# Numerical Linear Algebra

Math 45 — Linear Algebra

David Arnold

David-Arnold@Eureka.redwoods.cc.ca.us

**Abstract**

In this activity you will learn how to solve systems of linear equations using LU decomposition, with both forward and back substitution. In addition, you will also be introduced to function files and a number of MATLAB's control structures. *Prerequisites: Some knowledge of how to enter vectors and matrices in* MATLAB. *Familiarity with Gaussian elimination, elementary matrices, and LU decomposition.*

## 1 Introduction

This is an interactive document designed for online viewing. We've constructed this onscreen document because we want to make a conscientious effort to cut down on the amount of paper wasted at the College. Consequently, printing of the onscreen document has been purposefully disabled. However, if you are extremely uncomfortable viewing documents onscreen, we have provided a print version. If you click on the Print Doc button, you will be transferred to the print version of the document, which you can print from your browser or the Acrobat Reader. We respectfully request that you only use this feature when you are at home. Help us to cut down on paper use at the College.

Much effort has been put into the design of the onscreen version so that you can comfortably navigate through the document. Most of the navigation tools are evident, but one particular feature warrants a bit of explanation. The section and subsection headings in the onscreen and print documents are interactive. If you click on any section or subsection header in the onscreen document, you will be transferred to an identical location in the print version of the document. If you are in the print version, you can make a return journey to the onscreen document by clicking on any section or subsection header in the print document.

Finally, the table of contents is also interactive. Clicking on an entry in the table of contents takes you directly to that section or subsection in the document.

### 1.1 Working with Matlab

This document is a working document. It is expected that you are sitting in front of a computer terminal where the Matlab software is installed. You are not supposed to read this document as if it were a short story. Rather, each time your are presented with a Matlab command, it is expected that you will enter the command, then hit the Enter key to execute the command and view the result. Furthermore, it is expected that you will ponder the result. Make sure that you completely understand why you got the result you did before you continue with the reading.

## 2 Numerical Linear Algebra

Numerical linear algebra devotes its existence to the study of numerical computations using matrices. Here is a quote from the preface of *Numerical Linear Algebra and Its Applications* by Biswa Nath Datta of Northern Illinois University.

"Numerical linear algebra, no longer a subtopic of numerical analysis, has grown into an independent topic for research and teaching. Because it is crucial to scientific computing (a major component of modern applied and engineering research), numerical linear algebra is becoming integral to courses in mathematics, computer science, and engineering."

In this activity we present a small taste of a fascinating subject. We begin with a presentation of functions.

## 3 Functions in MATLAB

Script files are simply a convenient way to gather a list of MATLAB commands you wish to execute in sequence. For example, open MATLAB's editor and enter the following code.[1]

```
format rat %rational format
A=[1,2,-3,4;...
        2,0,-1,4;...
        3,1,0,6;...
        -4,4,8,0]
b=[1;1;1;-1]
M=[A,b]
R=rref(M)
x=R(:,5)
format %back to default format
```

Save the file as `example.m`. Now, at the MATLAB prompt, enter

```
>> example
```

and each line in `example.m` will be executed in the order presented in the script file.

Function M-files are similar to script files. You still enter code to be executed in sequence. However, function files afford the user the opportunity to enter input, and functions can also return output to the calling function (in many cases MATLAB's command window).

Suppose, for example, that you would like to code the function defined by $f(x) = x^2$. Open the MATLAB editor and enter these two lines of code.

```
function y=f(x)
y=x^2;
```

Save the file as `f.m`. Now, test your function at the MATLAB prompt with the following command.

```
>> t=8;
>> z=f(t)
z =
    64
```

Note that when calling the function, you need not match the independent variable $t$ with that in the function definition, namely $x$. Nor do the names of the dependent variables have to match. We could have just as easily executed these commands.

```
>> skunk=8
skunk =
     8
```

[1] The ellipses are MATLAB's line continuation character. Everything in a line following a % character is a comment. It is ignored by the compiler.

```
>> lots_of_skunks=f(skunk)
lots_of_skunks =
    64
```

Similarly, you are free to change the function name.

```
function y=square(x)
y=x^2;
```

However, MATLAB requires that you save the file with the same name. Thus, this file must be saved as `square.m`. This function will behave in precisely the same way as our previous definition. Only the names have changed.

```
>> skunk=8
skunk =
     8
>> lots_of_skunks=square(skunk)
lots_of_skunks =
    64
```

Functions can have more than one input and they can have more than one output. For example, consider the function defined by $g(x, y) = x^2 + y^2$. Code this function as follows.

```
function z=g(x,y)
z=x^2+y^2;
```

Save the file as `g.m` and execute the following at the command line.

```
 >> u=3; v=4;
>> z=g(u,v)
z =
    25
```

We will soon encounter functions with more than one output.

