrm{J}_2} x_{\mathrm{J}_2} \\
&= \left[y_i(x) - B_{is} y_r(x) - \sum_{\substack{j=1 \\ j \neq s}}^{n} B_{ij} x_j \right], \qquad \begin{matrix} i = 1, 2, \ldots, m, \\ i \neq r \end{matrix} \\
&= \left[\sum_{j=1}^{n} A_{ij} x_j - \frac{A_{is}}{A_{rs}} \sum_{j=1}^{n} A_{rj} x_j - \sum_{\substack{j=1 \\ j \neq s}}^{n} (A_{ij} - \frac{A_{is} A_{rj}}{A_{rs}}) x_j \right], \qquad \begin{matrix} i = 1, 2, \ldots, m, \\ i \neq r \end{matrix} \\
&= 0,
\end{aligned}$$

verifying the claims. □

The following result shows that if two tableaus have identical dependent variables for all possible values of the independent variables, then the tableau entries are also identical.

Proposition 2.1.2. *If the linear function y is defined by $y(x) = Ax$ and also by $y(x) = Bx$, then $A = B$.*

Proof. Since $(A - B)x = 0$ for all $x \in \mathbf{R}^n$, we can choose $x = e_i$, where e_i is the ith column of the identity matrix. We deduce that the ith columns of A and B are identical. Since this fact is true for all $i = 1, 2, \ldots, n$, we conclude that $A = B$. □

2.2 Linear Independence

A simple geometric way to solve a system of two equations in two unknowns is to plot the corresponding lines and determine the point where they intersect. Of course, this technique fails when the lines are parallel to one another. A key idea in linear algebra is that of *linear dependence*, which is a generalization of the idea of parallel lines. Given a matrix $A \in \mathbf{R}^{m \times n}$, we may ask if any of its rows are redundant. In other words, is there a row $A_{k\cdot}$ that can be expressed as a linear combination of the other rows? That is,

$$A_{k\cdot} = \sum_{\substack{i=1 \\ i \neq k}}^{m} \lambda_i A_{i\cdot}. \tag{2.7}$$

If so, then the rows of A are said to be *linearly dependent*.

As a concrete illustration, consider the matrix

$$A = \begin{bmatrix} 1 & 2 & 4 \\ 3 & 4 & 8 \\ 5 & 6 & 12 \end{bmatrix}.$$

The third row of this matrix is redundant, because it can be expressed as a linear combination of the first two rows as follows:

$$A_{3\cdot} = 2A_{2\cdot} - A_{1\cdot}$$

If we rearrange this equation, we see that

$$\begin{bmatrix} -1 & 2 & -1 \end{bmatrix} A = 0,$$

that is for $z' = \begin{bmatrix} -1 & 2 & -1 \end{bmatrix}$, $z'A = 0$ with $z \neq 0$.

We define linear dependence of the rows of a matrix A formally as follows:

$$z'A = 0 \text{ for some nonzero } z \in \mathbf{R}^m.$$

(We see that (2.7) can be expressed in this form by taking $z_i = \lambda_i$, $i \neq k$, $z_k = -1$.) The negation of linear dependence is *linear independence* of the rows of A, which is defined by the implication

$$z'A = 0 \implies z = 0.$$

The idea of linear independence extends also to functions, including the linear functions y defined by $y(x) := Ax$ that we have been considering above. The functions $y_i(x)$, $i = 1, 2, \ldots, m$ defined by $y(x) := Ax$ are said to be linearly dependent if

$$z'y(x) = 0 \text{ for all } x \in \mathbf{R}^n \text{ for some nonzero } z \in \mathbf{R}^m$$

and linearly independent if

$$z'y(x) = 0 \text{ for all } x \in \mathbf{R}^n \implies z = 0. \tag{2.8}$$

The equivalence of the linear independence definitions for matrices and functions is clear when we note that

$$\begin{aligned} &z'Ax = 0 \text{ for all } x \in \mathbf{R}^n \text{ for some nonzero } z \in \mathbf{R}^m \\ \iff\ &z'A = 0 \text{ for some nonzero } z \in \mathbf{R}^m. \end{aligned}$$

Thus the functions $y(x)$ are linearly independent if and only if the rows of the matrix A are linearly independent.

Proposition 2.2.1. *If the m linear functions y_i are linearly independent then any p of them are also linearly independent, where $p \leq m$.*

Proof. Obvious from contrapositive statement: y_i, $i = 1, 2, \ldots, p$ are linearly dependent implies y_i, $i = 1, 2, \ldots, m$ are linearly dependent. □

Proposition 2.2.2. *If the linear functions y defined by $y(x) = Ax$, $A \in \mathbf{R}^{m \times n}$ are linearly independent, then $m \leq n$. Furthermore, in the tableau representation, all m dependent y_i's can be made independent, that is they can be exchanged with m independent x_j's.*

