2 Vectors and Matrices in Matlab

In this chapter we will introduce vectors and matrices in Matlab. We will begin with vectors, how to store them in Matlab, then define how to multiply a vector by a scalar and add two vectors of equal length. We'll perform similar tasks for matrices.

We'll also introduce Matlab's editor, which can be used to write script files. Script files are created by taking sequences of commands that you would normally enter at the command line, writing them to a file, then executing the script at the command line prompt by entering the name of the script file. We'll also show how to work in the editor with cell mode enabled and we'll publish results to a web page on the MSEMac Server.

Finally, we'll also introduce array operations and use them to plot the graphs of functions, sets of parametric equations, and polar equations in the plane.

Table of Contents

2.1 Vectors in Matlab

Matlab gets its name from the fact that it is a

MATrix **LAB**oratory.

The basic data structure in Matlab is the matrix. Even scalars (numbers) are considered to be 1×1 matrices (1 row and 1 column).

```
>> x=3
x =
     3
>> whos('x')
  Name      Size                    Bytes  Class

  x         1x1                         8  double array
```

Note that Matlab recognizes the **Size** of the variable x as a 1-by-1 matrix.

Let's now look at how vectors are stored in Matlab.

Row Vectors

A row vector has the form

$$\mathbf{v} = \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix}, \tag{2.1}$$

where v_1, v_2, ..., v_n are usually scalars (either real or complex numbers)[2].

When you enter a row vector in Matlab, you can separate the individual elements with commas.

```
>> v=[1, 2, 3, 4, 5]
v =
     1     2     3     4     5
```

You can also delimit the individual elements with spaces.

---

[1] Copyrighted material. See: http://msenux.redwoods.edu/Math4Textbook/

[2] We will see in later work that the entries in a vector can be of other data types, such as strings, for example.

```
>> v=[1 2 3 4 5]
v =
     1     2     3     4     5
```

You can determine the length of a row vector with Matlab's **length** command.

```
 >> length(v)
ans =
     5
```

You can access the element of **v** at position k with Matlab's indexing notation **v(k)**. For example, the element in the third position of vector **v** is found as follows.

```
 >> v(3)
ans =
     3
```

You can access the 1st, 3rd, and 4th entries of vector **v** as follows.

```
>> v([1,3,4])
ans =
     1     3     4
```

You can use indexing to change the entry in the 5th position of **v** as follows.

```
>> v(5)=500
v =
     1     2     3     4   500
```

You can change the 1st, 3rd, and 5th entries as follows. Note that the vector on the right must have the same length as the area to which it is assigned.

```
>> v([1,3,5])=[-10 -20 -30]
v =
   -10     2   -20     4   -30
```

If you break the equal length rule, Matlab will respond with an error message.

```
>> v([2,4])=[100 200 300]
???  In an assignment  A(I) = B, the number of elements in B and
 I must be the same.
```

There is one exception to this rule. You may assign a single value to a range of entries in the following manner.

```
>> v([1,3,5])=0
v =
     0     2     0     4     0
```

Column Vectors

A column vector has the form

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \tag{2.2}$$

where v_1, v_2, ..., v_n are usually scalars (either real or complex numbers).

In Matlab, use the semicolon to end a row and begin a new row. This is useful in building column vectors.

```
>> w=[1;2;3]
w =
     1
     2
     3
```

The length of a column vector is determined in exactly the same way the length of a row vector is determined.

```
>> length(w)
ans =
     3
```

Indexing works the same with column vectors as it does with row vectors. You can access the second element of the vector **w** as follows.

```
>> w(2)
ans =
     2
```

You can use indexing notation to change the entry in the second position of **w** as follows.

```
>> w(2)=15
w =
     1
    15
     3
```

Matlab's Transpose Operator

The transpose of a row vector is a column vector, and vice-versa, the transpose of a column vector is a row vector. Mathematically, we use the following symbolism to denote the transpose of a vector.

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}^T = \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix}$$

In general, we have need of two differnt types of transpose operators: (1) a regular transpose operator, and (2) a conjugate transpose operator. To take a regular transpose, transposing a row vector to a column vector, or vice versa, a column vector to a row vector, Matlab uses the the operator .' (that's a period followed by a single apostrophe).

```
>> u=[1;3;5;7]
u =
     1
     3
     5
     7
>> u.'
ans =
     1     3     5     7
```

On the other hand, the conjugate transpose operator is a single apostrophe. Like a the regular transpose, the conjugate transpose also changes row vectors to column vectors, and vice-versa. But it also takes the complex conjugagte[3] of each entry.

```
>> c=[2+3i, i, 9, -1-2i]
c =
   2.0000 + 3.0000i   0 + 1.0000i  9.0000   -1.0000 - 2.0000i
>> c'
ans =
   2.0000 - 3.0000i
        0 - 1.0000i
   9.0000
  -1.0000 + 2.0000i
```

Increment Notation

One can easily create vectors with a constant increment between entries. You use Matlab's `start:increment:finish` construct for this purpose. In the case where no increment is supplied, the increment is understood to be 1.

```
>> x=1:10
x =
     1     2     3     4     5     6     7     8     9    10
```

As a second example, **0:0.5:10** will create a vector that contains entries starting at zero and finishing at 10, and the difference (the increment) between any two entries is 0.5.

[3] The complex conjugate of $a + bi$ is $a - bi$.

```
>> x=0:0.5:10
x =
  Columns 1 through 6
         0    0.5000    1.0000    1.5000    2.0000    2.5000
  Columns 7 through 12
    3.0000    3.5000    4.0000    4.5000    5.0000    5.5000
  Columns 13 through 18
    6.0000    6.5000    7.0000    7.5000    8.0000    8.5000
  Columns 19 through 21
    9.0000    9.5000   10.0000
```

In this case, not that the row vector is too long to be contained in the width of the screen. Thus, the result wraps around. We can keep track of the column we're in by noting the headers. For example, **Columns 1 through 8** indicates that we're looking at the entries 1 through eight of the row vector **x**.

Of course, if we'd rather store a column vector in **x**, we can use the transpose operator.

```
>> y=(0:0.2:1).'
y =
         0
    0.2000
    0.4000
    0.6000
    0.8000
    1.0000
```

Increment notation can be extremely useful in indexing. For example, consider the following vector **v**.

```
>> v=100:-10:10
v =
   100    90    80    70    60    50    40    30    20    10
```

We can access every entry, starting at the 4th entry and extending to the last entry in the vector with the following notation.

```
>> v(5:end)
ans =
    60    50    40    30    20    10
```

We can access the entry in the even positions of the vector as follows.

```
>> v(2:2:end)
ans =
    90    70    50    30    10
```

Suppressing Output. There are times we will want to suppress the output of a command, particularly if it is quite lengthy. For example, the output of the command **z=0:0.1:100** will have over 1000 entries. If we place a semicolon at the end of this command, output of the command to the screen is *suppressed*. That is, the vector is stored in the variable **z**, but nothing is printed to the screen.

```
>> z=0:0.1:100;
```

Initialization with Zeros

Matlab has a number of commands to build special vectors. Let's explore just a few.

Unlike most computer languages, Matlab does not require that you declare the type or the dimension of variable ahead before assigning it a value. However, the Mathworks recommends that in certain situtaions it is more efficient to initialize a variable before acessing and/or assigning values with a program.

► **Example 1.** *It is well known that the infinite series*

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots + \frac{x^n}{n!} + \cdots \tag{2.3}$$

converges to e^x. Use the first 20 terms of the series to approximate e.

We are presented with an immediate difficulty. Many mathematical notations start their indexing at zero, as we see in the summation notation for the infinite series (**2.3**). However, the first entry of every vector in Matlab has index 1, not index 0. Often, we can shift the index in a mathematical expression to alleviate this conflict. For example, we can raise the index in the summation notation in

(2.3) by 1 if we subsequently lower the index in the expression by 1. In this manner, we arrive at

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = \sum_{n=1}^{\infty} \frac{x^{n-1}}{(n-1)!}.$$

Note that this latter expression produces the same series. That is,

$$e^x = \sum_{n=1}^{\infty} \frac{x^{n-1}}{(n-1)!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots + \frac{x^n}{n!} + \cdots. \tag{2.4}$$

To find e, or equivalently, e^1, we must substitute $x = 1$ into the infinite series (2.4). We will also use the first 20 terms to find an approximation of e. Thus,

$$e^1 = \sum_{n=1}^{\infty} \frac{(1)^{n-1}}{(n-1)!} \approx \sum_{n=1}^{20} \frac{1}{(n-1)!} = 1 + 1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdot + \frac{1}{19!}. \tag{2.5}$$

We will first use Matlab's **zeros** command to initialize a column vector having 20 entries, all of which are initially zero.

```
>> S=zeros(20,1);
```

We've suppressed the output. You might want to remove the semicolon at the end of this command to see that you have a column vector with 20 entries, all of which are zero.

We'll now write a **for** loop[4] to compute the partial sums of series (2.5) and record them in the row vector **S**. In the first position of **S** we'll place the first term of the series (2.5). After the loop completes, the second entry of **S** will contain the sum of the first two terms of (2.5), the third entry of **S** will contain the sum of the first three terms of (2.5), etc. The twentieth entry of the vector **S** will contain the sum of the first twenty terms of (2.5).

We employ several new ideas on which we should comment.

1. The command **format long**[5] displays more than three times as many digits than the default format.

2. Note that several commands can be put on a single command line. We need only separate the commands with commas. In the case where we want to suppress the output, we use a semicolon instead of a comma.

[4] We'll discuss **for** loops in detail in a later section.
[5] To return to default display, type **format**.

3. The command **for k=2:20** uses **start:increment:finish** notation. Because there is no increment in **2:20**, the increment is assumed to be 1. Hence, the command **for k=2:20** sets k equal to 2 on the first pass through the loop, sets $k = 3$ on the second pass through the loop, and so on, then finally sets $k = 20$ on the last pass through the loop.

4. Matlab's command for $k!$ is **factorial(k)**.

5. The entries of **S** contain the partial sums of the series (**2.5**). Hence, to get the kth entry of **S**, we must add the kth term of the series (**2.5**) to the $(k-1)$th term of **S**.

```
>> format long
>> S(1)=1; for k=2:20, S(k)=S(k-1)+1/factorial(k-1); end, S
S =
   1.00000000000000
   2.00000000000000
   2.50000000000000
   2.66666666666667
   2.70833333333333
   2.71666666666667
   2.71805555555556
   2.71825396825397
   2.71827876984127
   2.71828152557319
   2.71828180114638
   2.71828182619849
   2.71828182828617
   2.71828182844676
   2.71828182845823
   2.71828182845899
   2.71828182845904
   2.71828182845905
   2.71828182845905
   2.71828182845905
```

Note the apparent convergence to the number 2.71828182845905.

In Matlab, the expression **exp(x)** is equivalent to the mathematical expression e^x.

```
>> exp(1)
ans =
   2.71828182845905
```

Hence, it appears that the series (2.3) converges to e, at least as far as indicated by the sum of the first twenty terms in (2.5).

The following command will plot the entries in **S** versus their indicies (1:20).

```
>> plot(S,'*')
```

Note the rapid convergence to e in **Figure 2.1**.

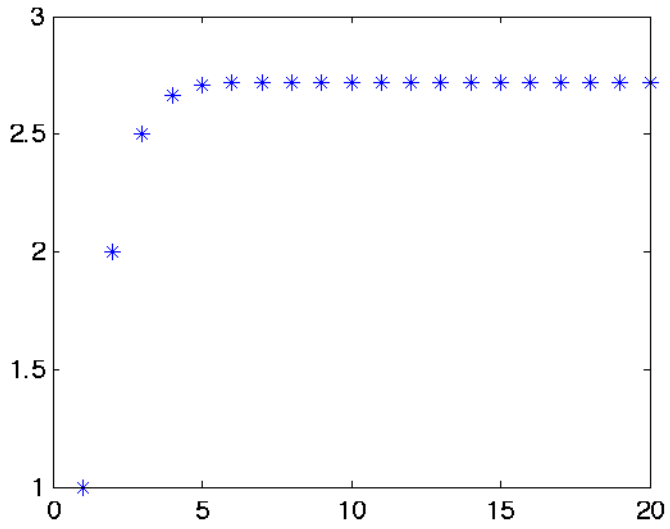


Figure 2.1. Plotting partial sums of series (2.5).

Scalar Multiplication

A scalar is a number, usually selected from the real or complex numbers. To multiply a scalar times a vector, simply multiply each entry of the vector by the scalar. In symbols,

$$\begin{aligned} \alpha\mathbf{v} &= \alpha\begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix} \\ &= \begin{bmatrix} \alpha v_1 & \alpha v_2 \dots & \alpha v_n \end{bmatrix}. \end{aligned}$$

For example,

$$2\begin{bmatrix}1 & 2 & 3 & 4 & 5\end{bmatrix} = \begin{bmatrix}2 & 4 & 6 & 8 & 10\end{bmatrix}.$$

Matlab handles scalar multiplication easily.

```
>> v=1:5
v =
     1     2     3     4     5
>> w=2*v
w =
     2     4     6     8    10
```

To multiply a column vector by a scalar, multiply each entry of the column vector by the scalar. Note that scalar multiplication is handled identically for both row and column vectors.

```
>> v=(1:3)'
v =
     1
     2
     3
>> w=2*v
w =
     2
     4
     6
```

Let's look at another example.

▶ **Example 2.** *Generate a vector that contains 1000 uniform random numbers on the interval* $[0, 10]$. *Draw a histogram of the data set contained in the vector.*

Matlab's **rand** command is used to generate uniform random numbers on the interval $[0, 1]$. By uniform, we mean any number in the interval $[0, 1]$ has an equally like chance of being selected.

```
>> v=rand(1000,1);
```

The histogram in **Figure 2.2** is created with Matlab's **hist(v)** command.[6]

```
>> hist(v)
```

In **Figure 2.2**, note that each of the bins contains approximately 100 numbers, lending credence that any real number from the interval $[0, 1]$ has an equally likely chance of being selected.

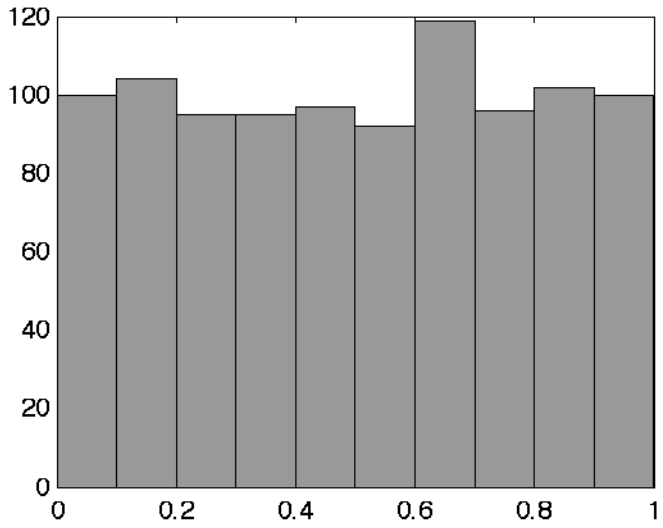


Figure 2.2. A histogram of 1000 uniform random numbers selected from the interval $[0, 1]$.

We can represent the fact that Matlab's **rand** command selects a random number from the interval $[0, 1]$ with the symbolism

$$0 \leq \text{rand} \leq 1.$$

If we want to generate uniform random numbers on the interval $[0, 10]$, multiply all three members of the last inequality by 10 to produce the result

$$0 \leq 10 \cdot \text{rand} \leq 10. \tag{2.6}$$

Hence, the expression $10 \cdot \text{rand}$ should generate uniform random numbers on the interval $[0, 10]$.

[6] Some readers might want to learn how to produce more granular control over the appearance of their histogram. If so, type **help hist** for more information.

To generate 1000 uniform random numbers from the interval $[0, 10]$, we start by generating a column vector of length 1000 with the expression **rand(1000,1)** (1000 rows and 1 column). This vector contains uniform random numbers selected from the interval $[0, 1]$. Multiply this vector by 10, which multiplies each entry by 10 to produce uniform random numbers from the interval $[0, 10]$. Matlab's **hist(v)** command produces the histogram in **Figure 2.3**

```
>> v=10*rand(1000,1); hist(v)
```

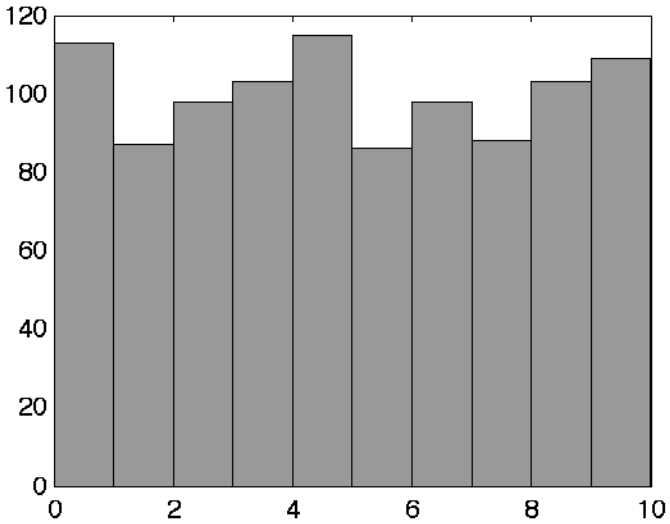


Figure 2.3. A histogram of 1000 uniform random numbers selected from the interval $[0, 10]$.

Note that each of the 10 bins of the histogram in **Figure 2.3** appears to have approximately 100 elements, lending credence to a uniform distribution, one where each real number between 0 and 10 has an equal chance of being selected.