## 4 Back Substitution

Consider the system of linear equations

$$\begin{aligned} 2x_1 + x_2 - 2x_3 &= 4 \\ -2x_2 + x_3 &= -3 \\ 4x_3 &= 8. \end{aligned} \tag{1}$$

In matrix form, this system is represented as follows.

$$\begin{bmatrix} 2 & 1 & -2 \\ 0 & -2 & 1 \\ 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ -3 \\ 8 \end{bmatrix}$$

Thus, we have written **system 1** in the form

$$U\mathbf{x} = \mathbf{c},$$

where

$$U = \begin{bmatrix} 2 & 1 & -2 \\ 0 & -2 & 1 \\ 0 & 0 & 4 \end{bmatrix}, \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \qquad \text{and} \qquad \mathbf{c} = \begin{bmatrix} 4 \\ -3 \\ 8 \end{bmatrix}.$$

Note that matrix $U$ is square $(3 \times 3)$ as the system has an equal number of equations and unknowns. The system is upper triangular, because each entry below the main diagonal in the coefficient matrix $U$ is zero. Further, note that matrix $U$ is nonsingular, having a full set of nonzero pivots on its main diagonal.

Such systems are easily solved using a technique called *back substitution*. First, solve the last equation of **system 1** for $x_3$.

$$\begin{aligned} 4x_3 &= 8 \\ x_3 &= 8/4 \\ x_3 &= 2 \end{aligned}$$

Substitute $x_3 = 2$ into the second equation in **system 1** and solve for $x_2$.

$$\begin{aligned} -2x_2 + x_3 &= -3 \\ x_2 &= (-3 - x_3)/(-2) \\ x_2 &= (-3 - 2)/(-2) \\ x_2 &= 5/2 \end{aligned}$$

Substitute $x_3 = 2$ and $x_2 = 5/2$ in the first equation of **system 1** and solve for $x_1$.

$$\begin{aligned} 2x_1 + x_2 - 2x_3 &= 4 \\ x_1 &= (4 - x_2 + 2x_3)/2 \\ x_1 &= (4 - 5/2 + 2(2))/2 \\ x_1 &= 11/4 \end{aligned}$$

In general, if matrix $U$ is square, upper triangular, and nonsingular, then for any $\mathbf{c}$, the system $U\mathbf{x} = \mathbf{c}$ has a unique solution. This solution is easily found using back substitution.

$$\begin{aligned} u_{11}x_1 + u_{12}x_2 + \cdots + u_{1n}x_n &= c_1 \\ u_{22}x_2 + \cdots + u_{2n}x_n &= c_2 \\ &\vdots \\ u_{nn}x_n &= c_n \end{aligned}$$

First, solve the last equation for $x_n$.

$$x_n = c_n/u_{nn} \tag{2}$$

Use this result and the second to last equation to solve for $x_{n-1}$.

$$\begin{aligned} u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n &= c_{n-1} \\ x_{n-1} &= (c_{n-1} - u_{n-1,n}x_n)/u_{n-1,n-1} \end{aligned}$$

Continue in this manner solving for the rest of the unknowns. Indeed, the $k$th equation

$$u_{k,k}x_k + u_{k,k+1}x_{k+1} + \cdots + u_{kn}x_n = c_k$$

leads to the solution

$$x_k = (c_k - u_{k,k+1}x_{k+1} - \cdots - u_{kn}x_n)/u_{kk}.$$

This can be written using summation notation.

$$x_k = \left(c_k - \sum_{j=k+1}^{n} u_{kj}x_j\right) \Big/ u_{kk} \tag{3}$$

It is this last equation that allows us to automate the process of back substitution.