Proof. Suppose that the linear functions $y(x) = Ax$ are linearly independent. Exchange y's and x's in the tableau until no further pivots are possible, at which point we are blocked by a tableau of the following form (after a possible rearrangement of rows and columns):

$$\begin{array}{rcc} & y_{\mathrm{I}_1} & x_{\mathrm{J}_2} \\ x_{\mathrm{J}_1} = & B_{\mathrm{I}_1\mathrm{J}_1} & B_{\mathrm{I}_1\mathrm{J}_2} \\ y_{\mathrm{I}_2} = & B_{\mathrm{I}_2\mathrm{J}_1} & 0 \end{array}$$

If $\mathrm{I}_2 \neq \emptyset$, we have by Theorem 2.1.1 that

$$y_{\mathrm{I}_2}(x) = B_{\mathrm{I}_2\mathrm{J}_1} y_{\mathrm{I}_1}(x), \text{ for all } x \in \mathbf{R}^n,$$

which we can rewrite as follows:

$$\begin{bmatrix} -B_{\mathrm{I}_2\mathrm{J}_1} & I \end{bmatrix} \begin{bmatrix} y_{\mathrm{I}_1}(x) \\ y_{\mathrm{I}_2}(x) \end{bmatrix} = 0.$$

By taking z to be any row of the matrix $\begin{bmatrix} -B_{\mathrm{I}_2\mathrm{J}_1} & I \end{bmatrix}$, note that z is nonzero and that $z'y(x) = 0$. According to definition (2.8), the existence of z implies that the functions $y(x) = Ax$ are linearly dependent. Hence, we must have $\mathrm{I}_2 = \emptyset$, and therefore $m \leq n$ and all the y_i's have been pivoted to the top of the tableau, as required. □

Example 2.2.1. For illustration, consider the matrix defined earlier:

$$A = \begin{bmatrix} 1 & 2 & 4 \\ 3 & 4 & 8 \\ 5 & 6 & 12 \end{bmatrix}.$$

Then

```
≫ load ex2.2.1
≫ T = totbl(A);
```

		x_1	x_2	x_3
y_1	=	1	2	4
y_2	=	3	4	8
y_3	=	5	6	12

```
≫ T = ljx(T,1,1);
```

		y_1	x_2	x_3
x_1	=	1	−2	−4
y_2	=	3	−2	−4
y_3	=	5	−4	−8

```
≫ T = ljx(T,2,2);
```

		y_1	y_2	x_3
x_1	=	−2	1	0
x_2	=	1.5	−0.5	−2
y_3	=	−1	2	0

Note that we cannot pivot any more y's to the top, and that $y_3 = -y_1 + 2y_2$. This final relationship indicates the linear dependence relationship amongst the rows of A, namely that $A_{3\cdot} = -A_{1\cdot} + 2A_{2\cdot}$. ∎

The above result can be strengthened to the following fundamental theorem, which can be taken as an alternative and constructive definition of linear independence.

Theorem 2.2.3 (Steinitz). *For a given matrix $A \in \mathbf{R}^{m\times n}$, the linear functions y, defined by $y(x) := Ax$, are linearly independent if and only if for the corresponding tableau all the y_i's can be exchanged with some m independent x_j's.*

Proof. The "only if" part follows from Proposition 2.2.2, so we need to prove just the "if" part. If all the y_i's can be exchanged to the top of the tableau then we have (by rearranging rows and columns if necessary) that

$$\begin{array}{rcl} & \begin{array}{cc} x_{J_1} & x_{J_2} \end{array} & \\ y \;=\; & \boxed{\begin{array}{cc} A_{\cdot J_1} & A_{\cdot J_2} \end{array}} & \end{array} \qquad \longrightarrow \qquad \begin{array}{rcl} & \begin{array}{cc} y & x_{J_2} \end{array} & \\ x_{J_1} \;=\; & \boxed{\begin{array}{cc} B_{\cdot J_1} & B_{\cdot J_2} \end{array}} & \end{array}$$

Suppose now that there is some z such that $z'A = 0$. We therefore have that $z'Ax = 0$ for all $x \in \mathbf{R}^n$. In the right hand tableau above, we may set the independent variables $y = z$, $x_{J_2} = 0$, whereupon $x_{J_1} = B_{\cdot J_1} z$. For this particular choice of x and y, we have $y = Ax$ from Theorem 2.1.1, so it follows that

$$0 = z'Ax = z'y = z'z,$$

implying that $z = 0$. We have shown that the only z for which $z'A = 0$ is the zero vector $z = 0$, verifying that the rows of A and hence the functions y are linearly independent. □

A consequence of this result is that given a matrix A, the number of linearly independent rows in A is the maximum number of components of y that can be exchanged to the top of the tableau for the functions $y(x) := Ax$.

When not all the rows of A are linearly independent, we reach a tableau in which one or more of the y_i's are expressed in terms of other components of y. These relationships show the linear dependencies between the functions $y(x)$ and, therefore, between the rows of the matrix A.

Example 2.2.2. Let the matrix A be defined by

$$A = \begin{bmatrix} -1 & 0 & 3 \\ 2 & -2 & 4 \\ 0 & -2 & 10 \end{bmatrix}.$$

By using `ljx.m`, find out how many linearly independent rows A has. If there are linear dependencies, write them out explicitly.