Vector Addition

You add two row vectors of equal length by adding the corresponding entries. That is,

$$\begin{bmatrix} u_1 & u_2 & \dots & u_n \end{bmatrix} + \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix} = \begin{bmatrix} u_1 + v_1 & u_2 + v_2 & \dots & u_n + v_n \end{bmatrix}.$$

Matlab handles vector addition flawlessly.

```
>> u=1:4
u =
     1     2     3     4
>> v=5:8
v =
     5     6     7     8
>> u+v
ans =
     6     8    10    12
```

Column vectors are added in the same manner as row vectors. That is, add the corresponding entries.

```
>> u=[1,2]', v=[3,4]'
u =
     1
     2
v =
     3
     4
>> u+v
ans =
     4
     6
```

You cannot add two vectors of different lengths.

```
>> u=1:3
u =
     1     2     3
>> v=1:4
v =
     1     2     3     4
>> u+v
??? Error using ==> plus
Matrix dimensions must agree.
```

It is not legal to add a scalar to a vector. That is, the expression

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix} + 5$$

makes no sense whatsoever. However, it turns out that this is such a common requirement, Matlab allows the addition of a scalar and vector. To do so, Matlab interprets

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix} + 3$$

to mean

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix} + \begin{bmatrix}3 & 3 & 3\end{bmatrix},$$

which is easily seen in the following computation.

```
>> v=1:3
v =
     1     2     3
>> w=v+3
ans =
     4     5     6
```

Note that 3 was added to each entry of vector **v**.

Let's look at example where this feature is useful.

▶ **Example 3.** *Generate 1000 random numbers from a normal distribution with mean 100 and standard deviation 50. Produce a histogram of the data.*

Matlab's **randn** command is used to generate random numbers from the *standard* normal distribution having mean 0 and standard deviation 1.

```
>> v=randn(1000,1); hist(v)
```

The resulting histogram in **Figure 2.4** appears to possess the familiar bell-shape of the standard normal distribution. Note that the histogram appears to be centered about the number 0, lending credence to the fact that we drew numbers from the standard normal distribution, which is known to have mean zero. Further, note that most of the data (in this case all) is contained between -3 and 3. The standard deviation of the standard normal distribution is 1, and it is a a fact that most, if not all, of the data should fall within three standard deviations of the mean (which it does in **Figure 2.4**).

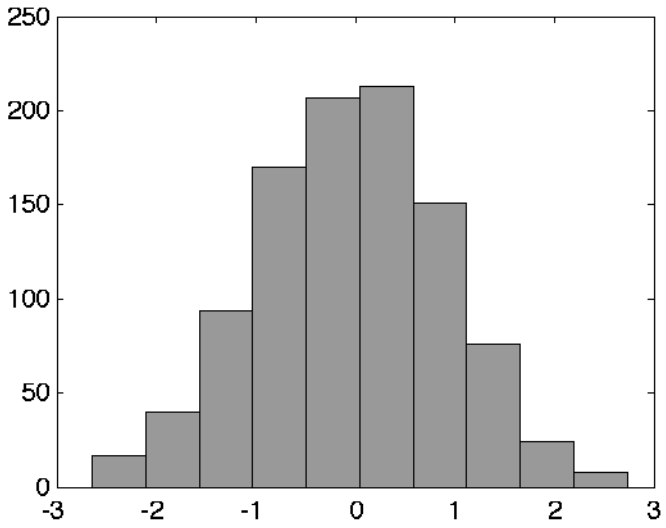


Figure 2.4. A histogram of 1000 random numbers selected from the standard normal distribution having mean 0 and standard deviation 1.

In creating **Figure 2.4**, the Matlab command **randn(1000,1)** produced numbers that fell between −3 and 3. It is reasonable to expect that if we we multiply each of these entries by 50, the range of numbers will now fall between −150 and 150. To center the new distribution about 1000, we need only shift the numbers 100 units to the right by adding 100 to every entry. The resulting histogram is shown in **Figure 2.5**.

```
>> v=100+50*randn(1000,1); hist(v)
```

Two attributes of the histogram in **Figure 2.5** seem reasonable.

1. The histogram appears to be bell-shaped and centered about 100, making the mean approximately 100 as requested.

2. Three standard deviations of 50 is 150. If we subtract this from 100, then add it to 100, then the random numbers should range from −50 to 150, which is in approximate agreement with the histogram in **Figure 2.5**.

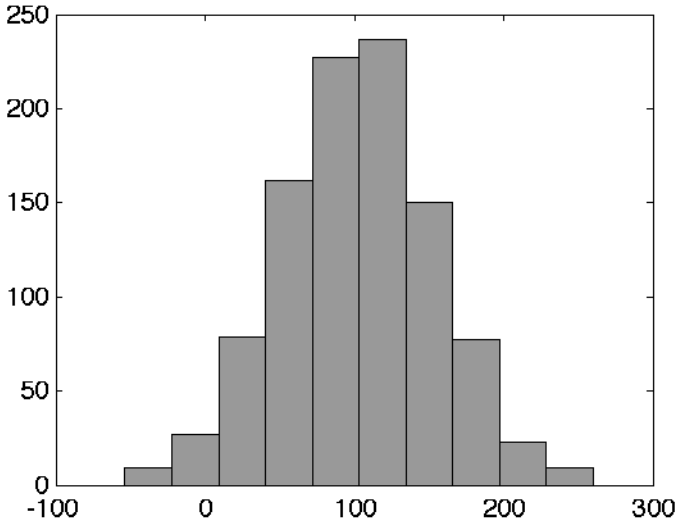


Figure 2.5. A histogram of 1000 random numbers selected from a normal distribution with mean 100 and standard deviation 50.

2.1 Exercises

In **Exercises 1-4**, use Matlab's **length** command to find the length of each of the given vectors.

1. **x=1:0.01:5**

2. **y=2:0.005:3**

3. **z=(100:0.5:200).'**

4. **x=(10:10:1000).'**

In **Exercises 5-6**, perform each of the following tasks.

i. Use Matlab's **linspace(a,b,n)** command to generate n equally spaced numbers between a and b for the given values of a, b, and n.
ii. Use Matlab's indexing notation to zero out every odd indexed entry.
iii. Use Matlab's **stem** command to plot the elements in the resulting vector versus their indices.

5. $a = 0$, $b = 10$, $n = 20$.

6. $a = -5$, $b = 5$, $n = 25$.

7. Use Matlab's **sum** command and **start:increment:finish** construct to find the sum of the integers from 1 to 1000.

8. Use Matlab's **sum** command and **start:increment:finish** construct to find the sum of the integers from 1000 to 10000.

9. Use Matlab's **sum** command and **start:increment:finish** construct to find the sum of the even integers from 1 to 1000.

10. Use Matlab's **sum** command and **start:increment:finish** construct to find the sum of the odd integers from 1 to 1000.

In **Exercises 9-14**, perform each of the following tasks for the given vector.

i. Use Matlab's **sum** command to find the sum of all entries in the given vector.
ii. Use Matlab's **sum** command and **start:increment:finish** construct to find the sum of of all the numbers with even indices in the given vector.
iii. Use Matlab's **sum** command and **start:increment:finish** construct to find the sum of of all the numbers with odd indices in the given vector.
iv. Verify that the results in part (ii) and (iii) sum to the result in part (i).

11. **x=0:0.05:10**

12. **x=10:3:120**

13. **x=(1000:10:2000).'**

14. **x=(150:5:350).'**

15. Let $\mathbf{u} = [1, 2, 3]$ and $\mathbf{v} = [4, 5, 6]$.

a) Use Matlab to compute $(\mathbf{u} + \mathbf{v})^T$.

b) Use Matlab to compute $\mathbf{u}^T + \mathbf{v}^T$ and compare to the result in part (a). Explain what you learned in this exercise.

16. Let $\mathbf{u} = [1, 2, 3]$ and $\alpha = 4$.

a) Use Matlab to compute $(\alpha\mathbf{u})^T$.

b) Use Matlab to compute $\alpha\mathbf{u}^T$ and compare to the result in part (a). Explain what you learned in this exercise.

In **Exercises 15-20**, write a simple **for** loop to populate a column vector with the first 24 terms of the given sequence. Use the **plot** command to plot the terms of the sequence versus it indices.

17. $a_n = (0.85)^n$

18. $a_n = (1.25)^n$

19. $a_n = \sin(\pi n/8)$

20. $a_n = -3\cos(\pi n/12)$

---

21. If r is a random number between 0 and 1, determine the range of the expression $(b - a)r + a$ where a and b are any real numbers with $a < b$.

In **Exercises 20-25**, perform each of the following tasks.

i. Fill a vector with 1000 uniformly distributed random numbers on the given interval. *Hint: For a helpful hint, see **Exercise 19**.*
ii. Use Matlab's **hist** command to draw a histogram of the random numbers stored in your vector.
iii. Explain why your histogram is a reasonable answer.

22. $[2, 6]$

23. $[1, 11]$

24. $[-5, 5]$

25. $[-10, 20]$

In **Exercises 24-29**, perform each of the following tasks.

i. Fill a vector with 1000 random numbers that are drawn from a normal distribution with the given mean μ and standard deviation σ.
ii. Use Matlab's **hist** command to draw a histogram of the random numbers stored in your vector.
iii. Explain why your histogram is a reasonable answer.

26. $\mu = 100$, $\sigma = 20$

27. $\mu = 50$, $\sigma = 10$

28. $\mu = 200$, $\sigma = 40$

29. $\mu = 150$, $\sigma = 30$

---

30. Read the documentation for Matlab's **primes** command, and use it to store the first 100 primes less than or equal to 1000 in a vector.

a) Find the sum of the first 100 primes.

b) Find the sum of the first, 20th, and 97th primes.

31. The **bar(v)** command generates a bar graph of the elements of the vector **v**. Assume that you are the CEO of a corporation that sells scented shoes on the Internet. You have just received data explaining that your company's profits between 1995 and 2004 were, respectively, \$2M, \$3M, \$7M, \$8M, \$14M, \$15M, \$6M, \$3M, \$2M, and \$1M. Meanwhile, your company spent a steady \$3M per year manufacturing the shoes. Use your knowledge of vectors and the **bar** command to generate a bar graph of your company's net profits[7] for 1995–2004, and decide whether or not you should keep your stock options in the company.

[7] Gross profits are the total amount earned in a given time period, while net profits take into account expenditures as well.

2.1 Answers

1.

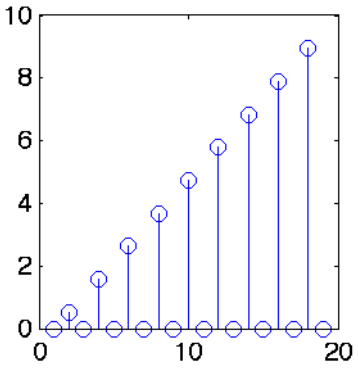
```
>> x=1:0.01:5;
>> length(x)
ans =
   401
```

3.

```
>> z=(100:0.5:200).';
>> length(z)
ans =
   201
```

5.

```
>> x=linspace(0,10,20);
>> x(1:2:end)=0;
>> stem(x)
```



7.

```
>> x=1:1000;
>> sum(x)
ans =
      500500
```

9.

```
>> x=2:2:1000;
>> sum(x)
ans =
      250500
```

11.

```
>> sum(x)
ans =
      250500
>> x=0:0.05:10;
>> sum(x(2:2:end))
ans =
   500
>> sum(x(1:2:end))
ans =
   505
>> sum(x)
ans =
   1005
```

13.

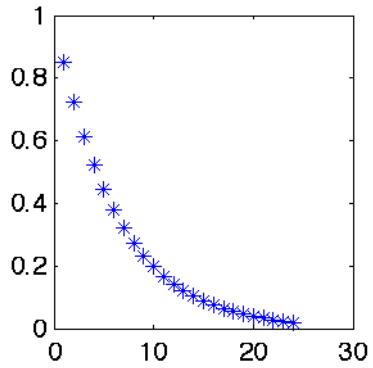
```
>> sum(x)
ans =
   1.0050e+03
>> x=(1000:20:2000).';
>> sum(x(2:2:end))
ans =
       37500
>> sum(x(1:2:end))
ans =
       39000
>> sum(x)
ans =
       76500
```

15. The transpose of a sum of two vectors is the sum of the transposes of the two vectors.

```
>> u=1:3; v=4:6;
>> (u+v).'
ans =
     5
     7
     9
>> u.'+v.'
ans =
     5
     7
     9
```

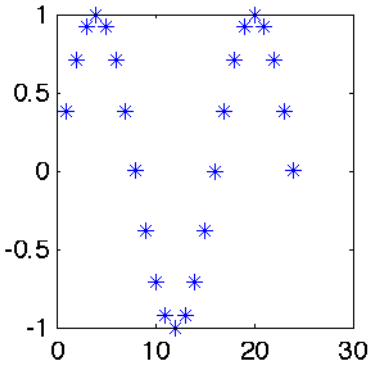
17.

```
>> a=zeros(24,1);
>> for n=1:24, a(n)=(0.85)^n;
end
>> plot(a,'*')
```



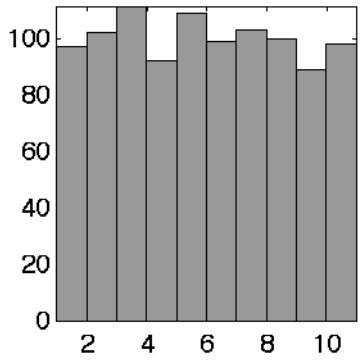
19.

```
>> a=zeros(24,1);
>> for n=1:24, a(n)=sin(pi*n/8);
end
>> plot(a,'*')
```



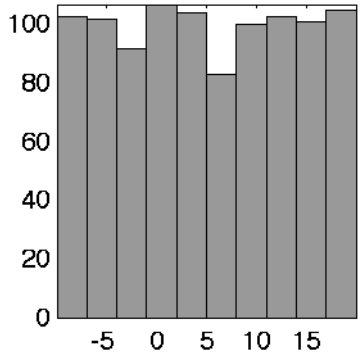
23. In the following histogram, there appears to be an equal amount of numbers in each bin over the range $[1, 11]$, lending evidence that the following set of commands select 1000 uniform random numbers from the interval $[1, 11]$.

```
>> a=1; b=11;
>> x=(b-a)*rand(1000,1)+a;
>> hist(x)
```



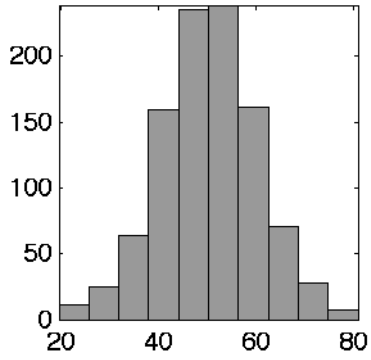
25. In the following histogram, there appears to be an equal amount of numbers in each bin over the range $[-10, 20]$, lending evidence that the following set of commands select 1000 uniform random numbers from the interval $[-10, 20]$.

```
>> a=-10; b=20;
>> x=(b-a)*rand(1000,1)+a;
>> hist(x)
```



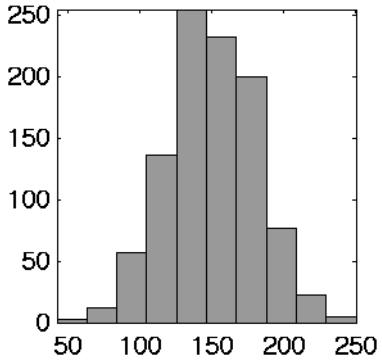
27. In the following histogram, the histogram appears to be centered around the mean $\nu = 50$. Further, if $\sigma = 10$, three standard deviations to the left of $\mu = 50$ is 10, and three standard deviations to the right of $\mu = 50$ is 80. In the histogram that follows, it would appear that the range is from 50 to 80, lending evidence that we've selected random numbers from a normal distribution with mean $\mu = 50$ and standard deviation $\sigma = 10$.

```
>> mu=50; sigma=10;
>> x=mu+sigma*randn(1000,1);
>> hist(x)
```



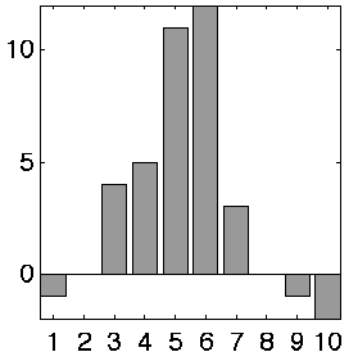
29. In the following histogram, the histogram appears to be centered around the mean $\nu = 150$. Further, if $\sigma = 30$, three standard deviations to the left of $\mu = 150$ is 60, and three standard deviations to the right of $\mu = 150$ is 240. In the histogram that follows, it would appear that the range is approximately from 50 to 250, lending evidence that we've selected random numbers from a normal distribution with mean $\mu = 150$ and standard deviation $\sigma = 30$.

```
>> mu=150; sigma=30;
>> x=mu+sigma*randn(1000,1);
>> hist(x)
```



31. The prognosis does not look good for your company's health.

```
>> v=[2 3 7 8 14 15 6 3 2 1]-
3;
>> bar(v)
```



2.2 Matrices in Matlab

You can think of a matrix as being made up of 1 or more row vectors of equal length. Equivalently, you can think of a matrix of being made up of 1 or more column vectors of equal length. Consider, for example, the matrix

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

One could say that the matrix A is made up of 3 rows of length 4. Equivalently, one could say that matrix A is made up of 4 columns of length 3. In either model, we have 3 rows and 4 columns. We will say that the dimensions of the matrix are 3-by-4, sometimes written 3×4.

We already know how to enter a matrix in Matlab: delimit each item in a row with a space or comma, and start a new row by ending a row with a semicolon.

```
>> A=[1 2 3 0;5 -1 0 0;3 -2 5 0]
A =
     1     2     3     0
     5    -1     0     0
     3    -2     5     0
```

We can use Matlab's **size** command to determine the dimensions of any matrix.

```
>> size(A)
ans =
     3     4
```

That's 3 rows and 4 columns!

Indexing

Indexing matrices in Matlab is similar to the indexing we saw with vectors. The difference is that there is another dimension[9].

To access the element in row 2 column 3 of matrix A, enter this command.