Open MATLAB's editor and begin by naming the function and its inputs and outputs. We pass the function the coefficient matrix $U$, which must be square ($n \times n$), and the vector $\mathbf{c}$ of right-hand sides of our equations. The function returns the solution of $U\mathbf{x} = \mathbf{c}$ in the variable $\mathbf{x}$.

```
function x=backsub(U,c)
```

Next, store the size of matrix $U$ in the variables $m$ (number of rows) and $n$ (number of columns).

```
[m,n]=size(U);
```

If matrix $U$ is not square, we have a problem. The system may not have a unique solution. So we check. If the matrix $U$ is not square, then the program halts, displaying a warning message for the user.

```
if m~=n
disp('Matrix U is not square.')
return
end
```

Next, prepare some space to store the solution.

```
x=zeros(n,1);
```

Use **equation 2** to calculate $x_n$ and store the result.

```
x(n)=c(n)/U(n,n);
```

Using back substitution, we must now calculate $x_k$ for $k = n-1, n-2, \ldots, 1$. This is a repetitive task which can be implemented with a for loop (`for k=n-1:-1:1` ... `end`). The inner for loop calculates the sum in **equation 3**. Finally, **equation 3**is used to compute the value of $x_k$.

```
for k=n-1:-1:1
 sum=0;
 for j=k+1:n
  sum=sum+U(k,j)*x(j);
 end
 x(k)=(c(k)-sum)/U(k,k);
end
```

Open MATLAB's editor and enter the completed code, repeated here for emphasis and ease of transcription.

```
function x=backsub(U,c)
[m,n]=size(U);
if m~=n
    disp('Matrix U is not square.')
    return;
end
x=zeros(n,1);
x(n)=c(n)/U(n,n);
for k=n-1:-1:1
    sum=0;
    for j=k+1:n
        sum=sum+U(k,j)*x(j);
    end
    x(k)=(c(k)-sum)/U(k,k);
end
```

Save the file as `backsub.m`. We will now test our `backsub` routine with **system 1**. First, enter $U$ and $\mathbf{c}$.

```
>> U=[2,1,-2;0,-2,1;0,0,4]
U =
      2            1           -2
      0           -2            1
      0            0            4
>> c=[4;-3;8]
c =
      4
     -3
      8
```

Note that $\mathbf{c}$ is entered as a column vector. The solution is garnered with the following command.

```
>> format rat
>> x=backsub(U,c)
x =
    11/4
     5/2
      2
```

Note how this agrees with the solution of **system 1** that we found earlier with hand calculations.

## 5 Forward Substitution

Consider the system of linear equations

$$\begin{aligned} c_1 &= 4 \\ 2c_1 + c_2 &= 5 \\ -3c_1 + 2c_2 + c_3 &= -10 \end{aligned} \quad . \tag{4}$$

In matrix form, this system is represented by the matrix equation

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & 2 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \\ -10 \end{bmatrix}.$$

Thus, we have written **system 4** in the form

$$L\mathbf{c} = \mathbf{b},$$

where

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & 2 & 1 \end{bmatrix}, \qquad \mathbf{c} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}, \qquad \text{and} \qquad \mathbf{b} = \begin{bmatrix} 4 \\ 5 \\ -10 \end{bmatrix}.$$

Note that matrix $L$ is square $(3 \times 3)$ as the system has an equal number of equations and unknowns. However, this system is lower triangular, because each entry above the main diagonal in the coefficient matrix $L$ is zero. Further, note that each entry on the main diagonal is a one, a fact that greatly simplifies the calculation of the solution.

Such equations are easily solved using a technique called *forward substitution*. Begin by solving the first equation for $c_1$.

$$c_1 = 4$$

Substitute $c_1 = 4$ into the second equation in **system 4** and solve for $c_2$.

$$\begin{aligned} 2c_1 + c_2 &= 5 \\ c_2 &= 5 - 2c_1 \\ c_2 &= 5 - 2(4) \\ c_2 &= -3 \end{aligned}$$

Substitute $c_1 = 4$ and $c_2 = -3$ into the third equation in **system 4** and solve for $c_3$.

$$\begin{aligned} -3c_1 + 2c_2 + c_3 &= -10 \\ c_3 &= -10 + 3c_1 - 2c_2 \\ c_3 &= -10 + 3(4) - 2(-3) \\ c_3 &= 8 \end{aligned}$$

In general, if matrix $L$ is square, lower triangular, with ones on the main diagonal, then for any $\mathbf{b}$, the system $L\mathbf{c} = \mathbf{b}$ has a unique solution. This solution is easily solved using forward substitution.