After entering the matrix A into MATLAB, we construct a tableau and perform two Jordan exchanges to make y_1 and y_2 independent variables:

```
≫ T=totbl(A);
```

$$\begin{array}{rcl} & \begin{array}{ccc} x_1 & x_2 & x_3 \end{array} & \\ y_1 \;=\; & \boxed{\begin{array}{ccc} -1 & 0 & 3 \end{array}} & \\ y_2 \;=\; & \boxed{\begin{array}{ccc} 2 & -2 & 4 \end{array}} & \\ y_3 \;=\; & \boxed{\begin{array}{ccc} 0 & -2 & 10 \end{array}} & \end{array}$$

```
≫ T=ljx(T,2,1);
≫ T=ljx(T,1,2);
```

		y_2	y_1	x_3
x_2	=	−0.5	−1	5
x_1	=	0	−1	3
y_3	=	1	2	0

We cannot exchange y_3 with x_3 because there is a zero in the pivot position. We conclude that this matrix has two linearly independent rows. By reading across the final row of the tableau, we see that the components of y are related as follows:

$$y_3 = y_2 + 2y_1.$$

In this example, we could have done the Jordan exchanges in some other way; for example `T=ljx(T,1,1)` followed by `T=ljx(T,3,3)`. However, the relationship that we derive between the components of y will be equivalent. ∎

Exercise 2.2.3. Let

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 3 & 0 \\ 1 & 3 & -3 & -8 \end{bmatrix}.$$

Using `ljx.m`, find out how many linearly independent rows A has. By working with A', find out how many linearly independent columns it has. If there are linear dependencies, write them out explicitly.

2.3 Matrix Inversion

An $n \times n$ matrix is nonsingular if the rows of A are linearly independent, otherwise the matrix is singular.

Theorem 2.3.1. *The system $y = Ax$ with $A \in R^{n\times n}$ can be inverted to $x = By$ if and only if A is nonsingular. In this case, the matrix B is unique and is called the inverse of A and is denoted by A^{-1}. It satisfies $AA^{-1} = A^{-1}A = I$.*

Proof. Apply Steinitz's Theorem 2.2.3 to A to get B such that $x = By$. B is unique by Proposition 2.1.2. Finally, $y = Ax = ABy$, for all y shows that $I - AB = 0$, and $x = By = BAx$ for all x shows that $I - BA = 0$. □

Example 2.3.1. Invert the matrix

$$A = \begin{bmatrix} 2 & -1 & -1 \\ 0 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}$$

using MATLAB and the `ljx` function.

```
≫ load ex2.3.1
≫ T = totbl(A);
```

		x_1	x_2	x_3
y_1	=	2	−1	−1
y_2	=	0	2	−1
y_3	=	0	−1	1

```
≫ T = ljx(T,1,1);
```

		y_1	x_2	x_3
x_1	=	0.5	0.5	0.5
y_2	=	0	2	−1
y_3	=	0	−1	1

```
≫ T = ljx(T,2,2);
```

		y_1	y_2	x_3
x_1	=	0.5	0.25	0.75
x_2	=	0	0.5	0.5
y_3	=	0	−0.5	0.5

```
≫ T = ljx(T,3,3);
```

		y_1	y_2	y_3
x_1	=	0.5	1	1.5
x_2	=	0	1	1
x_3	=	0	1	2

Note that the inverse of A is now found in this tableau, that is

```
≫ invA = T.val
```

$$A^{-1} = \begin{bmatrix} 0.5 & 1 & 1.5 \\ 0 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

The command `T.val` extracts the matrix from the tableau `T` (that is, it strips off the labels).

We can check that the computed matrix is indeed A^{-1} by evaluating $AA^{-1} - I$ using the following code:

```
≫ A*invA-eye(3)
```

(where `eye(3)` is MATLAB notation for a 3×3 identity matrix.) This code should result in a 3×3 matrix whose elements are zero or very small numbers (which may not be exactly zero because of numerical roundoff error).

As an alternative method for solving this problem, we show that it is possible not to pivot along the diagonal. For the matrix B of Theorem 2.3.1 to be A^{-1}, the subscript indices of x and y must be both arranged in ascending order. If this is not the case, reordering of rows and/or columns is necessary.

```
≫ load ex2.3.1
≫ T = totbl(A);
≫ T = ljx(T,1,3);
```

		x_1	x_2	y_1
x_3	=	2	−1	−1
y_2	=	−2	3	1
y_3	=	2	−2	−1

```
≫ T = ljx(T,3,1);
```

		y_3	x_2	y_1
x_3	=	1	1	0
y_2	=	−1	1	0
x_1	=	0.5	1	0.5

```
≫ T = ljx(T,2,2);
```

		y_3	y_2	y_1
x_3	=	2	1	0
x_2	=	1	1	0
x_1	=	1.5	1	0.5

Notice that the numbers in this final tableau are identical to those obtained after the previous sequence of pivots, but that the rows and columns have been reordered according to the labels. To restore the correct ordering of the rows and columns and recover the inverse A^{-1}, we use the command

```
≫ Atemp = T.val;
```