[8] Copyrighted material. See: http://msenux.redwoods.edu/Math4Textbook/

[9] We'll see later that we can have more than two dimensions.

```
>> A(2,3)
ans =
     0
```

This is indeed the element in row 2, column 3 of matrix A.

You can access an entire row with Matlab's colon operator. The command **A(2,:)** essentially means "row 2 every column" of matrix A.

```
>> A(2,:)
ans =
     5    -1     0     0
```

Note that this is the second row of matrix A.

Similarly, you can access any column of matrix A. The notation **A(:,2)** is pronounced "every row column 2" of matrix A.

```
>> A(:,2)
ans =
     2
    -1
    -2
```

Note that this is the second column of matrix A.

You can also extract a submatrix from the matrix A with indexing. Suppose, for example, that you would like to extract a submatrix using rows 1 and 3 and columns 2 and 4.

```
>> A([1,3],[2,4])
ans =
     2     0
    -2     0
```

Study this carefully and determine if we've truly selected rows 1 and 3 and columns 2 and 4 of matrix A. It might help to repeat the contents of matrix A.

```
>> A
A =
     1     2     3     0
     5    -1     0     0
     3    -2     5     0
```

You can assign a new value to an entry of matrix A.

```
>> A(3,4)=12
A =
     1     2     3     0
     5    -1     0     0
     3    -2     5    12
```

When you assign to a row, column, or submatrix of matrix A, you must replace the contents with a row, column, or submatrix of equal dimension. For example, this next command will assign new contents to the first row of matrix A.

```
>> A(1,:)=20:23
A =
    20    21    22    23
     5    -1     0     0
     3    -2     5    12
```

There is an exception to this rule. If the right side contains a single number, then that number will be assigned to every entry of the submatrix on the left. For example, to make every entry in column 2 of matrix A equal to 11, try the following code.

```
>> A(:,2)=11
A =
    20    11    22    23
     5    11     0     0
     3    11     5    12
```

It's interesting what happens (and very powerful) when you try to assign a value to an entry that has a row or column index larger than the corresponding dimension of the matrix. For example, try this command.

```
>> A(5,5)=777
A =
    20    11    22    23     0
     5    11     0     0     0
     3    11     5    12     0
     0     0     0     0     0
     0     0     0     0   777
```

Note that Matlab happily assigns 777 to row 5, column 5, expanding the dimensions of the matrix and padding the missing entries with zeros.

```
>> size(A)
ans =
     5     5
```

The Transpose of a Matrix

You can take the transpose of a matrix in exactly the same way that you took the transpose of a row or column vector. For example, form a "magic" matrix with the following command.

```
>> A=magic(4)
A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

You can compute A^T with the following command.

```
>> A.'
ans =
    16     5     9     4
     2    11     7    14
     3    10     6    15
    13     8    12     1
```

Note that the first row of matrix A^T was previously the first column of matrix A. The second row of matrix A^T was previously the second column of matrix A, and so on for the third and fourth columns of matrix A^T. In essence, taking the transpose reflects the matrix A across its main diagonal (upper left corner to lower right corner), so the rows of A become columns of A^T and the columns of A become rows of A^T.

Building Matrices

Matlab has some powerful capabilities for building new matrices out of one or more matrices and/or vectors. For example, start by building a 2×3 matrix of ones.

```
>> A=ones(2,3)
A =
     1     1     1
     1     1     1
```

Now, build a new matrix with A as the first column and A as the second column. As we are not starting a new row, we can use either space or commas to delimit the row entries.

```
>> C=[A A]
C =
     1     1     1     1     1     1
     1     1     1     1     1     1
```

On the other hand, suppose that we want to build a new matrix with A as the first row and A as the second row. To start a new row we must end the first row with a semicolon.

```
>> C=[A; A]
C =
     1     1     1
     1     1     1
     1     1     1
     1     1     1
```

Let's create a 2×3 matrix of all zeros.

```
>> D=zeros(2,3)
D =
     0     0     0
     0     0     0
```

Now, let's build a matrix out of the matrices A and D.

```
>> E=[A D;D A]
E =
     1     1     1     0     0     0
     1     1     1     0     0     0
     0     0     0     1     1     1
     0     0     0     1     1     1
```

The possibilities are endless, with one caveat. The dimensions must be correct or Matlab will report an error. For example, create a 2×2 matrix of ones.

```
>> A=ones(2,2)
A =
     1     1
     1     1
```

And a 2×3 matrix of zeros.

```
>> B=zeros(2,3)
B =
     0     0     0
     0     0     0
```

It's possible to build a new matrix with A and B as row elements.

```
>> C=[A B]
C =
     1     1     0     0     0
     1     1     0     0     0
```

But it's not possible to build a new matrix with A and B as column elements.

```
>> C=[A;B]
??? Error using ==> vertcat
CAT arguments dimensions are not consistent.
```

This happens because A has 2 columns, but B has 3 columns, so the columns don't line up.

We'll see in later work that the matrix building capabilities of Matlab are a powerful ally.

Scalar-Matrix Multiplication

If asked to multiply a matrix by a scalar, one would hope that the operation of scalar-matrix multiplication would be carried out in exactly the same manner as scalar-vector multiplication. That is, simply multiply each entry of the matrix by the scalar.

▶ **Example 1.** *If A is the matrix*

$$A = 3\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix},$$

perform the scalar-matrix multiplication $3A$.

Simply multiply 3 times each entry of the matrix.

$$3A = 3\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 3 & 6 & 9 \\ 12 & 15 & 18 \\ 21 & 24 & 27 \end{bmatrix}$$

Matlab understands scalar-matrix multiplication. First, enter matrix A.

```
>> A=[1 2 3;4 5 6;7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
```

Now compute $3A$.

```
>> 3*A
ans =
     3     6     9
    12    15    18
    21    24    27
```

Matrix Addition

If two matrices have the same dimension, then add the matrices by adding the corresponding entries in each matrix.

▶ **Example 2.** *If A and B are the matrices*

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix},$$

find the sum $A + B$.

Simply add the corresponding entries.

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

Matlab understands matrix addition.

```
>> A=[1 1 1;2 2 2;3 3 3]; B=[1 1 1;1 1 1;1 1 1];
>> A+B
ans =
     2     2     2
     3     3     3
     4     4     4
```

This is identical to the hand-calculated sum above.

Let's look what happens when the dimensions are not the same.

▶ **Example 3.** *If A and B are the matrices*

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} \qquad \textit{and} \qquad B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix},$$

then find the sum $A + B$.

Note the dimensions of each matrix.

```
>> A=[1 1 1;2 2 2;3 3 3]; B=[1 1 1;1 1 1];
>> size(A)
ans =
     3     3
>> size(B)
ans =
     2     3
```

The matrices A and B do not have the same dimensions. Therfore, it is not possible to sum the two matrices.

```
>> A+B
??? Error using ==> plus
Matrix dimensions must agree.
```

This error message is completely expected.

One final example is in order.

▶ **Example 4.** *If matrix* A *is*

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

compute $A + 1$.

Note that this addition of a matrix and a scalar makes no sense.

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

The dimensions are all wrong. However, this is such a common occurrence in algebraic calculations (as we will see throughout the course), Matlab allows this matrix-scalar addition.

```
>> A=[1 1 1;2 2 2;3 3 3];
>> A+1
ans =
     2     2     2
     3     3     3
     4     4     4
```

Matlab simply adds 1 to each entry of the matrix A. That is, Matlab interprets $A+1$ as if it were the matrix addition of **Example 2**.

Matrix addition enjoys several properties, which we will ask you to explore in the exercises.

1. Addition is *commutative*. That is, $A+B=B+A$ for all matrices A and B having the same dimension.

2. Addition is *associative*. That is, $(A+B)+C=A+(B+C)$, for all matrices A, B, and C having the same dimension.

3. The zero matrix is the *additive identity*. That is, if A is $m \times n$ and 0 is an $m \times n$ matrix of all zeros, then $A+0=A$.

4. Each matrix A has an *additive inverse*. Form the matrix $-A$ by negating each entry of the matrix A. Then, $A+(-A)=0$.

Matrix-Vector Multiplication

Consider the linear system of three equations in three unknowns.

$$\begin{aligned} 2x+3y+4z &= 6 \\ 3x+2y+4z &= 8 \\ 5x-3y+8x &= 1. \end{aligned} \tag{2.7}$$

Because each of the corresponding entries are equal, the following 3×1 vectors are also equal.

$$\begin{bmatrix} 2x+3y+4z \\ 3x+2y+4z \\ 5x-3y+8x \end{bmatrix} = \begin{bmatrix} 6 \\ 8 \\ 1 \end{bmatrix}$$

The left-hand vector can be written as a vector sum.

$$\begin{bmatrix} 2x \\ 3x \\ 5x \end{bmatrix} + \begin{bmatrix} 3y \\ 2y \\ -3y \end{bmatrix} + \begin{bmatrix} 4z \\ 4z \\ 8z \end{bmatrix} = \begin{bmatrix} 6 \\ 8 \\ 1 \end{bmatrix}$$

Scalar multiplication can be used to factor the variable out of each vector on the left-hand side.

$$x \begin{bmatrix} 2 \\ 3 \\ 5 \end{bmatrix} + y \begin{bmatrix} 3 \\ 2 \\ -3 \end{bmatrix} + z \begin{bmatrix} 4 \\ 4 \\ 8 \end{bmatrix} = \begin{bmatrix} 6 \\ 8 \\ 1 \end{bmatrix} \tag{2.8}$$

The construct on the left-hand side of this result is so important that we will pause to make a definition.

Definition 5. *Let α_1, α_2, ..., and α_n be scalars and let $\mathbf{v}_1$, $\mathbf{v}_2$, ..., and $\mathbf{v}_n$ be vectors. Then the construction*

$$\alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2 + \cdots + \alpha_n\mathbf{v}_n$$

is called a **linear combination** *of the vectors $\mathbf{v}_1$, $\mathbf{v}_2$, ..., and $\mathbf{v}_n$. The scalars α_1, α_2, ..., and α_n are called the* **weights** *of the linear combination.*

For example, we say that

$$x \begin{bmatrix} 2 \\ 3 \\ 5 \end{bmatrix} + y \begin{bmatrix} 3 \\ 2 \\ -3 \end{bmatrix} + z \begin{bmatrix} 4 \\ 4 \\ 8 \end{bmatrix}$$

is a linear combination of the vectors $[2, 3, 5]^T$, $[3, 2, -3]^T$, and $[4, 4, 8]^T$.[10]

Finally, we take one last additional step and write the system (2.8) in the form

$$\begin{bmatrix} 2 & 3 & 4 \\ 3 & 2 & 4 \\ 5 & -3 & 8 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 6 \\ 8 \\ 1 \end{bmatrix}. \tag{2.9}$$

Note that the system (2.9) has the form

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

where

[10] Here we use the transpose operator to save a bit of space in the document.

$$A = \begin{bmatrix} 2 & 3 & 4 \\ 3 & 2 & 4 \\ 5 & -3 & 8 \end{bmatrix}, \qquad \mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \qquad \text{and} \qquad \mathbf{b} = \begin{bmatrix} 6 \\ 8 \\ 1 \end{bmatrix}.$$

The matrix A in (2.9) is called the *coefficient matrix*. If you compare the coefficient matrix in (2.9) with the original system (2.7), you see that the entries of the coefficient matrix are simply the coefficients of x, y, and z in (2.7). On right-hand side of system (2.9), the vector $\mathbf{b} = [6, 8, 1]^T$ contains the numbers on the right-hand side of the original system (2.7). Thus, it is a simple matter to transform a system of equations into a matrix equation.

However, it is even more important to compare the left-hand sides of system (2.8) and system (2.9), noting that

$$\begin{bmatrix} 2 & 3 & 4 \\ 3 & 2 & 4 \\ 5 & -3 & 8 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = x \begin{bmatrix} 2 \\ 3 \\ 5 \end{bmatrix} + y \begin{bmatrix} 3 \\ 2 \\ -3 \end{bmatrix} + z \begin{bmatrix} 4 \\ 4 \\ 8 \end{bmatrix}.$$

This tells us how to multiply a matrix and a vector. One takes a linear combination of the columns of the matrix, using the entries in the vector as weights for the linear combination.

Let's look at an example of matrix-vector multiplication

▶ **Example 6.** *Multiply the matrix and vector*

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

To perform the multiplication, take a linear combination of the columns of the matrix, using the entries in the vector as weights.

$$\begin{aligned} \begin{bmatrix} 1 & 2 & -3 \\ 3 & 0 & 4 \\ 0 & -2 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix} &= 1 \begin{bmatrix} 1 \\ 3 \\ 0 \end{bmatrix} - 2 \begin{bmatrix} 2 \\ 0 \\ -2 \end{bmatrix} + 3 \begin{bmatrix} -3 \\ 4 \\ 2 \end{bmatrix} \\ &= \begin{bmatrix} 1 \\ 3 \\ 0 \end{bmatrix} + \begin{bmatrix} -4 \\ 0 \\ 4 \end{bmatrix} + \begin{bmatrix} -9 \\ 12 \\ 6 \end{bmatrix} \\ &= \begin{bmatrix} -12 \\ 15 \\ 10 \end{bmatrix} \end{aligned}$$

It's important to note that this answer has the same number of entries as does each column of the matrix.

Let's see if Matlab understands this form of matrix-vector multiplication. First, load the matrix and the vector.

```
>> A=[1 2 -3;3 0 4;0 -2 2]; x=[1; -2; 3];
```

Now perform the multiplication.

```
>> A*x
ans =
   -12
    15
    10
```

Note this is identical to our hand calculated result.

Let's look at another example.

▶ **Example 7.** *Multiply* $A\mathbf{x}$, *where*

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 0 & -2 \end{bmatrix} \quad \textit{and} \quad \mathbf{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

If you try to perform the matrix-vector by taking a linear combination using the entries of the vectors as weights,

$$A\mathbf{x} = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 0 & -2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1 \begin{bmatrix} 1 \\ 2 \end{bmatrix} + 1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + ? \begin{bmatrix} 1 \\ -2 \end{bmatrix}. \tag{2.10}$$

The problem is clear. There are not enough weights in the vector to perform the linear combination.

Let's see if Matlab understands this "weighty" problem.

```
>> A=[1 1 1;2 0 -2]; x=[1; 1];
>> A*x
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

Inner dimensions? Let's see if we can intuit what that means. In our example, matrix A has dimensions 2×3 and vector x has dimensions 2×1. If we juxtapose these dimensions in the form $(2 \times 3)(2 \times 1)$, then the inner dimensions don't match.

Dimension Requirement. If matrix A has dimensions $m \times n$ and vector $\mathbf{x}$ has dimensions $n \times 1$, then we say "the innner dimensions match," and the matrix-vector product $A\mathbf{x}$ is possible. In words, the number of columns of matrix A must equal the number of rows of vector $\mathbf{x}$.

Matrix-Matrix Multiplication

We would like to extend our definition of matrix-vector multiplication in order to find the product of matrices. Here is the needed definition.

Definition 8. *Let A and B be matrices and let $\mathbf{b}_1$, $\mathbf{b}_2$, ..., and $\mathbf{b}_n$ represent the columns of matrix B. Then,*

$$AB = A\big[\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\big] = \big[A\mathbf{b}_1, A\mathbf{b}_2, \ldots, A\mathbf{b}_n\big].$$

Thus, to take the product of matrices A and B, simply multiply matrix A times each vector column of matrix B. Let's look at an example.

▶ **Example 9.** *Multiply*

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

We multiply the first matrix times each column of the second matrix, then use linear combinations to perform the matrix-vector multiplications.

$$\begin{aligned} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & -2 \\ 2 & 1 \end{bmatrix} &= \left[\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} -2 \\ 1 \end{bmatrix} \right] \\ &= \left[1 \begin{bmatrix} 1 \\ 3 \end{bmatrix} + 2 \begin{bmatrix} 2 \\ 4 \end{bmatrix}, \quad -2 \begin{bmatrix} 1 \\ 3 \end{bmatrix} + 1 \begin{bmatrix} 2 \\ 4 \end{bmatrix} \right] \\ &= \begin{bmatrix} 5 & 0 \\ 11 & -2 \end{bmatrix} \end{aligned}$$

Let's see if Matlab understands this form of matrix-matrix multiplication. First, load the matrices A and B.

```
>> A=[1 2;3 4]; B=[1 -2;2 1];
```

Now, multiply.

```
>> A*B
ans =
     5     0
    11    -2
```

Note that this result is indentical to our hand calculation.

Again, the inner dimensions must match or the matrix-matrix multiplication is not possible. Let's look at an example where things go wrong.

▶ **Example 10.** *Multiply*

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

When we multiply the first matrix times each column of the second matrix, we immediately see difficulty with the dimensions.

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 0 & -2 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \left[\begin{bmatrix} 1 & 1 & 1 \\ 2 & 0 & -2 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \quad \begin{bmatrix} 1 & 1 & 1 \\ 2 & 0 & -2 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \end{bmatrix} \right] \tag{2.11}$$

In the first column of the matrix product, the matrix-vector multiplication is not possible. The number of columns of the matrix does not match the number of entries in the vector. Therefore, it is not possible to form the product of these two matrices.

Let's see if Matlab understands this dimension difficulty.

```
>> A=[1 1 1;2 0 -2]; B=[1 2;3 4];
>> A*B
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

The error message is precisely the one we would expect.

Dimension Requirement. If matrix A has dimensions $m \times n$ and matrix B has dimensions $n \times p$, then we say "the inner dimensions match," and the matrix-matrix product AB is possible. In words, the number of columns of matrix A must equal the number of rows of matrix B.

Let's look at another example.

▶ **Example 11.** *Multiply*

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

Load the matrices A nd B into Matlab and check their dimensions.

```
>> A=[1 1 1;2 0 -2]; B=[1 2;1 -2;2 0];
>> size(A)
ans =
     2     3
>> size(B)
ans =
     3     2
```

Thus, matrix A has dimensions 2×3 and B has dimensions 3×2. Therefore, the inner dimensions match (they both equal 3) and it is possible to form the product of A and B.