$$\begin{aligned} c_1 &= b_1 \\ l_{21}c_1 + c_2 &= b_2 \\ &\vdots \\ l_{n1}c_1 + l_{n2}c_2 + \cdots + c_n &= b_n \end{aligned}$$

Solve the first equation for $c_1$.

$$c_1 = b_1 \tag{5}$$

Use this result to solve the second equation for $c_2$.

$$\begin{aligned} l_{21}c_1 + c_2 &= b_2 \\ c_2 &= b_2 - l_{21}c_1 \end{aligned}$$

Continue in this manner solving for the rest of the unknowns. Indeed, the $k$th equation

$$l_{k1}c_1 + l_{k2}c_2 + \cdots + l_{k,k-1}c_{k-1} + c_k = b_k$$

leads to the solution

$$c_k = b_k - l_{k1}c_1 - l_{k2}c_2 - \cdots - l_{k,k-1}c_{k-1}.$$

This can be written using summation notation.

$$c_k = b_k - \sum_{j=1}^{k-1} l_{kj}c_j$$

We present the function `forwardsub` without explanation. The reader is encouraged to use the explanation of the `backsub` routine to aid in an understanding of the algorithm before proceeding to testing the routine.[2]

```
function c=forwardsub(L,b)
[m,n]=size(L);
if m~=n
    disp('Matrix L is not square.')
    return
end
c=zeros(n,1);
```

[2] You truly understand the routine only when you can code the algorithm without referring to these pages.

```
c(1)=b(1);
for k=2:n
    sum=0;
    for j=1:k-1
        sum=sum+L(k,j)*c(j);
    end
    c(k)=b(k)-sum;
end
```

Save the file as `forwardsub.m`. We will now test our forward substitution routine with **system 4**. First, enter $L$ and $\mathbf{b}$.

```
>> L=[1,0,0;2,1,0;-3,2,1]
L =
     1     0     0
     2     1     0
    -3     2     1
>> b=[4;5;-10]
b =
      4
      5
    -10
```

Again, note that $\mathbf{b}$ is a column vector. The solution is found with this command.

```
>> c=forwardsub(L,b)
c =
      4
     -3
      8
```

Note that this agrees with the solution of **system 4** that we found earlier with hand calculations.

## 6 LU Decomposition

We have been discussing LU decomposition in class, so here it is expected that readers have been exposed to this factorization. If not, pick up a copy of *Introduction to Linear Algebra*, Strang, Wellesley-Cambridge Press, and read Section 2.6, pages 95-102. If you don't have a copy of Strang, visit your library. You can also find LU decomposition explained in a number of linear algebra books.

Consider the system of equations

$$\begin{aligned} 2x_1 + x_2 - 2x_3 &= 4 \\ 4x_1 - 3x_3 &= 5 \\ -6x_1 - 7x_2 + 12x_3 &= -10. \end{aligned}$$

In matrix form, this system is represented by the matrix equation

$$A\mathbf{x} = \mathbf{b},$$

where

$$A = \begin{bmatrix} 2 & 1 & -2 \\ 4 & 0 & -3 \\ -6 & -7 & 12 \end{bmatrix}, \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \qquad \text{and} \qquad \mathbf{b} = \begin{bmatrix} 4 \\ 5 \\ -10 \end{bmatrix}.$$

We use elementary row operations to place matrix $A$ in upper triangular form, recording multipliers in the appropriate position of a lower triangular matrix $L$ as we proceed.

Compute the first multiplier with

$$l_{21} = a_{21}/a_{11} = 4/2 = 2.$$

Subtract $l_{21}$ times row 1 from row 2.

$$E_{21}A = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & -2 \\ 4 & 0 & -3 \\ -6 & -7 & 12 \end{bmatrix} = \begin{bmatrix} 2 & 1 & -2 \\ 0 & -2 & 1 \\ -6 & -7 & 12 \end{bmatrix}$$

Compute the second multiplier with

$$l_{31} = a_{31}/a_{11} = -6/2 = -3.$$

Subtract $l_{31}$ times row 1 from row 2.

$$E_{31}(E_{21}A) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & -2 \\ 0 & -2 & 1 \\ -6 & -7 & 12 \end{bmatrix} = \begin{bmatrix} 2 & 1 & -2 \\ 0 & -2 & 1 \\ 0 & -4 & 6 \end{bmatrix}$$

Compute the third multiplier with

$$l_{32} = a^*_{32}/a^*_{22} = -4/(-2) = 2,$$

where $a^*_{22}$ and $a^*_{32}$ are the entries is row 2 column 2 and row 3 column 2 of $E_{31}E_{21}A$.