to extract the matrix from the tableau, then use standard MATLAB commands to reorder the rows and columns of the 3×3 matrix `Atemp` to obtain A^{-1}. In this case, we note from the row labels that rows 1, 2, and 3 of T must appear as rows 3, 2, and 1 of A^{-1}, respectively; while from the column labels we see that columns 1, 2, and 3 of T must appear as rows 3, 2, and 1 of A^{-1}, respectively. We can define permutation vectors `I` and `J` to effect the reordering, and then define A^{-1} as follows:

```
≫ I=[3 2 1]; J=[3 2 1];
≫ invA(I,J)=Atemp;
```

Alternatively, we can avoid generating `Atemp` and instead use

```
≫ invA(I,J)=T.val;
```

Note that the permutations correspond to the row and column label orderings after the final Jordan exchange. That is, the vector `I` shows the final ordering of the row labels (x_3, x_2, x_1), and the vector `J` shows the final ordering of the column labels (y_3, y_2, y_1). This scheme for choosing the reordering vectors will work in general, provided we put the reordering on the left-hand side of the assignment of `T.val` to `invA`, as above. ∎

Of course, for the example above, one can avoid the final reordering step by pivoting down the diagonal in order. However, there may be problems for which such a pivot sequence is not possible. A simple example is given by the matrix

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

for which the (1, 1) element cannot be used as the first pivot. Another example follows.

Example 2.3.2. Calculate A^{-1} using `ljx`, where

$$A = \begin{bmatrix} 0 & 1 & 3 \\ 4 & 3 & 2 \\ 1 & 6 & 6 \end{bmatrix}.$$

```
≫ load ex2.3.2
≫ T = totbl(A);
```

		x_1	x_2	x_3
y_1	=	0	1	3
y_2	=	4	3	2
y_3	=	1	6	6

```
≫ T = ljx(T,3,1);
```

		y_3	x_2	x_3
y_1	=	0	1	3
y_2	=	4	−21	−22
x_1	=	1	−6	−6

```
≫ T = ljx(T,1,2);
```

		y_3	y_1	x_3
x_2	=	0	1	−3
y_2	=	4	−21	41
x_1	=	1	−6	12

```
≫ T = ljx(T,2,3);
```

		y_3	y_1	y_2
x_2	=	0.2927	−0.5366	−0.0732
x_3	=	−0.0976	0.5122	0.0244
x_1	=	−0.1707	0.1463	0.2927

We now extract the numerical values from the tableau, define the permutation vectors, and perform the reordering as follows:

```
≫ I=[2 3 1]; J=[3 1 2];
≫ invA(I,J) = T.val;
```

$$A^{-1} = \begin{bmatrix} 0.1463 & 0.2927 & -0.1707 \\ -0.5366 & -0.0732 & 0.2927 \\ 0.5122 & 0.0244 & -0.0976 \end{bmatrix}$$

∎

Exercise 2.3.3. Calculate A^{-1} using `ljx.m`, where

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 2 \\ 1 & 1 & 1 \end{bmatrix}.$$

Exercise 2.3.4. Use `ljx.m` to find the inverses of the following matrices in MATLAB.

$$A = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 1 & 1 \\ 3 & 0 & 0 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}, \qquad C = \begin{bmatrix} 2 & 0 & 1 \\ 1 & 2 & 0.5 \\ 0 & 1 & 1 \end{bmatrix}.$$

If a matrix is singular, show the linear dependence between the rows of the matrix. (Use `T.val` and perform any reordering needed on the resulting MATLAB matrix to obtain the final result.)

At this point, we note that the pivot rules described in Section 2.1 in terms of matrix elements have simple matrix block analogues. That is, instead of pivoting on A_{rs}, we could instead pivot on the submatrix A_{RS}, where $R \subseteq \{1, 2, \ldots, m\}$ and $S \subseteq \{1, 2, \ldots, n\}$ are two index sets with the same number of elements. The only real difference is that the inverse of A_{RS} is used in place of $1/A_{rs}$. MATLAB code for the block Jordan exchange is given in `bjx.m`. The code incorporates changes to the labels, so it is an extension of `ljx` rather than of `jx`. In addition, the code updates the matrix A instead of creating the new matrix B, so it needs to perform the operations in a slightly different order from the analogous operations in `ljx`.

Two successive Jordan exchanges can be effected as one block pivot of order two. Thus by induction, any sequence of Jordan exchanges can be effected by a single block pivot. What are the algebraic consequences of this observation? Consider a system that leads to the following tableau:

(2.9)

$$\boxed{\begin{array}{cc} A & B \\ C & D \end{array}}$$

where A is square and invertible. Applying a block pivot to the matrix A, the transformed tableau is:

(2.10)

$$\boxed{\begin{array}{cc} A^{-1} & -A^{-1}B \\ CA^{-1} & D - CA^{-1}B \end{array}}$$

The matrix $D - CA^{-1}B$ is called the Schur complement of A in $\begin{bmatrix} A & B \\ C & D \end{bmatrix}$. The algebraic formula for the block pivot operation will prove to be very useful in the sequel. For example, if A^{-1} exists, then the original matrix (2.9) is invertible if and only if the Schur complement $D - CA^{-1}B$ is invertible.