```
>> C=A*B
C =
     4     0
    -2     4
```

Note the dimensions of the answer.

```
>> size(C)
ans =
     2     2
```

Recall that A was 2×3 and B was 3×2. Note that the "outer dimensions" are 2×2, which give the dimensions of the product.

Dimensions of the Product. If matrix A is $m \times n$ and matrix B is $n \times p$, then the dimensions of AB will be $m \times p$. We say that the "outer dimensions give the dimension of the product."

Properties of Matrix Multiplication

Matrix multiplication is *associative*. That is, for any matrices A, B, and C, providing the dimensions are right,

$$(AB)C = A(BC).$$

Let's look at an example.

▶ **Example 12.** *Given*

$$A = \begin{bmatrix} 2 & 2 \\ 3 & 3 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 1 \\ 2 & 5 \end{bmatrix}, \quad \textit{and} \quad C = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix},$$

use Matlab to demonstrate that $(AB)C = A(BC)$.

Load the matrices A, B, and C into Matlab, then calculate the left-hand side of $(AB)C = A(BC)$.

```
>> A=[2 2;3 3]; B=[1 1;2 5]; C=[3 3;2 5];
>> (A*B)*C
ans =
    42    78
    63   117
```

Next, calculate the right-hand side of $(AB)C = A(BC)$.

```
>> A*(B*C)
ans =
    42    78
    63   117
```

Hence, $(AB)C = A(BC)$.

Matrix Multiplication is Associative. In general, if A, B, and C have dimensions so that the multiplications are possible, matrix multiplication is associative. That is, it is always the case that

$$(AB)C = A(BC).$$

Unfortunately, matrix multiplication is **not** *commutative*. That is, even if A and B are of correct dimensions, it is possible that $AB \neq BA$. Let's look at an example.

▶ **Example 13.** *Let*

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 3 & 5 \\ 2 & 7 \end{bmatrix}.$$

Do the matrices A and B commute? That is, does $AB = BA$?

Load the matrices into Matlab, then compute AB.

```
>> A=[1 2;3 4]; B=[3 5;2 7];
>> A*B
ans =
     7    19
    17    43
```

Now compute BA.

```
>> B*A
ans =
    18    26
    23    32
```

Thus, $AB \neq BA$.

Matrix Multiplication is not Commutative. In general, even if the dimensions of A and B allow us to reverse the order of multiplication, matrices A and B will not commute. That is,

$$AB \neq BA.$$

Any change in the order of multiplication of matrices will probably change the answer.

Some matrices do commute, making this even more complicated.

```
>> A=[5 3;7 4],B=[-4 3;7 -5];
>> A*B
ans =
     1     0
     0     1
>> B*A
ans =
     1     0
     0     1
```

In this case, $AB = BA$.

However, in general, matrix multiplication is not commutative. The loss of the commutative property is not to be taken lightly. Any time you change the order of multiplication, you are risking an incorrect answer. There are many insidious ways that changes of order can creep into our calculations. For example, if you multiply the left-hand side of equation on the left by a matrix A, then multiply the right-hand side of the equation on the right by the same matrix A, you've changed the order and should expect an incorrect answer. We will explore how the loss of the commutative property can adversely affect other familiar algebraic properties in the exercises.

Here is a list of matrix properties you can depend on working all of the time. Let A and B be matrices of the correct dimension so that the additions and multiplications that follow are possible. Let α and β be scalars.

$A(B+C) = AB + AC$ $\qquad$ $\alpha(A+B) = \alpha A + \alpha B.$
$(A+B)C = AC + BC.$ $\qquad$ $\alpha(\beta A) = (\alpha\beta)A.$
$(\alpha+\beta)A = \alpha A + \beta A$ $\qquad$ $(\alpha A)B = \alpha(AB) = A(\alpha B).$

For example, as stated above, matrix multiplication is distributive over addition. That is, $A(B+C) = AB + AC$.

```
>> A=[2 3;-1 4]; B=[1 2;0 9]; C=[-3 2;4 4];
>> A*(B+C)
ans =
     8    47
    18    48
>> A*B+A*C
ans =
     8    47
    18    48
```

We will explore the remaining properties in the exercises.

2.2 Exercises

1. Given the matrices

$$A = \begin{bmatrix} 3 & 3 \\ 2 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 1 \\ 2 & 3 \end{bmatrix},$$

and

$$C = \begin{bmatrix} 3 & 1 \\ 5 & 8 \end{bmatrix},$$

use Matlab to verify each of the following properties. Note that 0 represents the zero matrix.

a) $A + B = B + A$

b) $(A + B) + C = A + (B + C)$

c) $A + 0 = A$

d) $A + (-A) = 0$

2. The fact that matrix multiplication is not commutative is a **huge** loss. For example, with real numbers, the following familiar algeraic properties hold.

i. $(ab)^2 = a^2b^2$
ii. $(a + b)^2 = a^2 + 2ab + b^2$
iii. $(a + b)(a - b) = a^2 - b^2$

Use Matlab and the matrices

$$A = \begin{bmatrix} 1 & 1 \\ 4 & 2 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 2 & 3 \\ 1 & 6 \end{bmatrix}$$

to show that none of these properties is valid for these choices of A and B. Can you explain why each of properties (i-iii) is not valid for matrix multiplication? *Hint: Try to expand the left-hand side of each property to arrive at the right-hand side.*

3. Given the matrices

$$A = \begin{bmatrix} 1 & 0 \\ 2 & 5 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 1 \\ 2 & 7 \end{bmatrix},$$

and

$$C = \begin{bmatrix} 1 & 2 \\ 0 & 9 \end{bmatrix},$$

use Matlab to verify each of the following forms of the distributive property.

a) $A(B + C) = AB + AC$

b) $(A + B)C = AC + BC$

4. Given the matrices

$$A = \begin{bmatrix} 2 & 2 \\ 4 & 7 \end{bmatrix}, \quad B = \begin{bmatrix} 3 & 1 \\ 8 & 9 \end{bmatrix},$$

and the scalars $\alpha = 2$ and $\beta = -3$, use Matlab to verify each of the following properties.

a) $(\alpha + \beta)A = \alpha A + \beta A$

b) $\alpha(A + B) = \alpha A + \alpha B$

c) $\alpha(\beta A) = (\alpha\beta)A$

d) $(\alpha A)B = \alpha(AB) = A(\alpha B)$

5. Enter the matrices **A=pascal(3)** and **B=magic(3)**.

a) Use Matlab to compute $(A+B)^T$.

b) Use Matlab to compute $A^T + B^T$ and compare your result with the result from part (a). Explain what your learned in this exercise.

6. Enter the matrix **A=pascal(4)** and the scalar $\alpha = 5$.

a) Use Matlab to compute $(\alpha A)^T$.

b) Use Matlab to compute αA and compare your result with the result from part (a). Explain what your learned in this exercise.

7. Using hand calculations only, calculate the following matrix-vector product, then verify your result in Matlab.

$$\begin{bmatrix} 1 & 1 & 2 \\ 3 & 4 & 0 \\ 0 & 5 & 6 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ -5 \end{bmatrix}$$

8. Write the following system of linear equations in matrix-vector form.

$$\begin{aligned} 2x + 2y + 3z &= -3 \\ 4x + 2y - 8z &= 12 \\ 3x + 2y + 5z &= 10 \end{aligned}$$

9. Using hand calculations only, calculate the following matrix-matrix product, then verify your result in Matlab.

$$\begin{bmatrix} 2 & 3 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 5 \end{bmatrix} \begin{bmatrix} 1 & 1 & 4 \\ 0 & 0 & 5 \\ 3 & 5 & 2 \end{bmatrix}$$

10. Enter the matrix **magic(8)**. What Matlab command will zero out all of the even rows? Use Matlab to verify your conjecture.

11. Enter the matrix **pascal(8)**. What Matlab command will zero out all of the odd columns? Use Matlab to verify your conjecture.

12. Enter the matrix **A=pascal(4)**.

a) What is the result of the Matlab command **A(:,2)=[ ]**? Note: **[ ]** is the empty matrix.

b) Refresh matrix A with **A=pascal(4)**. What is the result of the Matlab command **A(3,:)=[ ]**?

13. Enter the matrix **A=pascal(5)**.

a) What command will add a row of all ones to the bottom of matrix A? Use Matlab to verify your conjecture.

b) What command will add a column of all ones to the right end of matrix A? Use Matlab to verify your conjecture.

14. Enter the matrix **A=magic(3)**. Execute the command **A(5,4)=0**. Explain the resulting matrix.

15. Enter the matrix **A=ones(5)**.

a) Explain how you can insert a row of all 5's betwen rows 2 and 3 of matrix A. Use Matlab to verify your conjecure.

b) Explain how you can insert a column of all 5's betwen columns 3 and 4 of matrix A. Use Matlab to verify your conjecure.

16. Enter the matrix

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}.$$

a) What is the output of the Matlab command **A=A([1,3,2],:)**?

b) Refresh matrix A to its original value. What Matlab command will swap columns 1 and 3 of matrix A? Use Matlab to verify your conjecture.

17. Enter the matrix

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}.$$

a) Enter the Matlab command **A(2,:)=A(2,:)-4*A(1,:)**? Explain the result of this command.

b) Continue with the resulting matrix A from part (a). What is the output of the Matlab command **A(3,:)=A(3,:)-7*A(1,:)**? Explain the result of this command.

18. Type **format rat** to change the display to rational format. Create a 3×3 Hilbert matrix with the command **H=hilb(3)**.

a) What is the output of the Matlab command **H(1,:)=6*H(:,1)**? Explain the result of this command.

b) Continue with the resulting matrix H from part (a). What command will clear the fractions from row 2 of this result?

19. Enter the matrices **A=magic(3)** and **B=pascal(3)**. Execute the command **C=A+i*B**. *Note: You may have to enter* **clear i** *to return i to its default (the square root of -1).*

a) What is the transpose of the matrix C? Use Matlab to verify your response.

b) What is the conjugate transpose of the matrix C? Use Matlab to verify your response.

20. Use Matlab's **hadamard(n)** command to form Hadarmard matrices of order $n = 2$, 4, 8, and 16. In each case, use Matlab to calculate $H^T H$. Note the pattern. Explain in your own words what would happen if your formed the matrix product $H^T H$, where H is a Hadamard matrix of order 256.

21. Enter the Matlab command **magic(n)** to form a "magic" matrix of order $n = 8$. Use Matlab's **sum** command to sum both the columns and the rows of your "magic" matrix. Type **help sum** to learn how to use the syntax **sum(X,dim)** to accomplish this goal. What is "magic" about this matrix?

22. Enter the Matlab command **A=magic(n)** to form a "magic" matrix of order $n = 8$. Use Matlab's **sum** command to sum the columns of your "magic" matrix. Explain how you can use matrix-vector multilication to sum the columns of matrix A.

23. Set **A=pascal(5)** and then set **I=eye(5)**, then find the matrix product AI. Why is I called the *identity matrix*? Describe what a 256×256 identity matrix would look like.

24. Set **A=pascal(4)** and then set **B=magic(4)**. What operation will produce the second column of the matrix product AB? Can this be done

without finding the product AB?

25. Set the vector **v=(1:5).'** and the vector **w=(2:6).'**.

a) The product $\mathbf{v}^T\mathbf{w}$ is called an *inner product* because of the position of the transpose operator. Use Matalb to compute the inner product of the vectors $\mathbf{v}$ and $\mathbf{w}$.

b) The product $\mathbf{v}\mathbf{w}^T$ is called an *outer product* because of the position of the transpose operator. Use Matalb to compute the outer product of the vectors $\mathbf{v}$ and $\mathbf{w}$.

26. Enter **A=[0.2 0.6;0.8 0.4]**. Calculate A^n for $n = 2$, 3, 4, 5, etc. Does this sequence of matrices converge? If so, to what approximate matrix do they converge?

27. Use Matlab **ones** command to create the matrices

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix},$$

and

$$\begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix}.$$

Craft a Matlab command that will build the block diagonal matrix

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

where the zeros in this matrix represent matrices of zeros of the appropriate size.

28. Enter the Matlab command **hankel(x)** to form a Hankel matrix H, where $\mathbf{x}$ is the vector $[1, 2, 3, 4]$. The help file for the **hankel** commands describes the Hankel matrix as a *symmetric matrix*. Take the transpose of H. Describe what is mean by a symmetric matrix.

29. A Hilbert matrix H is defined by $H(i, j) = 1/(i + j - 1)$, where i ranges from 1 to the number of rows and j ranges from 1 to the number of columns. Use this definition and hand calculations to find a Hilbert matrix of dimension 4×4. Use **format rat** and Matlab's **hilb** command to check your result.

30. The number of ways to choose k objects from a set of n objects is defined and calcualted with the formula

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

Define a Pascal matrix P with the formula

$$P(i, j) = \binom{i+j-2}{i-1},$$

where i ranges from 1 to the number of rows and j ranges from 1 to the number of columns. Use this definition and hand calculations to find a Pascal matrix of dimension 4×4. Use Matlab's **pascal** command to check your result.

2.2 Answers

1.

a) Enter the matrices.

```
>> A=[3 3;2 1]; B=[1 1;2 3];
```

Calculate $A + B$.

```
>> A+B
ans =
     4     4
     4     4
```

Calculate $B + A$.

```
>> B+A
ans =
     4     4
     4     4
```

b) Enter the matrices

```
>> A=[3 3;2 1]; B=[1 1;2 3];
>> C=[3 1;5 8];
```

Calculate $(A + B)C$.

```
>> (A+B)+C
ans =
     7     5
     9    12
```

Calculate $AC + BC$.

```
>> A+(B+C)
ans =
     7     5
     9    12
```

c) Enter the matrix A and the zero matrix.

```
>> A=[3 3;2 1]; O=zeros(2,2);
```

Calculate $A + 0$.

```
>> A+O
ans =
     3     3
     2     1
```

Calculate A.

```
>> A
A =
     3     3
     2     1
```

d) Enter the matrix A and the zero matrix.

```
\startMatlab
>> A=[3 3;2 1]; O=zeros(2,2);
```

Calculate $A + (-A)$.

```
>> A+(-A)
ans =
     0     0
     0     0
```

Calculate the zero matrix.

```
>> O
O =
     0     0
     0     0
```

3.

a) Enter the matrices A, B, and C.

```
>> A=[1 0;2 5]; B=[0 1;2 7];
>> C=[1 2;0 9];
```

Compare $A(B+C)$ and $AB+AC$.

```
>> A*(B+C)
ans =
     1     3
    12    86
>> A*B+A*C
ans =
     1     3
    12    86
```

b) Enter the matrices A, B, and C.

```
>> A=[1 0;2 5]; B=[0 1;2 7];
>> C=[1 2;0 9];
```

Compare $(A+B)C$ and $AC+BC$.

```
>> (A+B)*C
ans =
     1    11
     4   116
>> A*C+B*C
ans =
     1    11
     4   116
```

5.

a) Ener the matrices A and B.

```
>> A=pascal(3); B=magic(3);
```

Compute $(A+B)^T$.

```
>> (A+B).'
ans =
     9     4     5
     2     7    12
     7    10     8
```

b) Compute $A^T + B^T$.

```
>> A.'+B.'
ans =
     9     4     5
     2     7    12
     7    10     8
```

The transpose of the sum of two matrices is equal to the sum of the transposes of the two matrices.

7. Enter matrix A and vector $\mathbf{x}$.

```
>> A=[1 1 2;3 4 0;0 5 6];
>> x=[1 2 5].';
```

Calculate $A\mathbf{x}$.

```
>> A*x
ans =
    13
    11
    40
```

9. Enter matrices A and B.

```
>> A=[2 3 1;0 1 2;0 0 5];
>> B=[1 1 4;0 0 5;3 5 2];
```

Calculate AB.

```
>> A*B
ans =
     5     7    25
     6    10     9
    15    25    10
```

11. Enter matrix A.

```
>> A=pascal(8);
```

The following command will zero out all the odd columns.

```
>> A(:,1:2:end)=0;
```

13.

a) Enter the matrix A.

```
>> A=pascal(5)
```

To add a row of all ones to the bottom of the matrix, execute the following command.

```
>> A(6,:)=ones(5,1)
```

b) Enter the matrix A.

```
>> A=pascal(5)
```

To add a column of all ones to the right end of the matrix, execute the following command.

```
>> A(:,6)=ones(5,1)
```

15.

a) Enter the matrix A.

```
>> A=ones(5);
```

We'll build a new matrix using the first two roes of matrix A, then a row of 5's, then the last three rows of matrix A. Note that we separate new columns with commas.

```
>> B=[A(1:2,:);5*ones(1,5);
A(3:5,:)]
B =
   1    1    1    1    1
   1    1    1    1    1
   5    5    5    5    5
   1    1    1    1    1
   1    1    1    1    1
   1    1    1    1    1
```

b) Enter the matrix A.

```
>> A=ones(5);
```

We'll build a new matrix using the first 3 columns of matrix A, then a column of 5's, then the last two columns of matrix A. Note that we separate new rows with semicolons.

```
>> B=[A(:,1:3),5*ones(5,1),
A(:,4:5)]
B =
  1    1    1    5    1    1
  1    1    1    5    1    1
  1    1    1    5    1    1
  1    1    1    5    1    1
  1    1    1    5    1    1
```

17.

a) Enter the matrix A.

```
>> A=[1 2 3;4 5 6;7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
```

The next command will subtract 4 times row 1 from row 2.

```
>> A(2,:)=A(2,:)-4*A(1,:)
A =
     1     2     3
     0    -3    -6
     7     8     9
```

b) Continuing with the last value of matrix A, the next command will subtract 7 times row 1 from row 3.

```
>> A(3,:)=A(3,:)-7*A(1,:)
A =
     1     2     3
     0    -3    -6
     0    -6   -12
```

19.

a) Enter the matrices A and B and compute C.