Subtract $l_{32}$ times row 2 from row 3.

$$E_{32}(E_{31}E_{21}A) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & -2 \\ 0 & -2 & 1 \\ 0 & -4 & 6 \end{bmatrix} = \begin{bmatrix} 2 & 1 & -2 \\ 0 & -2 & 1 \\ 0 & 0 & 4 \end{bmatrix} = U$$

Thus,

$$U = \begin{bmatrix} 2 & 1 & -2 \\ 0 & -2 & 1 \\ 0 & 0 & 4 \end{bmatrix}.$$

We build $L$ from the identity matrix by placing the multipliers $l_{ij}$ in the corresponding positions of $L$.

$$L = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & 2 & 1 \end{bmatrix}$$

The reader will note that $U$ and $L$ are the upper and lower triangular matrices of **Section 4** and **Section 5**, respectively. Therefore, $A = LU$ and the equation

$$A\mathbf{x} = \mathbf{b}$$

becomes

$$(LU)\mathbf{x} = \mathbf{b}$$
$$L(U\mathbf{x}) = \mathbf{b}.$$

This can be written

$$L\mathbf{c} = \mathbf{b} \qquad \text{where} \qquad U\mathbf{x} = \mathbf{c}.$$

These equations were solved in **Section 5** and **Section 4**, respectively. Hence, the solution of the system $A\mathbf{x} = \mathbf{b}$ is

$$\mathbf{x} = \begin{bmatrix} 11/4 \\ 5/2 \\ 2 \end{bmatrix}.$$

That $\mathbf{x}$ is the solution of $A\mathbf{x} = \mathbf{b}$, or of

$$\begin{bmatrix} 2 & 1 & -2 \\ 4 & 0 & -3 \\ -6 & -7 & 12 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \\ -10 \end{bmatrix},$$

is easily checked.

```
>> A=[2,1,-2;4,0,-3;-6,-7,12]
A =
      2            1           -2
      4            0           -3
     -6           -7           12

>> x=[11/4;5/2;2]
x =
    11/4
     5/2
      2

>> A*x
ans =
      4
      5
    -10
```

Note that this equals `b=[4;5;-10]`.

We now write a routine to perform the LU decomposition. The routine takes input matrix $A$, then outputs lower and upper triangular matrices $L$ and $U$ such that $A = LU$.[3]

```
function [L,U]=mylu(A)
```

Again, if $A$ is not square, we alert the user.

```
[m,n]=size(A);
if m~=n
 disp('Matrix A is not square.')
 return
end
```

Matrix $L$ begins as an identity matrix.

```
L=eye(n);
```

We march through the matrix $A$, using the pivots on the diagonal to eliminate entries below the pivots. Because there are no entries below the pivot in row $n$, column $n$, we need only apply elimination $n - 1$ times.

[3] This simple routine is based on the assumption that $A$ is square and nonsingular. It also presumes that no row exchanges are required in the elimination process.

```
for k=1:n-1
```

At the $k$th elimination step, matrix $A$ will have the following form, where we have used $a^*_{ij}$ to denote the entries, because elimination has changed them from their original $a_{ij}$ values (elimination doesn't change the first row).

$$\begin{pmatrix} a_{11} & \cdots & a_{1k} & a_{1,k+1} & \cdots & a_{1n} \\ \vdots & \ddots & \times & \times & \cdots & \times \\ 0 & \cdots & a^*_{kk} & a^*_{k,k+1} & \cdots & a^*_{kn} \\ 0 & \cdots & a^*_{k+1,k} & a^*_{k+1,k+1} & \cdots & a^*_{k+1,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & a^*_{nk} & a^*_{n,k+1} & \cdots & a^*_{nn} \end{pmatrix}$$

Note that the rows below $a^*_{kk}$ run from row $k+1$ through row $n$.

```
 for i=k+1:n
```

Determine the multiplier. It's important to note that the entries in $A$ are current, all calculated in the $k-1$ elimination steps before we arrived at this $k$th step.

```
 L(i,k)=A(i,k)/A(k,k)
```

Next, we use the multiplier to eliminate $a_{k+1,k}$. Note that we are currently in column $k$ and the elimination will affect all entries to the right of this column as we march across row $k+1$.

```
 for j=k:n
  A(i,j)=A(i,j)-L(i,k)*A(k,j);
 end
```

Close the two previous for loops.

```
 end
end
```

Matrix $A$ is now upper triangular. We need only export $U$ as $A$.

```
U=A;
```

Note that $L$ is already complete. We need do nothing special to $L$ at this stage.