Exercise 2.3.5. The following tableau expresses how the dependent variables y_{I_1} and y_{I_2} can be expressed in terms of the independent variables x_{J_1} and x_{J_2}:

$$\begin{array}{ccc} & x_{J_1} & x_{J_2} \\ y_{I_1} = & \boxed{\begin{array}{cc} A & B \\ C & D \end{array}} & \\ y_{I_2} = & & \end{array}$$

MATLAB FILE bjx.m: labeled block Jordan exchange

```
function A = bjx(A,R,S)
% syntax: B = bjx(A,R,S)
% input: tableau A, integer vectors R,S
% perform a block Jordan exchange with pivot A(R,S)

R = R(:); S = S(:);
[m,n] = size(A.val);

% setdiff(1:m,R) := {1,...,m}\R
I = setdiff(1:m,R); J = setdiff(1:n,S);

% note that values are updated in place
% update pivot column
A.val(R,S) = inv(A.val(R,S));
A.val(I,S) = A.val(I,S)*A.val(R,S);

% update remainder of tableau
A.val(I,J) = A.val(I,J)-A.val(I,S)*A.val(R,J);

% update pivot row
A.val(R,J) = -A.val(R,S)*A.val(R,J);

% now update the labels
swap = A.bas(R);
A.bas(R) = A.nonbas(S);
A.nonbas(S) = swap;

if isfield(A,'dualbas')
  swap = A.dualbas(S);
  A.dualbas(S) = A.dualnonbas(R);
  A.dualnonbas(S) = swap;
end

tbl(A);

return;
```

By performing some simple manipulations, derive the following formulae:

$$x_{\mathrm{I}_1} = A^{-1}y_{\mathrm{I}_1} - A^{-1}Bx_{\mathrm{I}_2} \text{ and } y_{\mathrm{I}_2} = CA^{-1}y_{\mathrm{I}_1} + (D - CA^{-1}B)x_{\mathrm{I}_2},$$

thus justifying the block Jordan exchange formula (2.10).

2.4 Exact Solution of m Equations in n Unknowns

At this stage, we have considered only square systems where the number of variables is the same as the number of equations. In optimization applications, it is more likely that the systems under consideration have a different numbers of variables than equations. Such systems pose additional concerns. For example, the system

$$x_1 + x_2 = 1$$

having one equation and two variables has infinitely many solutions, whereas the system

$$x_1 = 1, \qquad x_1 = 2$$

clearly has no solution.

Suppose we wish to solve $Ax = b$ (or determine that no solution exists) where $A \in \mathbf{R}^{m\times n}$ and $b \in \mathbf{R}^m$, with m and n not necessarily equal. The Jordan exchange gives a simple method for solving this problem under no assumption whatever on A.

1. Write the system in the following tableau form:

$$\begin{array}{cc} & \begin{array}{cc} x & 1 \end{array} \\ y = & \left[\begin{array}{c|c} A & -b \end{array}\right] \end{array}$$

   Our aim is to seek x and y related by this tableau *such that* $y = 0$.

2. Pivot as many of the y_i's to the top of the tableau, say y_{I_1}, until no more can be pivoted, in which case we are blocked by a tableau as follows: (with row and column reordering)

$$\begin{array}{cc} & \begin{array}{cc} y_{\mathrm{I}_1} & x_{\mathrm{J}_2} \end{array} \\ \begin{array}{c} x_{\mathrm{J}_1} = \\ y_{\mathrm{I}_2} = \end{array} & \left[\begin{array}{cc|c} B_{\mathrm{I}_1\mathrm{J}_1} & B_{\mathrm{I}_1\mathrm{J}_2} & d_{\mathrm{I}_1} \\ B_{\mathrm{I}_2\mathrm{J}_1} & 0 & d_{\mathrm{I}_2} \end{array}\right] \end{array}$$

   We now ask the question: Is it possible to find x and y related by this tableau such that $y = 0$?

3. The system is solvable if and only if $d_{\mathrm{I}_2} = 0$, since *we require* $y_{\mathrm{I}_1} = 0$ *and* $y_{\mathrm{I}_2} = 0$. When $d_{\mathrm{I}_2} = 0$, we obtain by writing out the relationships in the tableau explicitly that

$$\begin{aligned} y_{\mathrm{I}_1} &= 0 \\ y_{\mathrm{I}_2} &= B_{\mathrm{I}_2\mathrm{J}_1} y_{\mathrm{I}_1} = 0 \\ x_{\mathrm{J}_2} &\quad \text{is arbitrary} \\ x_{\mathrm{J}_1} &= B_{\mathrm{I}_1\mathrm{J}_2} x_{\mathrm{J}_2} + d_{\mathrm{I}_1}. \end{aligned}$$

Note that m could be less than or greater than n.