```
>> A=magic(3); B=pascal(3);
>> C=A+i*B
```

The transpose of

$$C = \begin{bmatrix} 8+i & 1+i & 6+i \\ 3+1 & 5+2i & 7+3i \\ 4+i & 9+3i & 2+6i \end{bmatrix}$$

is

$$C^T = \begin{bmatrix} 8+i & 3+i & 4+i \\ 1+i & 5+2i & 9+3i \\ 6+i & 7+3i & 2+6i \end{bmatrix}.$$

This result is verified with the following Matlab command.

```
>> C.'
```

b) The conjugate transpose of

$$C = \begin{bmatrix} 8+i & 1+i & 6+i \\ 3+1 & 5+2i & 7+3i \\ 4+i & 9+3i & 2+6i \end{bmatrix}$$

is

$$C^T = \begin{bmatrix} 8-i & 3-i & 4-i \\ 1-i & 5-2i & 9-3i \\ 6-i & 7-3i & 2-6i \end{bmatrix}.$$

This result is verified with the following Matlab command.

```
>> C'
```

21. Enter matrix A.

```
>> A=magic(8)
```

You sum the rows along the first dimension with the following command. You'll note that the sum of each column is 260.

```
>> sum(A,1)
```

You sum the columns along the second dimension with the following command. You'll note that the sum of each row is 260.

```
>> sum(A,2)
ans =
   260
   260
   260
   260
   260
   260
   260
   260
```

23. Store A with the following command.

```
>> A=pascal(5)
A =
     1     1     1     1     1
     1     2     3     4     5
     1     3     6    10    15
     1     4    10    20    35
     1     5    15    35    70
```

You store I with the following command.

```
>> I=eye(5)
I =
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
```

Note that AI is identical to matrix A.

```
>> A*I
ans =
   1    1    1    1    1
   1    2    3    4    5
   1    3    6   10   15
   1    4   10   20   35
   1    5   15   35   70
```

A 256×256 identity matrix would have 1's on its main diagonal and zeros in all other entries.

25.

a) Store the vectors $\mathbf{v}$ and $\mathbf{w}$.

```
>> v=(1:5).'; w=(2:6).';
```

The inner product $\mathbf{v}^T\mathbf{w}$ is computed as follows.

```
>> v.'*w
ans =
    70
```

You should be able to compute $\mathbf{v}^T\mathbf{w}$ manually and get the same result.

b) The outer product $\mathbf{v}\mathbf{w}^T$ is computed as follows.

```
>> v*w.'
ans =
   2    3    4    5    6
   4    6    8   10   12
   6    9   12   15   18
   8   12   16   20   24
  10   15   20   25   30
```

You should be able to compute $\mathbf{v}\mathbf{w}^T$ manually and get the same result.

27. Load the matrices A and B.

```
>> A=ones(2); B=2*ones(3);
```

Load the matrix C.

```
>> C=3*ones(2);
```

You can construct the required matrix with the following command.

```
>> D=[A,zeros(2,3),zeros(2,2);
zeros(3,2), B, zeros(3,2);
zeros(2,2), zeros(2,3), C]
```

29. The entry in row 1 column 1 would be $H(1,1) = 1/(1+1-1) = 1$. The entry in row 1 column 2 would be $H(1,2) = 1/(1+2-1) = 1/2$. Continuing in this manner, we arrive at a 4×4 Hilbert matrix.

$$H = \begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1/3 & 1/4 & 1/5 \\ 1/3 & 1/4 & 1/5 & 1/6 \\ 1/4 & 1/5 & 1/6 & 1/7 \end{bmatrix}$$

This result can be verified by these commands.

```
>> format rat
>> H=hilb(4)
```

2.3 Inverses in Matlab

In this section we will discuss the inverse of a matrix and how it relates to solving systems of equations. Not all matrices have inverses and this leads seamlessly to the discussion of the determinant. Finally, Matlab has some powerful built-in routines for solving systems of linear equations and we will investigate these as well.

Let's begin with a discussion of the identity matrix.

The Identity Matrix

In this section we will restrict our attention to *square matrices*; i.e., matrices of dimension $n \times n$, i.e., matrices having an equal number of rows and columns. A square matrix having ones on its main diagonal and zeros in all other entries is called an *identity matrix*. For example, a 3×3 identity matrix is the matrix

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

To see why I is called the identity matrix, let

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}.$$

Note that A times the first column of I is

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = 1 \begin{bmatrix} 1 \\ 4 \\ 7 \end{bmatrix} + 0 \begin{bmatrix} 2 \\ 5 \\ 8 \end{bmatrix} + 0 \begin{bmatrix} 3 \\ 6 \\ 9 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \\ 7 \end{bmatrix}.$$

Multiplying matrix A times $[1, 0, 0]^T$ simply strips off the first column of matrix A. In similar fashion, it is not hard to show that multiplying matrix A times $[0, 1, 0]^T$ and $[0, 0, 1]^T$, the second and third columns of I, strips off the second and third columns of matrix A.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ 8 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \\ 9 \end{bmatrix}.$$

Hence,

---

[11] Copyrighted material. See: http://msenux.redwoods.edu/Math4Textbook/

[12] Copyrighted material. See: http://msenux.redwoods.edu/IntAlgText/

$$AI = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = A.$$

Thus, $AI = A$. We'll leave it to our readers to check that $IA = A$. This result should make it clear why matrix I is called the identity matrix. If you multiply any matrix by I and you get the identical matrix back.

You use Matlab's **eye** command to create an identity matrix.

```
>> I=eye(3)
I =
     1     0     0
     0     1     0
     0     0     1
```

Checking the result above is a simple matter of entering the matrix A and performing the multiplications AI.

```
>> A=[1 2 3;4 5 6;7 8 9];
>> A*I
ans =
     1     2     3
     4     5     6
     7     8     9
```

The matrix I commutes with any matrix A.

```
>> I*A
ans =
     1     2     3
     4     5     6
     7     8     9
```

Note that $AI = A$ and $IA = A$.

Identity Property. Let A be a square matrix; i.e., a matrix of dimension $n \times n$. Create a square matrix I of equal dimension $(n \times n)$ which has ones on the main diagonal and all other entries are zero. Then,

$$AI = IA = A$$

The matrix I is called the **identity matrix**.

The Inverse of a Matrix

In the real number system, the number 1 acts as the multiplicative identity. That is,

$$a \cdot 1 = 1 \cdot a = a.$$

Then, for any nonzero real number a, there exists another real number, denoted by a^{-1}, such that

$$a \cdot a^{-1} = a^{-1} \cdot a = 1.$$

The number a^{-1} is called the *multiplicative inverse* of the number a. For example, $5 \cdot (1/5) = 1$, so the multiplicative inverse of 5 is $1/5$. That is, $5^{-1} = 1/5$.

Zero, however, has no multiplicative inverse, because there is no number whose product with zero will equal 1.

The situation with square matrices is similar. We have already established that I is the muliplicative identity; that is,

$$AI = IA = A$$

for all square matrices A. The next question to ask is this: given a square matrix A, can we find another square matrix A^{-1}, such that

$$AA^{-1} = A^{-1}A = I.$$

The answer is "Sometimes."

To find out when a square matrix has an inverse, we must first introduce the concept of the *determinant*. Every square matrix has a unique number associated with it that is called the determinant of the matrix. Finding the determinant of a 2×2 matrix is simple.

Determinant of a 2×2 Matrix. Let A be a 2×2 matrix, then the determinant of A is given by the following formula.

$$\det(A) = \det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad - bc.$$

Let's look at an example.

▶ **Example 1.** *Compute the determinant of*

$$A = \begin{bmatrix} 3 & 3 \\ 5 & -8 \end{bmatrix}.$$

Using the formula,

$$\det\left(\begin{bmatrix} 3 & 3 \\ 5 & -8 \end{bmatrix}\right) = (3)(-8) - (5)(3) = -39.$$

Matlab calculates determinants with ease.

```
>> A=[3 3;5 -8]
A =
     3     3
     5    -8
>> det(A)
ans =
   -39
```

Finding the determinants of higher order square matrices is more difficult. We encourage you to take a good linear algebra class to find our how to find higher ordered determinants. However, in this class, we'll let Matlab do the job for us.

▶ **Example 2.** *Find the determinant of the matrix*

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

We simply load the matrix into Matlab, then use Matlab's determinant operator **det**.

```
>> A=[1 -2 2 0 -2;-1 -1 -1 -2 0;1 -2 0 0 0;-1 -1 1 0 -2;-1 1 0 1
0]
A =
     1    -2     2     0    -2
    -1    -1    -1    -2     0
     1    -2     0     0     0
    -1    -1     1     0    -2
    -1     1     0     1     0
>> det(A)
ans =
     4
```

There is a simple test to determine whether a square matrix has an inverse.

Determining if an Inverse Exists. If the determinant of a square matrix is nonzero, then the inverse of the matrix exists.

Let's put this to the test.

▶ **Example 3.** *Determine if the inverse of the matrix*

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

exists. If it exists, find the inverse of the matrix.

First, determine the determinant of the matrix A.

```
>> A=[2 -3 2;1 -4 1;0 6 -2];
>> det(A)
ans =
    10
```

The determinant of A is 10, hence nonzero. Therefore, the inverse of matrix A exists. It is easily found with Matlab's **inv** command.

```
>> B=inv(A)
B =
    0.2000    0.6000    0.5000
    0.2000   -0.4000   -0.0000
    0.6000   -1.2000   -0.5000
```

We can easily check that matrix B is the inverse of matrix A. First, note that $AB = I$.

```
>> A*B
ans =
     1     0     0
     0     1     0
     0     0     1
```

Second, note that $BA = I$.

```
>> B*A
ans =
    1.0000         0         0
         0    1.0000    0.0000
         0    0.0000    1.0000
```

There is a little bit of roundoff error present here, but still enough evidence to see that $BA = I$. Hence, B is the inverse of matrix A.

Let's look at another example.

▶ **Example 4.** *Determine if the inverse of the matrix*

$$A = \begin{bmatrix} 5 & 0 & 5 \\ -5 & 3 & -8 \\ 2 & 0 & 2 \end{bmatrix}$$

exists. If it exists, find the inverse of the matrix.

Load the matrix into Matlab and calculate its determinant.

```
>> A=[5 0 5;-5 3 -8;2 0 2];
>> det(A)
ans =
     0
```

The determinant is zero, therefore the matrix A has no inverse. Let's see what happens when we try to find the inverse with Matlab's **inv** command.

```
>> B=inv(A)
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 2.467162e-18.
B =
   1.0e+15 *
  -3.60287970189640                   0   9.00719925474099
   3.60287970189640   0.00000000000000  -9.00719925474099
   3.60287970189640                   0  -9.00719925474099
```

A singular matrix is one that has determinant zero, or equivalently, one that has no inverse. Due to roundoff error, Matlab cannot determine exactly if the matrix is singular, but it suspects that this is the case. A check reveals that $AB \neq I$, evidence that matrix B is not the inverse of matrix A.

```
>> A*B
ans =
     1     0     0
     0     1     0
     0     0     0
```

Singular Versus Nonsingular. If the determinant of a matrix is zero, then we say that the matrix is **singular**. A singular matrix does not have an inverse. If the determinant of a matrix is nonzero, then we say that the matrix is **nonsingular**. A nonsingular matrix is invertible; i.e., the nonsingular matrix has an inverse.

Solving Systems of Linear Equations

Consider the system of linear equations

$$\begin{aligned} 2x + y + 3z &= -2 \\ x - 2y + 5z &= -13 \\ 3x + 4y - 2z &= 15 \end{aligned} \tag{2.12}$$

We can place this system into matrix-vector form

$$\begin{bmatrix} 2 & 1 & 3 \\ 1 & -2 & 5 \\ 3 & 4 & -2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -2 \\ -13 \\ 15 \end{bmatrix}. \tag{2.13}$$

System (**2.13**) is now in the form

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

where

$$A = \begin{bmatrix} 2 & 1 & 3 \\ 1 & -2 & 5 \\ 3 & 4 & -2 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} -2 \\ -13 \\ 15 \end{bmatrix}.$$

Let's calculate the determinant of the coefficient matrix A.

```
>> A=[2 1 3;1 -2 5;3 4 -2];
>> det(A)
ans =
    15
```

Thus, $\det(A) = 15$. Because the determinant of the matrix is nonzero, matrix A is nonsingular and A^{-1} exists. We can multiply both sides of equation $A\mathbf{x} = \mathbf{b}$ *on the left* (remembering that matrix multiplication is not commutative) to obtain

$$A^{-1}(A\mathbf{x}) = A^{-1}\mathbf{b}.$$

Matrix multiplication is associative, so we can change the grouping on the left-hand side of this last result to obtain

$$(A^{-1}A)\mathbf{x} = A^{-1}\mathbf{b}.$$

But multiplying inverses together produces the identity matrix I. Therefore,

$$I\mathbf{x} = A^{-1}\mathbf{b}.$$

Finally, because I is the identity matrix, $I\mathbf{x} = \mathbf{x}$, and we obtain the solution to the system $A\mathbf{x} = \mathbf{b}$.

$$\mathbf{x} = A^{-1}\mathbf{b}$$

Using Matlab and the solution $\mathbf{x} = A^{-1}\mathbf{b}$, let's find the solution to the system (**2.12**. First, enter $\mathbf{b}$, the vector of the right-hand side of system (**2.12**) (or the equivalent system (**2.13**)).

```
>> b=[-2;-13;15]
b =
    -2
   -13
    15
```

Use Matlab to calculate the solution $\mathbf{x} = A^{-1}\mathbf{b}$.

```
>> x=inv(A)*b
x =
    1.0000
    2.0000
   -2.0000
```

Thus, the solution of system (**2.12**) is $[x, y, z]^T = [1, 2, -2]^T$. This solution can be checked manually by substituting $x = 1$, $y = 2$, and $z = -2$ into each equation of the original system (**2.12**).

$$\begin{aligned} 2(1) + (2) + 3(-2) &= -2 \\ (1) - 2(2) + 5(-2) &= -13 \\ 3(1) + 4(2) - 2(-2) &= 15 \end{aligned}$$

Note that the solution satisfies each equation of system (**2.12**). However, we can also use Matlab to check that the result $\mathbf{x}$ satisfies the system's matrix-vector form (**2.13**) with the following calculation.

```
>> A*x
ans =
   -2.0000
  -13.0000
   15.0000
```

Note that the result $A\mathbf{x}$ equals the vector $\mathbf{b}$ (to roundoff error) and thus is a solution of system (**2.13**).

Matlab's backslash operator \ (also used for left division) can be used to solve systems of the form $A\mathbf{x} = \mathbf{b}$. If we might be allowed some leeway, the following abuse in notation outlines the key idea. We first "left-divide" both sides of the equation $A\mathbf{x} = \mathbf{b}$ on the left by the matrix A.

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

On the left, $A\backslash A$ is again the identity, so this leads to the solution

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

Enter the following in Matlab.

```
>> x=A\b
x =
    1.0000
    2.0000
   -2.0000
```

Note that this matches the previous result that was found with the computation $\mathbf{x} = A^{-1}\mathbf{b}$.

One needs an introductory linear algebra course to fully understand the use of Matlab's backslash operator. For example, it's possible that a system has no solutions.

▶ **Example 5.** *Solve the system*

$$\begin{aligned} x - y &= 1 \\ -x + y - z &= 1 \\ 5x - 5y + 3z &= 1. \end{aligned}$$

This system can be written in matrix-vector form $A\mathbf{x} = \mathbf{b}$, where

$$A = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 1 & -1 \\ 5 & -5 & 3 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

Enter the matrix A in Matlab and check the value of its determinant.

```
>> A=[1 -1 0;-1 1 -1;5 -5 3];
>> det(A)
ans =
     0
```

Because the determinant is zero, matrix A is singular and has no inverse. Hence, we will not be able to solve this system using $\mathbf{x} = A^{-1}\mathbf{b}$.

Let's find out what will happen if we try to solve the system using Matlab's backslash operator. Enter the vector $\mathbf{b}$ in Matlab and execute **x=A\b**.

```
>> b=[1;1;1];
>> x=A\b
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 6.608470e-18.
x =
   1.0e+15 *
  -7.20575940379279
  -7.20575940379279
  -0.00000000000000
```

Note that Matlab thinks that the coefficient matrix is nearly singular, but cannot decide due to roundoff error. So the backslash operator attempts to find a solution but posts a pretty serious warning that the solution may not be accurate. Therefore, we should attempt to check. The result satisfies the equation if and only if $A\mathbf{x} = \mathbf{b}$.

```
>> A*x
ans =
     1
     1
     0
```

Note that the vector this computation returns is not the vector $\mathbf{b} = [1, 1, 1]^T$, so the solution found by Matlab's backslash operator is not a solution of $A\mathbf{x} = \mathbf{b}$.

It's also possible that a system could have an infinite number of solutions. For example, in a system of three equations in three unknowns, the three planes represented by the equations in the system could intersect in a line of solutions.

▶ **Example 6.** *Solve the system*

$$\begin{aligned} x - y &= 1 \\ -x + y - z &= 1 \\ 5x - 5y + 3z &= -1. \end{aligned}$$

This system can be written in matrix-vector form $A\mathbf{x} = \mathbf{b}$, where

$$A = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 1 & -1 \\ 5 & -5 & 3 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}.$$

Note that the coefficient matrix of this system is identical to that of the system in **Example 5**, but the vector $\mathbf{b}$ is slightly different. Because the determinant of the coefficient matrix is zero, matrix A is singular and has no inverse. Hence, we will not be able to solve this system using $\mathbf{x} = A^{-1}\mathbf{b}$.

What will happen if we try to solve the system using Matlab's backslash operator? Enter the vector $\mathbf{b}$ in Matlab and execute **x=A\b**.

```
>> x=A\b
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 6.608470e-18.
x =
    -1
    -2
    -2
```

Note that Matlab still thinks that the coefficient matrix is nearly singular, but cannot decide due to roundoff error. So the backslash operator attempts to find a solution but posts a pretty serious warning that the solution may not be accurate. Therefore, we should attempt to check. The result satisfies the equation if and only if $A\mathbf{x} = \mathbf{b}$.