Open MATLAB's editor and enter the completed code, repeated here for emphasis and ease of transcription.

```
function [L,U]=mylu(A)
[m,n]=size(A);
if m~=n
    disp('Matrix A is not square.')
    return
end
L=eye(n);
for k=1:n-1
    for i=k+1:n
        L(i,k)=A(i,k)/A(k,k);
        for j=k:n
            A(i,j)=A(i,j)-L(i,k)*A(k,j);
        end
    end
end
U=A;
```

Finally, we test our routine `mylu`. Of course, we will use

$$A = \begin{bmatrix} 2 & 1 & -2 \\ 4 & 0 & -3 \\ -6 & -7 & 12 \end{bmatrix},$$

simply because our previous work shows that $A$ factors as

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

Enter A.

```
>> A=[2,1,-2;4,0,-3;-6,-7,12]
A =
      2            1           -2
      4            0           -3
     -6           -7           12
```

Use `mylu` to perform the LU decomposition.

```
>> [L,U]=mylu(A)
L =
      1            0            0
      2            1            0
     -3            2            1

U =
      2            1           -2
      0           -2            1
      0            0            4
```

These agree perfectly with previous results.

## 7 Summary

We now summarize results. Suppose that you are given a system of $n$ equations in $n$ unknowns.

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{21}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

Write the system in matrix form

$$A\mathbf{x} = \mathbf{b},$$

where

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}, \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_3 \end{bmatrix}, \qquad \text{and} \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

Factor[4] A as

$$A = LU,$$

where $L$ is lower triangular with ones on the main diagonal and $U$ is upper triangular. The system now becomes

$$A\mathbf{x} = \mathbf{b}$$
$$LU\mathbf{x} = \mathbf{b}.$$

This can be written as

$$L\mathbf{c} = \mathbf{b} \qquad \text{where} \qquad U\mathbf{x} = \mathbf{c}.$$

Use forward substitution to solve $L\mathbf{c} = \mathbf{b}$ for $\mathbf{c}$. Finally, use back substitution to solve $U\mathbf{x} = \mathbf{c}$ for $\mathbf{x}$.

**Example 1**

Consider again the system

$$\begin{aligned} 2x_1 + x_2 - 2x_3 &= 4 \\ 4x_1 - 3x_3 &= 5 \\ -6x_1 - 7x_2 + 12x_3 &= -10. \end{aligned}$$

Write this system in the form $A\mathbf{x} = \mathbf{b}$.

$$\begin{bmatrix} 2 & 1 & -2 \\ 4 & 0 & -3 \\ -6 & -7 & 12 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \\ -10 \end{bmatrix}$$

Enter $A$ and $\mathbf{b}$.

```
>> A=[2,1,-2;4,0,-3;-6,-7,12];
>> b=[4;5;-10];
```

Factor $A$.

```
>> format rat
>> [L,U]=mylu(A)
L =
     1     0     0
     2     1     0
    -3     2     1

U =
     2     1    -2
     0    -2     1
     0     0     4
```

Use forward substitution to solve $L\mathbf{c} = \mathbf{b}$ for $\mathbf{c}$.

```
>> c=forwardsub(L,b)
c =
     4
    -3
     8
```

[4] Our simple LU decomposition routines can only work if the elimination does not involve row exchanges. There are more sophisticate LU routines that will also handle row exchanges. Type `help lu` at the MATLAB prompt to see how MATLAB handles LU decomposition.

Use back substitution to solve $U\mathbf{x} = \mathbf{c}$ for $\mathbf{x}$.

```
>> x=backsub(U,c)
x =
    11/4
     5/2
      2
```

Check[5] that you have the correct solution of $A\mathbf{x} = \mathbf{b}$.

```
>> A*x-b
ans =
      0
      0
      0
```

That's pretty convincing, don't you think?