Example 2.4.1. Solve the following system

$$\begin{array}{rcrcrcr} x_1 & - & x_2 & + & x_3 & = & 2 \\ -x_1 & + & 2x_2 & + & x_3 & = & 3 \\ x_1 & - & x_2 & - & x_3 & = & -2. \end{array}$$

Remember that our technique is to pivot as many of the y_i's to the top of the tableau as possible, then check that the resulting tableau is still valid. In particular, the elements in the last column corresponding to the y_i's that we cannot pivot to the top—the subvector d_{i_2} in the notation above—should be zero.

Rewrite the problem as follows:

$$\begin{array}{rcrcrcrcr} y_1 & = & x_1 & - & x_2 & + & x_3 & - & 2 \\ y_2 & = & -x_1 & + & 2x_2 & + & x_3 & - & 3 \\ y_3 & = & x_1 & - & x_2 & - & x_3 & + & 2, \end{array}$$

and carry out the following pivot operations on its tableau representation:

```
≫ load ex2.4.1
≫ T = totbl(A,b);
```

		x_1	x_2	x_3	1
y_1	=	1	−1	1	−2
y_2	=	−1	2	1	−3
y_3	=	1	−1	−1	2

This gives rise to the following sequence of tableaus:

```
≫ T = ljx(T,1,1);
```

		y_1	x_2	x_3	1
x_1	=	1	1	−1	2
y_2	=	−1	1	2	−5
y_3	=	1	0	−2	4

```
≫ T = ljx(T,2,2);
```

		y_1	y_2	x_3	1
x_1	=	2	1	−3	7
x_2	=	1	1	−2	5
y_3	=	1	0	−2	4

```
≫ T = ljx(T,3,3);
```

		y_1	y_2	y_3	1
x_1	=	0.5	1	1.5	1
x_2	=	0	1	1	1
x_3	=	0.5	0	−0.5	2

The final solution can be read off the tableau, by setting $y_1 = y_2 = y_3 = 0$. We get $x_1 = 1$, $x_2 = 1$ and $x_3 = 2$. Note that if the calculations are being carried out by hand, the columns of the tableau that are labeled with a y_i can be suppressed since their values are never needed. This can result in significant saving in computation time, particularly if you are performing the steps by hand.

```
≫ load ex2.4.1
≫ T = ljx(T,1,1);
≫ T = delcol(T,'y1');
```

		x_2	x_3	1
x_1	=	1	−1	2
y_2	=	1	2	−5
y_3	=	0	−2	4

```
≫ T = ljx(T,2,1);
≫ T = delcol(T,'y2');
```

$$\begin{array}{ccc|c} & & x_3 & 1 \\ x_1 & = & -3 & 7 \\ x_2 & = & -2 & 5 \\ y_3 & = & -2 & 4 \end{array}$$

```
≫ T = ljx(T,3,1);
≫ T = delcol(T,'y3');
```

$$\begin{array}{ccc|c} & & & 1 \\ x_1 & = & & 1 \\ x_2 & = & & 1 \\ x_3 & = & & 2 \end{array}$$

Of course, by deleting the y_i columns, we lose access to the linear dependence relationships between these variables. ∎

Exercise 2.4.2. Test yourself on the following example

$$\begin{aligned} x_1 + x_2 + x_3 &= 1 \\ x_1 - x_2 - x_3 &= 1 \\ x_1 - x_2 + x_3 &= 3 \end{aligned}$$

which has solution $x = (1, -1, 1)$ using

```
≫ load ex2.4.2
≫ T = totbl(A,b);
≫ ...
```

Exercise 2.4.3. Solve the following systems of equations.

1.
$$\begin{array}{rcrcrcr} 2u & + & 3v & + & 3w & = & 2 \\ & & 5v & + & 7w & = & 2 \\ 6u & + & 9v & + & 8w & = & 5 \end{array}$$

2.
$$\begin{array}{rcrcrcr} u & + & 4v & + & 2w & = & -2 \\ -2u & - & 8v & + & 3w & = & 32 \\ & & v & + & w & = & 1 \end{array}$$

Example 2.4.4. Solve the following system of 3 equations in 4 unknowns:

$$\begin{array}{rcrcrcrcr} x_1 & - & x_2 & & & + & x_4 & = & 1 \\ x_1 & & & + & x_3 & & & = & 1 \\ x_1 & + & x_2 & + & 2x_3 & - & x_4 & = & 0. \end{array}$$