```
>> A*x
ans =
     1
     1
    -1
```

The vector this computation returns does equal the vector $\mathbf{b} = [1, 1, -1]^T$, so the solution found by Matlab's backslash operator is a solution of $A\mathbf{x} = \mathbf{b}$.

However, this one calculation does not reveal the whole picture. Indeed, one can check that the vector $\mathbf{x} = [2, 1, -2]^T$ is also a solution.

```
>> x=[2;1;-2]; A*x
ans =
     1
     1
    -1
```

Note that $A\mathbf{x} = \mathbf{b}$, so $\mathbf{x} = [2, 1, -2]^T$ is a solution.

Indeed, one can use Matlab's *Symbolic Toolbox* to show that

$$\mathbf{x} = \begin{bmatrix} -1 \\ -2 \\ -2 \end{bmatrix} + \alpha \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \tag{2.14}$$

is a solution of the system for all real values of α. First, declare **alpha** to be a symbolic variable.

```
>> syms alpha
```

Enter $\mathbf{x}$, as defined in **equation (2.14)**.

```
>> x=[-1;-2;-2]+alpha*[1;1;0]
x =
 -1+alpha
 -2+alpha
       -2
```

Now, calculate $A\mathbf{x}$ and compare the answer to the vector $\mathbf{b}$.

```
>> A*x
ans =
     1
     1
    -1
```

Thus, the vector $\mathbf{x} = [-1+\alpha, -2+\alpha, -2]^T$ is a solution of the system for all real values of α. For example, by letting $\alpha = 0$, we produce the solution produced by Matlab's backslash operator, $\mathbf{x} = [-1, -2, -2]^T$. By varying α, you can produce all solutions of the system. As a final example, if $\alpha = 10$, then we get the solution $\mathbf{x} = [9, 8, -2]^T$. Readers should use Matlab to check that this is actually a solution.

Explaining why this works is beyond the scope of this course. If you want to enter the fascinating world of solving systems of equations, make sure that you take a good introductory course in linear algebra. However, here are the pertinent facts.

Solving Systems of Equations. If you have a system $A\mathbf{x} = \mathbf{b}$ of m equations in n unknowns, there are three possible solutions scenarios.

1. The system has exactly one unique solution.
2. The system has no solutions.
3. The system has an infinite number of solutions.

When working with square systems $A\mathbf{x} = \mathbf{b}$, that is, n equations in n unknowns, here are the facts.

1. If $\det(A) \neq 0$, then matrix A is nonsingular and A^{-1} exists. In this case, the system $A\mathbf{x} = \mathbf{b}$ has a unique solution $\mathbf{x} = A^{-1}\mathbf{b}$. You can also find this solution with Matlab's backslash operator. That is, perform the calculation **x=A\b**.
2. If $\det(A) = 0$, then matrix A is singular and A^{-1} does not exist. In this case, the system $A\mathbf{x} = \mathbf{b}$ either has no solutions or an infinite number of solutions. If you use Matlab's backslash operator to calculate **x=A\b**, be sure to check that your solution satisfies $A\mathbf{x} = \mathbf{b}$.

2.3 Exercises

1. For each of the following matrices, form the appropriate identity matrix with Matlab's **eye** command, then use Matlab to show that $AI = IA = A$.

a)

$$A = \begin{bmatrix} 1 & 2 \\ 4 & 8 \end{bmatrix}$$

b)

$$A = \begin{bmatrix} 1 & 0 & 2 \\ 3 & -2 & 4 \\ 0 & 0 & -5 \end{bmatrix}$$

c)

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

2. If you reorder the rows or columns of the identity matrix, you obtain what is known as a *permuation* matrix.

a) Form a 3×3 identity matrix with the Matlab command **I=eye(3)**. Swap the first and third columns of I with **P=I(:,[3,2,1])** to form a permutation matrix P. Now, enter

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}.$$

Multiply A on the right by P. Explain the result. Next, multiply A on the left by P and explain the result.

b) Form a 4×4 identity matrix with the Matlab command **I=eye(4)**. Reorder the rows of I with the command **P=I([4,2,1,3],:)** to form a permutation matrix P. Now, enter

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}.$$

Multiply A on the left by P. Explain the result. Next, multiply A on the right by P and explain the result.

3. Use hand calculations to determine the determinant of the matrix

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

Use Matlab to verify your result.

4. Calculate the determinant of each of the following matrices, then classify the given matrix as *singular* or *nonsingular*.

a)

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

b)

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

c)

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

5. If

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix},$$

then following formula shows how to find the inverse of a 2×2 matrix.

$$A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Use hand calculations and the formula to determine the inverse of the matrix

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

Use Matlab's **inv** command to verify your result. *Hint: You might want to use* **format rat** *to view your results.*

6. Use Matlab's **inv** command to find the inverse B of each of the following matrices. In each case, check your result by calculating AB and BA. Comment on each result. *Hint: Use the determinant to check if each matrix is singular or nonsingular.*

a)

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

b)

$$A = \begin{bmatrix} 1 & -5 & -1 \\ -5 & 1 & 5 \\ -4 & -2 & 4 \end{bmatrix}$$

c)

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

7. Set

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 1 & 1 \\ 3 & 5 \end{bmatrix}.$$

Calculate $(AB)^{-1}$, $A^{-1}B^{-1}$, and $B^{-1}A^{-1}$. Which two of these three expressions are equal?

8. Set

$$A = \begin{bmatrix} 2 & 3 \\ -1 & 5 \end{bmatrix}.$$

Use Matlab commands to determine if $(A^T)^{-1} = (A^{-1})^T$.

9. Set

$$A = \begin{bmatrix} 1 & 1 \\ 2 & -4 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 3 & -1 \\ 2 & 5 \end{bmatrix}.$$

Use Matlab commands to determine if $(A + B)^{-1} = A^{-1} + B^{-1}$.

In **Exercises 10-13**, place the given system in matrix form $A\mathbf{x} = \mathbf{b}$. Check the determinant of the coefficient matrix, then solve the system with $x = A^{-1}\mathbf{b}$. Check your result by showing that $A\mathbf{x} = \mathbf{b}$.

10.

$$\begin{aligned} 4y - 2z &= 1 \\ -x + y + z &= -1 \\ 4x - 2y &= 3 \end{aligned}$$

11.

$$\begin{aligned} x + z &= 3 \\ 4x + 2y + 2z &= 5 \\ 3x + 2y + 2z &= -1 \end{aligned}$$

12.

$$\begin{aligned} -w - 3x - y &= 2 \\ 3x + 2y - 2z &= 1 \\ 4w - 2y - 2z &= 10 \\ -2w + 2y &= 1 \end{aligned}$$

13.

$$\begin{aligned} w + 5x + 2y + z &= 4 \\ -4w - 2x - 2y + z &= 0 \\ -5w + x - 4y - z &= 3 \\ -4w - y + 2z &= 5 \end{aligned}$$

In **Exercises 14-20**, place the given system in matrix form $A\mathbf{x} = \mathbf{b}$. Check the determinant of the coefficient matrix, then solve the system with Matlab's backslash operator **x=A\b**. Check your result by showing that $A\mathbf{x} = \mathbf{b}$. Classify the system as having a unique solution, no solutions, or an infinite number of solutions.

14.

$$\begin{aligned} -x + y &= 3 \\ y - z &= 2 \\ x - y - 3z &= 4 \end{aligned}$$

15.

$$\begin{aligned} -5x - y + z &= 7 \\ x + 3y - 3z &= 0 \\ -5x + y - z &= 8 \end{aligned}$$

16.

$$\begin{aligned} 5x + 3y - 5z &= 11 \\ 2x - y - 2z &= 0 \\ -2x - y + 2z &= -5 \end{aligned}$$

17.

$$\begin{aligned} -2x - 2y - 2z &= 12 \\ x - 3y - z &= 4 \\ 4x + 3z &= 12 \end{aligned}$$

18.

$$\begin{aligned} -w + x &= 4 \\ w + x - 3z &= 12 \\ -3x - 2y + z &= 0 \\ -3w + 3x - 2y + 2z &= 5 \end{aligned}$$

19.

$$\begin{aligned} w + y - z &= 0 \\ 2w + x + y + z &= 1 \\ -2w - x - y - z &= -1 \\ 2w + 2x &= 0 \end{aligned}$$

20.

$$\begin{aligned} w + y - z &= 0 \\ 2w + x + y + z &= 2 \\ -2w - x - y - z &= -1 \\ 2w + 2x &= 0 \end{aligned}$$

2.3 Answers

1.

a) Enter matrix A and the identity for 2×2 matrices.

```
>> A=[1 2;4 8]; I=eye(2);
```

Note that AI and IA both equal A.

```
>> A*I
ans =
       1              2
       4              8
>> I*A
ans =
       1              2
       4              8
```

b) Enter matrix A and the identity for 3×3 matrices.

```
>> A=[1 2;4 8]; I=eye(2);
```

Note that AI equals A.

```
>> A*I
ans =
     1          0          2
     3         -2          4
     0          0         -5
```

Note that IA equals A.

```
>> I*A
ans =
     1          0          2
     3         -2          4
     0          0         -5
```

c) Enter matrix A and the identity for 4×4 matrices.

```
>> A=[1 2 0 0;1 -1 1 -1;
0 0 2 2;3 -1 -2 0];
>> I=eye(4);
```

Note that AI equals A.

```
>> A*I
ans =
  1       2       0       0
  1      -1       1      -1
  0       0       2       2
  3      -1      -2       0
```

Note that IA equals A.

```
>> I*A
ans =
  1       2       0       0
  1      -1       1      -1
  0       0       2       2
  3      -1      -2       0
```

3.

$$\begin{bmatrix} -3 & 1 \\ -2 & -4 \end{bmatrix} = (-3)(-4) - (-2)(1)$$
$$= 14$$

Enter the matrix in Matlab and calculate its determinant.

```
>> A=[-3 1;-2 -4];
>> det(A)
ans =
      14
```

5. If

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

then

$$A^{-1} = \frac{1}{18}\begin{bmatrix} 6 & -3 \\ 4 & 1 \end{bmatrix}$$
$$= \begin{bmatrix} 1/3 & -1/6 \\ 2/9 & 1/18 \end{bmatrix}.$$

Enter the matrix in Matlab, select rational format, then use Matlab's **inv** command to calculate the inverse.

```
>> A=[1 3;-4 6];
>> format rat
>> inv(A)
ans =
       1/3           -1/6
       2/9            1/18
```

7. Enter matrices A and B and calculate $(AB)^{-1}$.

```
>> format rat
>> A=[1 2;3 4]; B=[1 1;3 5];
>> inv(A*B)
ans =
     -23/4           11/4
      15/4           -7/4
```

Calculate $A^{-1}B^{-1}$.

```
>> inv(A)*inv(B)
ans =
     -13/2            3/2
       9/2           -1
```

Calculate $B^{-1}A^{-1}$.

```
>> inv(B)*inv(A)
ans =
     -23/4           11/4
      15/4           -7/4
```

Note that $(AB)^{-1} \neq A^{-1}B^{-1}$. However, $(AB)^{-1} = B^{-1}A^{-1}$.

9. Enter matrices A and B and calculate $(A+B)^{-1}$.

```
>> format rat
>> A=[1 1;2 -4];
>> B=[3 -1;2 5];
>> inv(A+B)
ans =
       1/4             0
      -1               1
```

Calculate $A^{-1} + B^{-1}$.

```
>> format rat
>> inv(A)+inv(B)
ans =
      49/51          23/102
      11/51           1/102
```

Therefore, $(+B)^{-1} \neq A^{-1} + B^{-1}$.

11. In matrix form,

$$\begin{bmatrix} 1 & 0 & 1 \\ 4 & 2 & 2 \\ 3 & 2 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ -1 \end{bmatrix}.$$

Enter the matrix A and check the determinant of A.

```
>> A=[1 0 1;4 2 2;3 2 2];
>> det(A)
ans =
     2
```

Hence, A is nonsingular and the system has a unique solution. Enter the vector $\mathbf{b}$, then compute the solution $\mathbf{x}$ with $x = A^{-1}\mathbf{b}$.

```
>> b=[3; 5; -1];
>> x=inv(A)*b
x =
    6.0000
   -6.5000
   -3.0000
```

13. In matrix form,

$$\begin{bmatrix} 1 & 5 & 2 & 1 \\ -4 & -2 & -2 & 1 \\ -5 & 1 & -4 & -1 \\ -4 & 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 3 \\ 5 \end{bmatrix}$$

Enter the matrix A and check the determinant of A.

```
>> A=[1 5 2 1;-4 -2 -2 1;
-5 1 -4 -1;-4 0 -1 2];
>> det(A)
ans =
```

Hence, A is nonsingular and the system has a unique solution. Enter the vector $\mathbf{b}$, then compute the solution $\mathbf{x}$ with $x = A^{-1}\mathbf{b}$.

```
>> b=[4;0;3;5];
>> x=inv(A)*b
x =
   60.5000
   10.5000
  -93.0000
   77.0000
```

15. In matrix form,

$$\begin{bmatrix} -5 & -1 & 1 \\ 1 & 3 & -3 \\ -5 & 1 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 7 \\ 0 \\ 8 \end{bmatrix}.$$

Enter the matrix A and check the determinant of A.

```
>> A=[-5 -1 1;1 3 -3;-5 1 -1];
>> det(A)
ans =
     0
```

Hence, A is singular and the system either has no solutions or an infinite number of solutions. Enter the vector $\mathbf{b}$, then compute the solution $\mathbf{x}$ with

x=A\b.

```
>> b=[7;0;8];
>> x=A\b
Warning: Matrix is close to
singular or badly scaled.
Results may be inaccurate.
RCOND = 3.280204e-18.
x =
   -1.5000
    1.9231
    1.4231
```

Let's see if the answer checks.

```
>> A*x
ans =
    7.0000
    0.0000
    8.0000
```

Thus, it would appear that it is not the case that the system has no solutions. Therefore, the system must have an infinite number of solutions.

17. In matrix form,

$$\begin{bmatrix} -2 & -2 & -2 \\ 1 & -3 & -1 \\ 4 & 0 & 3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 12 \\ 4 \\ 12 \end{bmatrix}.$$

Enter the matrix A and check the determinant of A.

```
>> A=[-2 -2 -2;1 -3 -1;4 0 3];
>> det(A)
ans =
     8
```

Hence, A is nonsingular and the system either a unique solution. Enter the vector **b**, then compute the solution **x** with **x=A\b**.

```
>> b=[12;4;12];
>> x=A\b
x =
  -16.5000
  -15.5000
   26.0000
```

Let's see if the answer checks.

```
>> A*x
ans =
    12
     4
    12
```

This checks and the system has a unique solution.

19. In matrix form,

$$\begin{bmatrix} 1 & 0 & 1 & -1 \\ 2 & 1 & 1 & 1 \\ -2 & -1 & -1 & -1 \\ 2 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ -1 \\ 0 \end{bmatrix}$$

Enter the matrix A and calculate the determinant.

```
>> A=[1 0 1 -1;2 1 1 1;
-2 -1 -1 -1;2 2 0 0];
>> det(A)
ans =
     0
```

Hence, A is singular and the system

either has no solutions or an infinite number of solutions. Enter the vector **b**, then compute the solution **x** with **x=A\b**.

```
>> b=[0;1;-1;0];
>> x=A\b
Warning: Matrix is singular
to working precision.
x =
       NaN
       NaN
       NaN
    0.5000
```

Unfortunately, the presence of **NaN** ("not a number") will prevent us from futher analysis. In a later chapter, we'll learn how to determine a solution with a technique called *Gaussian Elimination*.

2.4 Array Operations in Matlab

When we multiply a scalar with a vector or matrix, we simply multiply the scalar times each entry of the vector or matrix. Addition and subtraction behave in the same manner, that is, if we add or subtract two vectors or matrices, we obtain the answer by adding or subtracting the correponding entries.

Matrix multiplication, as we've seen, is quite different. When we multiply two matrices, for example, we don't simply multiply the corresponding entries. For example, enter the following matrices A and B.

```
>> A=[1 2;3 4], B=[2 5;-1 3]
A =
     1     2
     3     4
B =
     2     5
    -1     3
```

Compute the product of A and B.

```
>> A*B
ans =
     0    11
     2    27
```

Note that

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 2 & 5 \\ -1 & 3 \end{bmatrix} = \begin{bmatrix} 0 & 11 \\ 2 & 27 \end{bmatrix}.$$

Matrix multiplication is not performed by multiplying the corresponding entries of each matrix.

Array Multiplication. However, there are occasions where we need to "multiply" two matrices by finding the product of the corresponding entries, so that the resulting "product" looks as follows.

Warning 1. *This is not matrix multiplication. Rather, it is a special type of multiplication called* **array multiplication**.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 2 & 5 \\ -1 & 3 \end{bmatrix} = \begin{bmatrix} 2 & 10 \\ -3 & 12 \end{bmatrix}$$

Note that the array multiplication is performed by simply multiplying the corresponding entries of each matrix.

Matlab provides an operator that allows us to perform *array multiplication*, namely the .* operator, sometimes pronounced "dot-times." In the case of our previously entered matrices A and B, note how array multiplication performs.

```
>> A,B
A =
     1     2
     3     4
B =
     2     5
    -1     3
>> A.*B
ans =
     2    10
    -3    12
```

Thus, when you use .*, Matlab's array multiplication operator, the product of two matrices or vectors is found by multiplying the corresponding entries in each matrix or vector.

Here is an example of using array multiplication with row vectors.

```
>> v=[1 2 3], w=[4 5 6]
v =
     1     2     3
w =
     4     5     6
>> v.*w
ans =
     4    10    18
```

Again, note that the array product of the row vectors **v** and **w** was found by mulitplying the corresponding entries of each vector.

Array multiplication works equally well for column vectors.