## 8 Exercises

1. Use the method of **Section 7** to find the solution of

$$\begin{aligned} -2x_1 - 3x_3 &= 6 \\ x_1 + 2x_2 + x_3 &= 4 \\ -3x_1 + x_2 - 5x_3 &= 15. \end{aligned}$$

2. Use the method of **Section 7** to find the solution of

$$\begin{aligned} -2x_1 - 3x_2 - 4x_3 &= 12 \\ -3x_1 + x_3 &= 9 \\ 3x_1 - x_2 - x_3 &= -3. \end{aligned}$$

3. Use the method of **Section 7** to find the solution of

$$\begin{aligned} 2x_1 - 2x_3 + 4x_4 &= 8 \\ -5x_1 + x_2 - x_3 - 2x_4 &= -10 \\ -2x_1 - 2x_3 + 2x_4 &= -2 \\ -3x_1 + 4x_2 - x_3 + x_4 &= 12. \end{aligned}$$

4. Use the method of **Section 7** to find the solution of

$$\begin{aligned} x_1 + 2x_2 + 2x_3 + 2x_4 &= -6 \\ 2x_1 + 3x_2 - x_3 + x_4 &= 6 \\ -x_1 - 2x_2 + x_3 + 2x_4 &= 4 \\ x_1 + 3x_2 + x_3 - x_4 &= -9. \end{aligned}$$

5. Craft a $3 \times 3$ singular matrix $A$ having no zero entries so that the routine `mylu` fails. What error message is reported by MATLAB? Explain why the routine fails on your matrix.

6. Craft a $3 \times 3$ nonsingular matrix $A$ having no zero entries so that the routine `mylu` fails. What error message is reported by MATLAB? Explain why the routine fails on your matrix.

[5] Readers familiar with MATLAB's `rref` command might want to form the augmented matrix with `M=[A,b]` and place this result in reduced row echelon form with `R=rref(M)`. How does this result contain the solution $\mathbf{x}$ to $A\mathbf{x} = \mathbf{b}$?

7. Dr. Strang has written two routines, `slu` and `slv`, which you can find on page 100 of *Introduction to Linear Algebra, 4th ed.*, Strang, Wellesley–Cambridge Press. Enter these routines on your system, then use them to solve the following system of equations.

$$\begin{aligned} 4x_1 + 3x_2 + 3x_3 - 3x_4 &= 12 \\ x_1 + 2x_2 + x_3 - 3x_4 &= -6 \\ x_1 - x_2 + 2x_3 + x_4 &= -2 \\ x_1 + x_2 - x_3 - x_4 &= 1 \end{aligned}$$

8. If a matrix $A$ requires row exchanges during elimination, then the routine `mylu` will fail (see **exercise 6**).

   a. Try to find an LU decomposition $A = LU$ for the following matrix $A$. Use both `mylu` and Dr. Strang's `slu`. Can you explain why both of these routines fail? *Note: If you are not sure why these routines fail, remember that they do not allow for row operations. Try finding an LU decomposition for A by hand to see what happens.*

$$A = \begin{pmatrix} -1 & -1 & -1 & 1 \\ 1 & 1 & 0 & -1 \\ 2 & -1 & -1 & 0 \\ 5 & -3 & -3 & 2 \end{pmatrix}$$

   b. MATLAB's LU routine, `lu`, has the ability to handle row exchanges during elimination (unlike our simple `mylu` routine and Dr. Strang's `slu`). MATLAB's `lu` routine keeps track of row operations in a permutation matrix $P$ and finds an $L$ and $U$ so that $PA = LU$. The pertinent section of the help file follows.

```
>> help lu
 LU     LU factorization.

    [L,U,P] = LU(X) returns lower triangular matrix L, upper
    triangular matrix U, and permutation matrix P so that
    P*X = L*U.
```

   Use this command to find matrices $L$, $U$, and $P$ so that $PA = LU$. Check your result with MATLAB.

   c. Let $\mathbf{b}$ be the vector

$$\mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

   Explain how you can use `lu`, `forwardsub`, and `backsub` to find the solution of $A\mathbf{x} = \mathbf{b}$. Use these routines and MATLAB to find the solution of $A\mathbf{x} = \mathbf{b}$.