The data file `ex2.4.4.mat` can be loaded into MATLAB enabling the following sequence of tableaus to be constructed:

```
≫ load ex2.4.4
≫ T = totbl(A,b);
```

		x_1	x_2	x_3	x_4	1
y_1	=	1	−1	0	1	−1
y_2	=	1	0	1	0	−1
y_3	=	1	1	2	−1	0

```
≫ T =ljx(T,2,1);
```

		y_2	x_2	x_3	x_4	1
y_1	=	1	−1	−1	1	0
x_1	=	1	0	−1	0	1
y_3	=	1	1	1	−1	1

```
≫ T =ljx(T,3,2);
```

		y_2	y_3	x_3	x_4	1
y_1	=	2	−1	0	0	1
x_1	=	1	0	−1	0	1
x_2	=	−1	1	−1	1	−1

At this point we are blocked—we cannot pivot y_1 to the top of the tableau because the pivot elements corresponding to x_3 and x_4 are both zero. (It makes no sense to exchange y_1 with either y_2 or y_3 since such a move would not increase the number of y_i's at the top of the tableau.) Since the element in the final column of the row labelled y_1 is nonzero, the system has *no solution* because, whenever $y_2 = 0$ and $y_3 = 0$, it follows that $y_1 = 1$. We are unable to set $y_1 = y_2 = y_3 = 0$ in the final tableau. In fact, we have

$$y_1(x) = 2y_2(x) - y_3(x) + 1.$$

Note that the tableau also shows that the rows of the coefficient matrix A of the original linear system are related by $A_{1\cdot} = 2A_{2\cdot} - A_{3\cdot}$, and thus are linearly dependent. ∎

Example 2.4.5. We now consider a modification of Example 2.4.4 in which the right-hand side of the first equation is changed. Our system is now as follows:

$$\begin{array}{rcrcrcrcl} x_1 & - & x_2 & & & + & x_4 & = & 2 \\ x_1 & & & + & x_3 & & & = & 1 \\ x_1 & + & x_2 & + & 2x_3 & - & x_4 & = & 0. \end{array}$$

The data file `ex2.4.5.mat` can be loaded into MATLAB enabling the following sequence of tableaus to be constructed:

```
≫ load ex2.4.5
≫ T = totbl(A,b);
```

		x_1	x_2	x_3	x_4	1
y_1	=	1	−1	0	1	−2
y_2	=	1	0	1	0	−1
y_3	=	1	1	2	−1	0

```
≫ T =ljx(T,2,1);
```

		y_2	x_2	x_3	x_4	1
y_1	=	1	−1	−1	1	−1
x_1	=	1	0	−1	0	1
y_3	=	1	1	1	−1	1

```
≫ T =ljx(T,3,2);
```

		y_2	y_3	x_3	x_4	1
y_1	=	2	−1	0	0	0
x_1	=	1	0	−1	0	1
x_2	=	−1	1	−1	1	−1

This system has *infinitely many solutions* because the final column of the tableau contains a zero in the location corresponding to the y_i column label *and* some of the x_j's are still independent variables and therefore their values can be chosen arbitrarily. Following the procedure above, we characterize the solution set by allowing x_3 and x_4 to be arbitrary, and defining

$$\begin{array}{rcrcrcl} x_1 & = & -x_3 & & & + & 1, \\ x_2 & = & -x_3 & + & x_4 & - & 1. \end{array}$$

Another way to express this result is to introduce arbitrary variables λ_1 and λ_2 (representing the arbitrary values x_3 and x_4, respectively) and writing the solution as

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 \\ -1 \\ 1 \\ 0 \end{bmatrix} \lambda_1 + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \lambda_2. \quad \blacksquare$$

Exercise 2.4.6. Using the MATLAB function `ljx`, solve the following systems of equations by carrying out all pivot operations. If a system has no solution, give a reason. If a system has infinitely many solutions, describe the solution set as in Example 2.4.5. If the rows of the matrix are linearly dependent, write down the actual linear dependence relations.

1. $Ax = a$, where

$$A = \begin{bmatrix} 2 & -1 & 1 & 1 \\ -1 & 2 & -1 & -2 \\ 4 & 1 & 1 & -1 \end{bmatrix}, \qquad a = \begin{bmatrix} 1 \\ 1 \\ 5 \end{bmatrix}.$$

2. $Bx = b$, where

$$B = \begin{bmatrix} 1 & -1 & 1 & 2 \\ 1 & 1 & 0 & -1 \\ 1 & -3 & 2 & 5 \end{bmatrix}, \qquad b = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}.$$

3. $Cx = c$, where

$$C = \begin{bmatrix} 1 & -1 & 1 \\ 2 & 1 & 1 \\ -1 & -1 & 2 \\ 1 & 1 & -1 \end{bmatrix}, \qquad c = \begin{bmatrix} 3 \\ 2 \\ 2 \\ -1 \end{bmatrix}.$$

Exercise 2.4.7. Find all solutions of $Ax = b$ where:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 0 & 1 & 0 \\ 1 & -1 & 0 & -1 \end{bmatrix}, \; b = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}.$$

Describe all solutions to the following set of equations and inequalities: $Ax = b, x_3 \geq 0$.

We end this section by deriving a fundamental theorem of linear algebra by means of the Jordan exchange. In what follows we shall refer to the initial tableau as the $y = Ax$ tableau and (if it is possible) the $x = By$ tableau as a tableau where all the y's have been pivoted to the top of the tableau.

Theorem 2.4.1. *Let $A \in \mathbf{R}^{n\times n}$. The following are equivalent:*

(a) A is nonsingular.