```
>> v=v.',w=w.'
v =
     1
     2
     3
w =
     4
     5
     6
>> v.*w
ans =
     4
    10
    18
```

Array Division. We will also need the ability to "divide" matrices or vectors by finding the quotient of the corresponding entries. The Matlab operator for *array division* is `./`, pronounced "dot-divided by." It works in exactly the same way as does array multiplication.

Warning 2. *This is not matrix division. Rather, it is a special type of division called* **array division**.

$$\begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix} \Big/ \begin{bmatrix} 3 & 5 \\ 7 & 9 \end{bmatrix} = \begin{bmatrix} 1/3 & 3/5 \\ 5/7 & 7/9 \end{bmatrix}$$

Note that array division is performed by dividing each entry of the matrix on the left by the corresponding entry of the matrix on the right.

Matlab provides an operator that allows us to perform *array right division*, namely the `./` operator, sometimes pronounced "dot-divided by." It helps to first change the display to rational format.

```
>> format rat
```

Enter the matrices A and B that are used in **Warning 2**.

```
>> A=[1 3;5 7], B=[3 5;7 9]
A =
       1              3
       5              7
B =
       3              5
       7              9
```

Now, use array right division to obtain the result shown in **Warning 2**.

```
>> A./B
ans =
       1/3            3/5
       5/7            7/9
```

Matlab also has an array left division operator that is sometimes useful.

```
>> A.\B
ans =
       3              5/3
       7/5            9/7
```

Note that each entry of the result is found by dividing each entry of the matrix on the right by the corresponding entry of the matrix on the left. Hence, the term "left division."

It's often useful to divide a scalar by a matrix or vector. With array right division, the scalar is divided by each entry of the vector or matrix.

```
>> v=1:5
v =
       1              2              3              4
  5
>> 1./v
ans =
       1              1/2            1/3            1/4
  1/5
```

With array left division, each entry of the vector or matrix is divided by the scalar.

```
>> w=[4;5;6]
w =
       4
       5
       6
>> 7.\w
ans =
       4/7
       5/7
       6/7
```

Array Exponentiation. We will also need the ability to raise each entry of a vector or matrix to a power. The Matlab operator for *array exponentiation* is `.^`, pronounced "dot raised to." It works in exactly the same manner as does array multiplication and division.

Warning 3. *This is not the usual way to raise a matrix to a power. Rather, it is a special type of exponentiation called* **array exponentiation**.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^2 = \begin{bmatrix} 1 & 4 \\ 9 & 16 \end{bmatrix}$$

Note that array exponentiation is performed by raising each entry of the matrix to the power of 2.

We can return to default formatting with the following command.

```
>> format
```

Now, suppose that we wish to raise each element of a vector to the third power. We use `.^` for this task.

```
>> v=3:7
v =
       3              4              5              6
  7
>> v.^3
ans =
      27             64            125            216
343
```

Note that raising the vector $\mathbf{v}$ to the third power is not possible because the dimensions will not allow $\mathbf{v}^2 = \mathbf{vvv}$.

```
>> v^3
??? Error using ==> mpower
Matrix must be square.
```

You can square each element of a matrix with the array exponentiation operator.

```
>> A=[1 2;3 4]
A =
       1              2
       3              4
>> A.^2
ans =
       1              4
       9             16
```

Note that array exponentiation completely differs from regular exponentiation.

```
>> A^2
ans =
       7             10
      15             22
```

Matlab Functions are Array Smart

Most of Matlab's elementary functions are what we like to call "array smart," that is, when wyou feed a Matlab function a vector or matrix, then that function is applied to each element of the vector or matrix.

Consider, for example, the behavior of the sine function. You can take the sine of a single number.

```
>> sin(pi/2)
ans =
       1
```

However, a powerful feature is the fact that matlab's sine function can be applied to a vector or matrix.

```
>> x=0:pi/2:2*pi
x =
         0    1.5708    3.1416    4.7124    6.2832
>> sin(x)
ans =
         0    1.0000    0.0000   -1.0000   -0.0000
```

Note that Matlab took the sine of each element of the vector (with a little round off error). You will see similar behavior if you take the sine of a matrix.

```
>> A=[0 pi/2; pi 3*pi/2]
A =
         0    1.5708
    3.1416    4.7124
>> sin(A)
ans =
         0    1.0000
    0.0000   -1.0000
```

Matlab took the sine of each entry of matrix A (to a little roundoff error).

You can take the natural logarithm[14] of a number with Matlab's **log** function.

```
>> log(1)
ans =
     0
```

Matlab's **log** function is array smart.

```
>> x=(1:5).'
x =
     1
     2
     3
     4
     5
>> log(x)
ans =
         0
    0.6931
    1.0986
    1.3863
    1.6094
```

Again, Matlab took the natural logarithm of each element of the vector. This is typical of the way most Matlab functions work.

You can access a list of Matlab's elementary functions with the command **help elfun**.

Basic Plotting

Because Matlab's elementary array functions are "array smart," it is a simple matter to obtain a plot of most elementary functions using Matlab's **plot** command.

In its simplest form, Matlab's **plot** command takes two vectors **x** and **y** which contain the x- and y-values of a collection of points (x, y) to be plotted. In its

[14] Students of mathematics are confused when they find that the Matlab command **log** is used to compute the natural logarithm of a number. Indeed, mathematicians usualy write $\ln x$ to denote the natural logarithm and log to denote the base ten logarithm. In Matlab **log** is use to compute the natural logarithm, while the command **log10** is used to compute the base ten logarithm.

default **plot(x,y)** form, Matlab's **plot** command plots the points in the order received and connects consecutive points with line segments.

The following commands store the numbers 0, $\pi/2$, π, $3\pi/2$ and 2π in the vector **x**, then evaluate the sine at each entry of the vector **x**, storing the results in the vector **y**.

```
>> x=0:pi/2:2*pi
x =
         0    1.5708    3.1416    4.7124    6.2832
>> y=sin(x)
y =
         0    1.0000    0.0000   -1.0000   -0.0000
```

The command **plot(x,y)** is used to produce the graph in **Figure 2.6**(a).

```
>> plot(x,y)
```

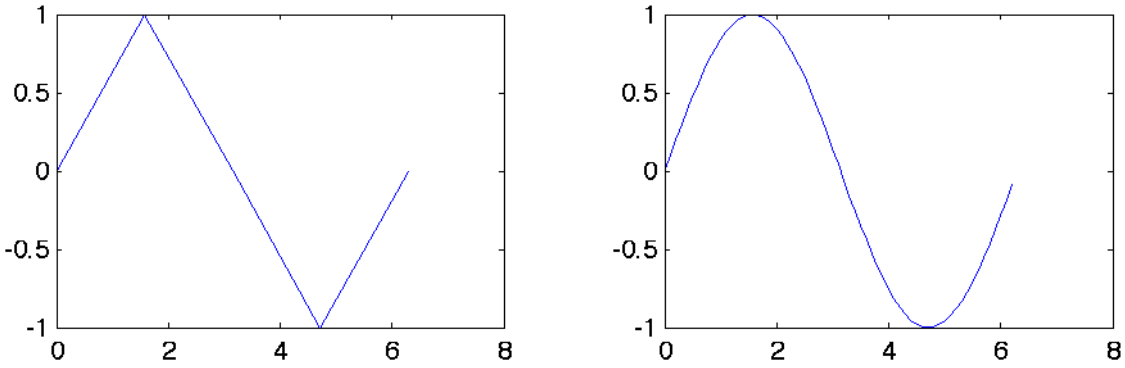


Figure 2.6. A plots of $y = \sin x$.

In **Figure 2.6**(a), there are not enough plotted points to give a true picture of the graph of $y = \sin x$. In a second attempt, we use Matlab's **start:increment:finish** construct to examine the value of the sine at an increased number of values of x. We use semicolons to suppress the output to the display.

```
>> x=0:0.1:2*pi;
>> y=sin(x);
>> plot(x,y)
```

The result of these commands is the plot of the graph of $y = \sin x$ shown in **Figure 2.6**(b).

Matlab's **linspace** command is useful for creating a range of values for plotting. The syntax **linspace(a,b,n)** generates n points between the values of a and b, including the values at a and b. The following sequence of commands were used to create the graph of $y = \sin x$ in **Figure 2.7**.

```
>> x=linspace(0,2*pi,200);
>> y=sin(x);
>> plot(x,y)
>> axis tight
>> xlabel('x-axis')
>> ylabel('y-axis')
>> title('The graph of y=sin(x).')
```

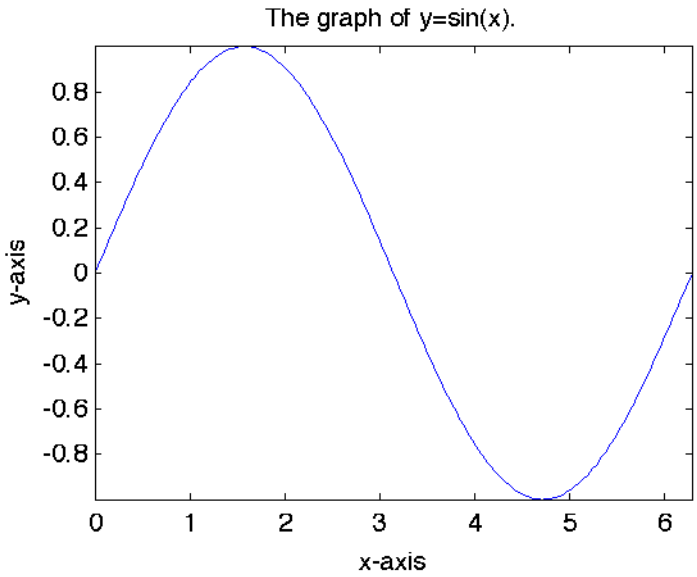


Figure 2.7. The graph of $y = \sin x$ on the interval $[0, 2\pi]$.

Some comments are in order.

1. The command **linspace(0,2*pi,200)** generates 200 equally spaced (linearly spaced) points between 0 and 2π, including the values of 0 and 2π.
2. The **axis tight** command "tightens" the axes window to the plot.
3. Matlab's **xlabel** accepts string input (delimited with single apostrophes) and uses the text to annotate the horizontal axis of the plot.

4. Similarly, the **ylabel** command is used to annotate the vertical axis.
5. Finally, Matlab's **title** command accepts string input and uses the text to annotate the plot with a title.

Script Files

One soon tires of typing commands interactively at the prompt in Matlab's command window. Commands that can be typed sequentially at the prompt in the Matlab command window can be placed into a file and executed en-mass.

As an example, type the following command at the Matlab prompt to open Matlab's editor.

```
>> edit
```

When the editor opens, type in the lines we used above to plot the graph of the sine function on the interval $[0, 2\pi]$.

```
x=linspace(0,2*pi,200);
y=sin(x);
plot(x,y)
axis tight
xlabel('x-axis')
ylabel('y-axis')
title('The graph of y=sin(x).')
```

Save the file in a directory of your choice as **sineplot.m**. You must always save a script file with the extension **.m** attached, as in **sineplot.m**. You are free to choose any filename you wish, but avoid naming the file with a name reserved by Matlab. For example, it would be unwise to name a script file **plot.m**, because you would no longer have access to Matlab's **plot** command.

To execute the script **sineplot.m**, you have two choices.

1. While in Matlab's Editor, press the F5 key to execute the script. *Note: If you save the file in a directory other than Matlab's current directory (the one shown in the navigator window in the toolbar), pressing F5 will bring up a dialog with choices: (1) Change the Matlab current directory, (2) Add directory to the top of Matlab path, and (3) Add directory to the bottom of*

the Matlab path. Select change Matlab current directory and click the OK button.

2. You can also return to the command window and enter the filename of the script at the Matlab prompt, as in **sineplot**. In this case, you need not include the extension.

The script should execute and display the graph of $y = \sin x$, as shown in **Figure 2.7**. If your script fails to execute, read further to determine the source of the problem.

Trouble Shooting. If your script won't execute, here are some things to check.

1. If a another script of the same name is being executed instead of yours, a helpful Matlab command is the command **which sineplot.m**. This will respond with the path to the script **sineplot.m** which will be executed when the user types **sineplot** at the command prompt. If the path is not as expected, you then know that Matlab is calling a different **sineplot.m**.
2. Another useful command is **type sineplot**. This will produce a listing of the file **sineplot.m** in the command window. You can then read the script and see if it is the one you expect to execute.

Matlab's Path. Matlab has a search path which contains a list of directories that it searches when you type a filename or command at the Matlab prompt. You can view the path by typing the following at the command prompt.

```
>> path
```

This will cause a long list of directories to be displayed on the screen. In searching for commands and files, Matlab obeys the following rules, in the following order.

1. Matlab searches the current directory first. You can determine the current directory by typing the command **pwd** (present working directory — a UNIX command that Matlab understands), or by viewing the navigation window on the toolbar atop of the command window.

2. After searching the current directory, Matlab will search directories according to the order shown in the output of the **path** command.

Changing the Current Directory. We've said that Matlab searches the current directory first. If you save your script in a location other than Matlab's current directory, Matlab will search for your script by looking in directories in

the order dictated by Matlab's path. Thus, the usual cure for a script that won't run is to change Matlab's current directory to the directory in which your script resides. This can be accomplished in one of two ways.

1. In the toolbar below the menus, there is a navigation window that indicates the path to the current directory. To the right of this navigation window is a button with three dots. If you click this, you can browse the directory hierarchy of your machine and select the directory containing your script file. The navigation window will reflect your change of directories, indicating the new current directory. If this directory now matches where you saved your script, typing the name of your script at the prompt in the command window should now execute your script.
2. If you are more command-line oriented, then you can use the UNIX command `cd` ("change directory") to change the current directory to the directory containing your script file. On a Mac, if I know I saved the file in `~/tmp`, then this command will change Matlab's current directory to that directory.

   ```
   >> cd ~/tmp
   ```

   You can check the result of this command with the UNIX command `pwd` ("present working directory").

   ```
   >> pwd
   ans =
   /Users/darnold/tmp
   ```

   On a PC, I might save my script in the file `C:\homework`. Executing the following command will set Matlab's current directory to match the directory containing your script.

   ```
   >> cd C:\homework
   ```

If your current directory matches the directory containing your script, you should be able to execute your script by typing **sineplot** at the prompt in the command window.

Adding Directories or Folders to Matlab's Path. If you have a collection of important files that you use quite often, you won't want to be constantly changing the current directory to use them. In this case, you can add the directory

in which these files reside to Matlab's path. Type the following command at the Matlab prompt.

```
>> pathtool
```

Click the button **Add Folder ...** then browse your directory hierarchy and select the folder or directory that you want to add to Matlab's path. Once a folder is selected, there are buttons to move it up or down in the path hierarchy (remember, Matlab searches for files in the order given in the path, top to bottom). You can **Close** the path tool, which means that your adjustment to Matlab's path is only temporary. Next time you start Matlab, changes made to the path will no longer exist. Alternatively, you can make changes to the path permanent (you can come back and make changes later) by using the **Save** button in the path tool. After saving, use the **Close** button to close the pathtool.

The pathtool should be used sparingly. If you are working on an assigment, it is better to have a folder dedicated to that assignment, then change the current directory to that folder and begin working. However, if you have a number of useful utility files to which you would like to maintain access at all times, then that is a folder of files that warrants being added to Matlab's path.

If you are working at school, there is a file named **pathdef.m** in your home directory (H:\ ) that is executed at startup to initialize Matlab's path. If you wish to make permanent changes to the path using the path tool, we recommend that you first change the current directory to your root home directory with this command.

```
>> cd ~/
```

Now you can execute the command **pathtool** and make additions, deletions, or edits to your path. Once you have saved your changes and closed the path tool, change the current directory back to where your were and continue working.

If you are working at home, the file **pathdef.m** is contained in Matlab's hierarchy, so it doesn't matter where you are when you open the **pathtool**.

2.4 Exercises

1. Enter **H=hilb(3)** and change the display output to rational format with the command **format rat**. Use the appropriate command to square each entry of matrix H.

2. Enter **P=pascal(3)** and change the display output to rational format with the command **format rat**. Use the appropriate command to square each entry of matrix P.

3. Create a diagonal matrix with the commands **v=1:3** and **D=diag(v)**. Take the exponetial of each element of the matrix D and explain the result.

4. Create the vector **v=1:5**, then take the exponential of each entry of the vector **v**. Create a diagnonal matrix with the resulting vector. How does this differ from the answer found in **Exercise 3**?

5. The sum of the squares of the integers from 1 to n is given by the formula

$$\frac{n(n+1)(2n+1)}{6}.$$

a) Use the formula to determine the sum of the squares of the integers from 1 to 20, inclusive.

b) The following **for** loop will sum the the squares of the integers from 1 to 20, inclusive.

```
s=1;
for k=2:20
    s=s+k^2;
end
s
```

Verify that this **for** loop produces the same result as in part (a).

c) The following code sums the squares of the integers from 1 to 20, inclusive.

```
s = sum((1:20).^2)
```

Explain why.

6. Use the formula of **Exercise 1** to find the sum of the squares of the integers from 1 to 1000, inclusive.

a) Write a **for** loop to calculate the sum of the squares of the integers from 1 to 1000, inclusive.

b) Use arrays and Matlab's **sum** command to calculate the sum of the squares of the integers from 1 to 1000.

7. The sum of the cubes of the integers from 1 to n can be computed with the formula

$$\left[\frac{n(n+1)}{2}\right]^2.$$

a) Use the formula to determine the

sum of the cubes of the integers from 1 to 1000, inclusive.

b) Write a **for** loop to calculate the sum of the cubes of the integers from 1 to 1000, inclusive.

c) Use arrays and Matlab's **sum** command to calculate the sum of the cubes of the integers from 1 to 1000, inclusive.

8. Save the following in a scriptfile named **quicker.m**.

```
tic
s=1;
for k=2:100000
    s=s+k^3;
end
s
toc

tic
s=sum((1:100000).^3)
toc
```

The Matlab pair **tic** and **toc** record the time for the commands they enclose to execute. Run the script several times by typing **quicker** at the Matlab prompt. Is the **for** loop slower or faster than the arrary technique for computing the sum of the cubes of the integers between 1 and 100 000, inclusive?

9. To learn why it is important to initialize vectors, execute the following code.

```
clc
clear all
N=1000000
tic
a=zeros(N,1);
for k=1:N
    a(k)=1/k;
end
toc
```

Now try the same thing without initializing the vector **a**.

```
clc
clear all
N=1000000
tic
for k=1:N
    a(k)=1/k;
end
toc
```

Results will vary per machine. This causes my machine to hang and I have to break the program by typing **Ctrl+C** in the Matlab command window. Trying adding or deleting a zero from $N = 1000000$ to see how your machine reacts. This is an important lesson on the importance of initializing memory.

In **Exercises 10-19**, perform each of the following tasks for the given sequence.

i. Initialize a column vector of zeros with **a=zeros(10,1)**. Write a **for** loop to populate the vector with the first ten terms of the given sequence. *Note: You might find the*

display output **format rat** *helpful.*

ii. Initialize a column vector **n** with **n=(1:10).'**. Use array operations to generate the first 10 terms of the sequence and store the results in the vector **a** (without the use of a for loop — array ops only).

iii. Plot the resulting vector **a** versus its index with **stem(a)**.

10. $a_n = (-3)^n$

11. $a_n = 2^n$

12. $a_n = \frac{1}{n}$

13. $a_n = \frac{1}{n^2}$

14. $a_n = \frac{(-1)^n}{n}$

15. $a_n = \frac{1}{3^n}$

16. $a_n = \frac{n}{n+1}$

17. $a_n = \frac{1-n}{n+2}$

18. $a_n = \sin\frac{n\pi}{5}$

19. $a_n = \cos\frac{2n\pi}{5}$

---

20. The *Fibonacci Sequence*,

$$1,\ 1,\ 2,\ 3,\ 5,\ 8,\ \ldots$$

is defined recursively by first setting $a_1 = 1$, $a_2 = 1$, and thereafter,

$$a_n = a_{n-1} + a_{n-2}.$$

Initialize a column vector **a** of length 100 with zeros, then write a **for** loop to populate the vector **a** with the first 100 terms of the Fibonacci Sequence. Use Matlab indexing to determine the 80th term of the sequence.

21. We define a sequence by first setting $a_1 = 1$. Thereafter,

$$a_n = \sqrt{2 + a_{n-1}}.$$

Initialize a column vector **a** of length 30 with zeros, then write a **for** loop to populate the vector **a** with the first 30 terms of the sequence. This sequence appears to converge to what number?

22. A sequence begins with the terms

$$\sqrt{2},\quad \sqrt{2\sqrt{2}},\quad \sqrt{2\sqrt{2\sqrt{2}}},\quad \ldots.$$

Write a **for** loop to generate the first 30 terms of this sequence. This sequence appears to converge to what number?

In **Exercises 22-30**, perform each of the following tasks.

i. Write a **for** loop to sum the given series.

ii. Use Matlab's **sum** command and array operations to perform the same task.

23. $\displaystyle\sum_{n=1}^{20} \frac{1}{n}$

24. $\displaystyle\sum_{n=1}^{20} \frac{1}{n^2}$

25. $\displaystyle\sum_{n=1}^{20} \frac{1}{2^n}$

26. $\displaystyle\sum_{n=1}^{20} \frac{n+1}{n}$

27. $\displaystyle\sum_{n=1}^{20} \frac{n}{n+1}$

28. $\displaystyle\sum_{n=1}^{20} 3^{-n}$

29. $\displaystyle\sum_{n=1}^{20} \frac{1}{n!}$

30. $\displaystyle\sum_{n=1}^{20} \frac{2^n}{n!}$

In **Exercises 30-38**, perform each of the following tasks.

i. In each case, create a script file to draw the graph of the given function. Include a printout of the resulting plot and the script file that produced it with your homework. *Hint: Type* **help elfun** *for help on using the given function in Matlab.*
ii. Use **x=linspace(a,b,n)** to produce enough domain values to produce a "smooth curve." Use the default form **plot(x,y)** to produce a solid blue curve.
iii. Label the horizontal and vertical axes with **xlabel** and **ylabel**.
iv. Use **title** to provide a table containing the equation of the given function and the requested domain.

31. Sketch $y = \cos x$ on the interval $[-2\pi, 2\pi]$.

32. Sketch $y = \sin^{-1} x$ on the interval $[-1, 2]$.

33. Sketch $y = |x|$ on the interval $[-2, 2]$.

34. Sketch $y = \ln x$ on the interval $[0.1, 10]$.

35. Sketch $y = e^x$ on the interval $[-2, 2]$.

36. Sketch $y = \cosh x$ on the interval $[-3, 3]$.

37. Sketch $y = \sinh^{-1} x$ on the interval $[-10, 10]$.

38. Sketch $y = \tan^{-1} x$ on the interval $[-10, 10]$.

2.4 Answers

1. Create the Hilbert matrix.

```
>> H=hilb(3)
H =
   1          1/2        1/3
   1/2        1/3        1/4
   1/3        1/4        1/5
```

Square each entry as follows.

```
>> H.^2
ans =
   1          1/4        1/9
   1/4        1/9        1/16
   1/9        1/16       1/25
```

Return to default display format.

```
>> format
```

3. Enter the vector **v**.

```
>> v=1:3
v =
     1     2     3
```

Create the diagonal matrix D.

```
>> D=diag(v)
D =
     1     0     0
     0     2     0
     0     0     3
```

Take the exponential of each entry of the matrix D.

```
>> E=exp(D)
E =
   2.7183    1.0000    1.0000
   1.0000    7.3891    1.0000
   1.0000    1.0000   20.0855
```

Off the diagonal, $e^0 = 1$, so each entry is a 1. On the diagonal $e^1 \approx 2.7183$, $e^2 \approx 7.3891$, and $e^3 \approx 20.0855$.

5. Set $n = 20$.

a) Then:

```
>> n=20;
>> n*(n+1)*(2*n+1)/6
ans =
        2870
```

b) Verify with **for** loop.

```
>> s=1;
>> for k=2:20
s=s+k^2;
end
>> s
s =
        2870
```

c) Array ops produce the same result.

```
>> s=sum((1:20).^2)
s =
        2870
```

The **(1:20)** produces a row vector with entries from 1 to 20. The array exponentiation **.^2** squares each entry of the vector. The **sum** command adds the entries of the resulting vector (the squares from 1 to 20).

7. Set $n = 1000$.

a) Then:

```
>> format long g
>> n=1000;
>> (n*(n+1)/2)^2
ans =
      250500250000
```

b) Sum with a **for** loop.

```
>> s=0;
>> for k=1:1000
s=s+k^3;
end
>> s
s =
    250500250000
```

c) Same result with array ops:

```
>> s=sum((1:1000).^3)
s =
    250500250000
```

11. A **for** loop.

```
>> a=zeros(10,1);
>> for k=1:10,
a(k)=2^k;
end
>> a
a =
           2
           4
           8
          16
          32
          64
         128
         256
         512
        1024
```

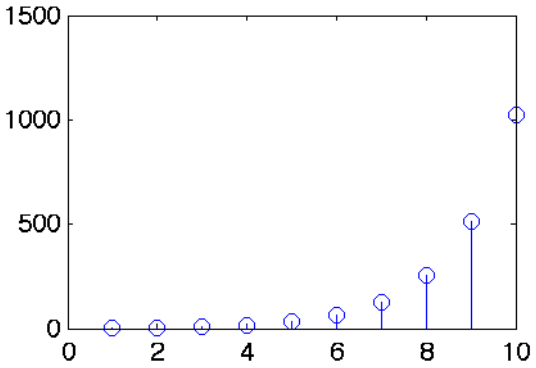
Same result with array ops.

```
>> a=2.^n
a =
           2
           4
           8
          16
          32
          64
         128
         256
         512
        1024
```

A stem plot.

```
>> stem(a)
```

The resulting stem plot.



13. A **for** loop.

```
>> a=zeros(10,1);
>> for k=1:10,
a(k)=1/k^2;
end
>> a
a =
       1
       1/4
       1/9
       1/16
       1/25
       1/36
       1/49
       1/64
       1/81
       1/100
```

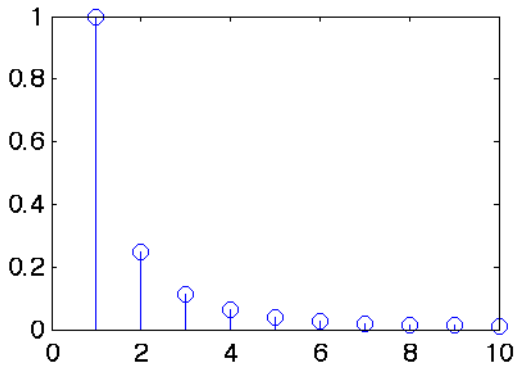
Same result with array ops.

```
>> n=(1:10).';
>> a=1./(n.^2)
a =
       1
       1/4
       1/9
       1/16
       1/25
       1/36
       1/49
       1/64
       1/81
       1/100
```

A stem plot.

```
>> stem(a)
```

The resulting stem plot.



15. A **for** loop.

```
>> a=zeros(10,1);
>> for k=1:10,
a(k)=1/(3^k);
end
>> a
a =
        1/3
        1/9
        1/27
        1/81
        1/243
        1/729
        1/2187
        1/6561
        1/19683
        1/59049
```

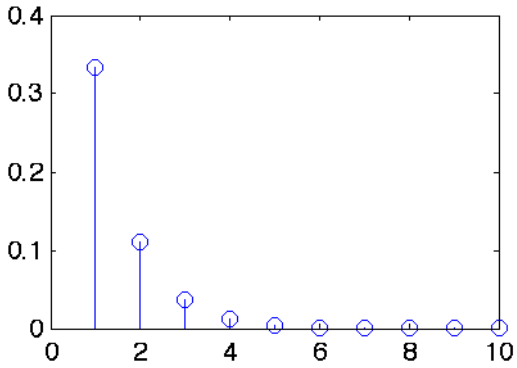
Same result with array ops.

```
>> a=1./(3.^n)
a =
        1/3
        1/9
        1/27
        1/81
        1/243
        1/729
        1/2187
        1/6561
        1/19683
        1/59049
```

A stem plot.

```
>> stem(a)
```

The resulting plot.



17. A **for** loop.

```
>> a=zeros(10,1);
>> for k=1:10,
a(k)=(1-k)/(k+2);
end
>> a
a =
         0
      -1/4
      -2/5
      -1/2
      -4/7
      -5/8
      -2/3
      -7/10
      -8/11
      -3/4
```

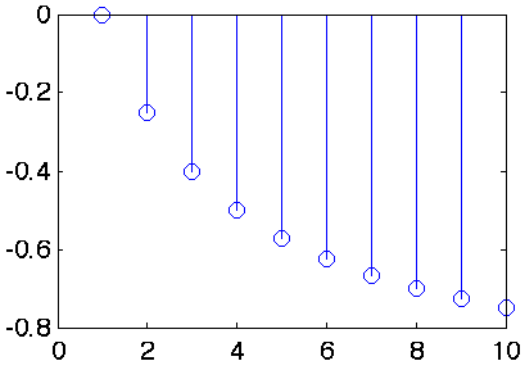
Same result with array ops.

```
>> a=(1-n)./(n+2)
a =
       0
      -1/4
      -2/5
      -1/2
      -4/7
      -5/8
      -2/3
      -7/10
      -8/11
      -3/4
```

A stem plot.

```
>> stem(a)
```

The resulting plot.



19. A **for** loop with default format.

```
>> format
>> a=zeros(10,1);
>> for k=1:10,
a(k)=cos(2*k*pi/5);
end
>> a
a =
    0.3090
   -0.8090
   -0.8090
    0.3090
    1.0000
    0.3090
   -0.8090
   -0.8090
    0.3090
    1.0000
```

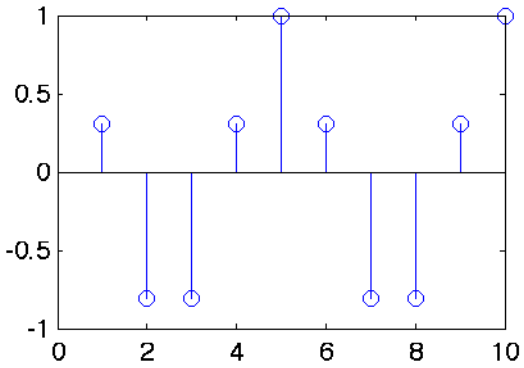
Same result with array ops.

```
>> n=(1:10).';
>> a=cos(2*n*pi/5)
a =
    0.3090
   -0.8090
   -0.8090
    0.3090
    1.0000
    0.3090
   -0.8090
   -0.8090
    0.3090
    1.0000
```

A stem plot.

```
>> stem(a)
```

The resulting plot.



21. Iniialization and **for** loop.

```
>> a=zeros(30,1);
>> a(1)=1;
>> for k=2:30
a(k)=sqrt(2+a(k-1));
end
```

Result.

```
>> format long
>> a
a =
   1.00000000000000
   1.73205080756888
   1.93185165257814
   1.98288972274762
   1.99571784647721
   1.99892917495273
   1.99973227581912
   1.99993306783480
   1.99998326688870
   1.99999581671780
   1.99999895417918
   1.99999973854478
   1.99999993463619
   1.99999998365905
   1.99999999591476
   1.99999999897869
   1.99999999974467
   1.99999999993617
   1.99999999998404
   1.99999999999601
   1.99999999999900
   1.99999999999975
   1.99999999999994
   1.99999999999998
   2.00000000000000
   2.00000000000000
   2.00000000000000
   2.00000000000000
   2.00000000000000
   2.00000000000000
```

It would appear that this sequence comverges to the number 2.

23. Summing with a **for** loop using default display format.

```
>> format
>> s=0;
>> for n=1:20,
s=s+1/n;
end
>> s
s =
   3.59773965714368
```

Using array ops.

```
>> n=1:20;
>> s=sum(1./n)
s =
   3.59773965714368
```

25. Summing with a **for** loop using default display format.

```
>> format
>> s=0;
>> for n=1:20,
s=s+1/(2^n);
end
>> s
s =
   0.99999904632568
```

Using array ops.

```
>> n=1:20;
>> s=sum(1./(2.^n))
s =
```

27. Summing with a **for** loop using default display format.

```
>> s=0;
>> for n=1:20,
s=s+n/(n+1);
end
>> s
s =
  17.35464129523727
```

Using array ops.

```
>> s=sum(n./(n+1))
s =
  17.35464129523727
```

29. Summing with a **for** loop using default display format.

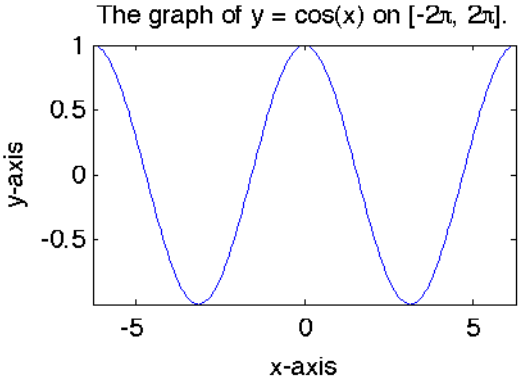
```
>> s=0;
>> for n=1:20,
s=s+1/factorial(n);
end
>> s
s =
   1.71828182845905
```

Using array ops.

```
>> s=sum(1./factorial(n))
s =
   1.71828182845905
```

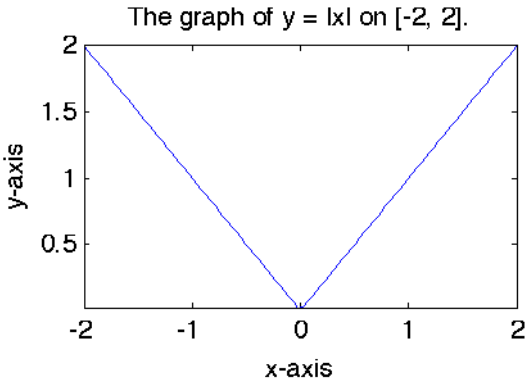
31. The following script file was used to produce that graph that follows.

```
clear all
close all
x=linspace(-2*pi,2*pi,200);
y=cos(x);
plot(x,y)
axis tight
xlabel('x-axis')
ylabel('y-axis')
title('The graph of y = cos(x)
on [-2\pi, 2\pi].')
```



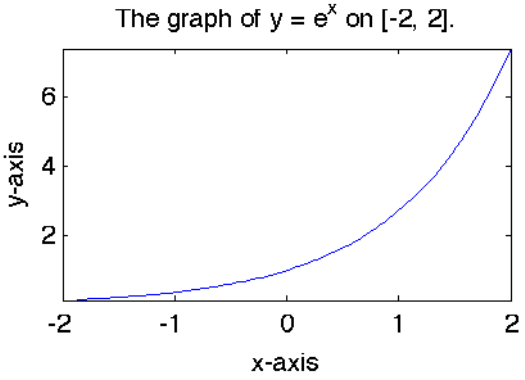
33. The following script file was used to produce that graph that follows.

```
clear all
close all
x=linspace(-2,2,100);
y=abs(x);
plot(x,y)
axis tight
xlabel('x-axis')
ylabel('y-axis')
title('The graph of y = |x| on
[-2, 2].')
```



35. The following script file was used to produce that graph that follows.

```
clear all
close all
x=linspace(-2,2,100);
y=exp(x);
plot(x,y)
axis tight
xlabel('x-axis')
ylabel('y-axis')
title('The graph of y = e^x on
[-2, 2].')
```



37. The following script file was used to produce that graph that follows.

```
clear all
close all
x=linspace(-10,10,100);
y=asinh(x);
plot(x,y)
axis tight
xlabel('x-axis')
ylabel('y-axis')
title('The graph of y = sinh^{-
1}(x) on [-10, 10].')
```