(b) $Ax = 0 \implies x = 0$, that is $\ker A := \{x \mid Ax = 0\} = \{0\}$.

(c) $Ax = b$ has a unique solution for each $b \in \mathbf{R}^n$.

(d) $Ax = b$ has a unique solution for some $b \in \mathbf{R}^n$.

Proof.

(a) $\implies$ (c) By Theorem 2.3.1, A^{-1} exists and $A^{-1}b$ solves $Ax = b$, for each b. Furthermore, if x^1 and x^2 both solve $Ax = b$, then $A(x^1 - x^2) = 0$. This implies that $x^1 - x^2 = A^{-1}0 = 0$, so solution is unique.

(c) $\implies$ (b) If we take $b = 0$ then (c) implies that $Ax = 0$ has a unique solution. Clearly $x = 0$ is that unique solution.

(b) $\implies$ (d) (d) is a special case of (b).

(d) $\implies$ (a) We actually prove the contrapositive $\sim$(a) $\implies$ $\sim$(d). Suppose that A is singular. Then, by definition, its rows are linearly dependent, so by Theorem 2.2.3, it will not be possible to pivot all the y_i's to the top of the tableau. By carrying out Jordan exchanges until we are blocked, we obtain a tableau of the form

$$\begin{array}{ccc|cc|c} & & & y_{\mathrm{I}_1} & x_{\mathrm{J}_2} & 1 \\ \hline x_{\mathrm{J}_1} & = & & B_{\mathrm{I}_1\mathrm{J}_1} & B_{\mathrm{I}_1\mathrm{J}_2} & d_{\mathrm{I}_1} \\ y_{\mathrm{I}_2} & = & & B_{\mathrm{I}_2\mathrm{J}_1} & 0 & d_{\mathrm{I}_2} \end{array}$$

(after a possible rearrangement). Since A is square, some x_j's will remain as column labels in this final tableau; that is, J_2 is not empty. If $d_{\mathrm{I}_2} = 0$, this final tableau indicates infinitely many solutions, since x_{J_2} is arbitrary. Alternatively, if $d_{\mathrm{I}_2} \neq 0$, there are no solutions. In no case is it possible to have a *unique* solution, so (d) cannot hold. □

Exercise 2.4.8. Find matrices A for which the number of solutions to $Ax = b$ has the following properties. Explain your answers.

1. one solution, regardless of b
2. zero or infinitely many solutions, depending on b
3. zero or one solution, depending on b
4. infinitely many solutions, independent of b.

Note that using a block pivot on an $n \times n$ system of equations as above is equivalent to the following tableau manipulation:

$$\begin{array}{c} y \\ = \end{array} \begin{array}{c} \begin{array}{cc} x & 1 \end{array} \\ \left[\begin{array}{c|c} A & -b \end{array}\right] \end{array} \qquad \begin{array}{c} x \\ = \end{array} \begin{array}{c} \begin{array}{cc} y & 1 \end{array} \\ \left[\begin{array}{c|c} A^{-1} & A^{-1}b \end{array}\right] \end{array}$$

2.5 Solving Linear Equations Efficiently

We turn our attention now to the computational cost of solving systems of equations of the form $Ax = b$, where A is a square $n \times n$, nonsingular matrix. We measure the cost by counting the floating-point operations (flops), $+$, $-$, $*$ and / required to obtain the solution x. We consider the Jordan-exchange-based approach of Sections 2.3 and 2.4, along with a more efficient approach based on the LU factorization.

The flops needed to perform one Jordan exchange are shown in the following table.

Element	Transform	flops
Pivot	B(r,s) = 1/A(r,s)	1
Pivot row	B(r,J) = -A(r,J)/A(r,s)	n-1
Pivot column	B(I,s) = A(I,s)/A(r,s)	m-1
Other elements	B(I,J) = A(I,J) - B(I,s)*A(r,J)	2(m-1)(n-1)

Thus there are $2mn - m - n + 1$ flops per Jordan exchange.

It requires n Jordan exchanges to invert an $n \times n$ matrix, each costing $2n^2 - 2n + 1$ flops, giving a total of $2n^3 - 2n^2 + n$ flops. For large n, we can approximate this count by $2n^3$. Having obtained A^{-1}, we can multiply it by b to obtain $x = A^{-1}b$. This matrix-vector multiplication requires an additional $2n^2$ flops, but for large n, this cost is small relative to the cost of calculating A^{-1}.

We now describe an alternative and faster method for solving square systems of equations called *Gaussian elimination*. This technique avoids calculating the inverse A^{-1} explicitly, and it will be used in Chapter 5 as a tool for implementing the simplex method efficiently. Gaussian elimination requires only about $1/3$ as many flops to solve $Ax = b$ as does the method above, based on Jordan exchanges.

Gaussian elimination can be described in terms of the LU factorization of the matrix A. This factorization computes a unit lower triangular matrix L, an upper triangular matrix U, and a permutation matrix P such that

$$PA = LU. \tag{2.11}$$