MATLAB: A Practical Introduction to Programming and Problem Solving

Fourth Edition

SOLUTION MANUAL

Stormy Attaway

College of Engineering
Boston University

Chapter 1: Introduction to MATLAB

Exercises

1) Create a variable *myage* and store your age in it. Subtract 2 from the value of the variable. Add 1 to the value of the variable. Observe the Workspace Window and Command History Window as you do this.

```
>> myage = 20;
>> myage = myage - 2;
>> myage = myage + 1;
```

2) Explain the difference between these two statements:

```
result = 9*2
result = 9*2;
```

Both will store 18 in the variable result. In the first, MATLAB will display this in the Command Window; in the second, it will not.

3) Use the built-in function **namelengthmax** to find out the maximum number of characters that you can have in an identifier name under your version of MATLAB.

```
>> namelengthmax
ans =
    63
```

4) Create two variables to store a weight in pounds and ounces. Use **who** and **whos** to see the variables. Use **class** to see the types of the variables. Clear one of them and then use **who** and **whos** again.

```
>> pounds = 4;
>> ounces = 3.3;
>> who

Your variables are:

ounces  pounds

>> whos
  Name         Size            Bytes  Class     Attributes

  ounces       1x1                 8  double
  pounds       1x1                 8  double
```

```
    >> clear pounds
    >> who

    Your variables are:

    ounces
```

5) Explore the **format** command in more detail. Use **help format** to find options. Experiment with **format bank** to display dollar values.

```
  >> format +
  >> 12.34
  ans =
  +
  >> -123
  ans =
  -
  >> format bank
  >> 33.4
  ans =
           33.40
  >> 52.435
  ans =
           52.44
```

6) Find a **format** option that would result in the following output format:

```
    >> 5/16 + 2/7
    ans =
         67/112

  >> format rat
  >> 5/16 + 2/7
  ans =
        67/112
```

7) Think about what the results would be for the following expressions, and then type them in to verify your answers.

```
    25 / 5 * 5
    4 + 3 ^ 2
    (4 + 3) ^ 2
    3 \ 12 + 5
    4 - 2 * 3

  >> 25/5*5
  ans =
     25
```

```
>> 4 + 3 ^ 2
ans =
    13
>> (4 + 3) ^ 2
ans =
    49
>> 3 \ 12 + 5
ans =
     9
>> 4 - 2 * 3
ans =
    -2
```

As the world becomes more "flat", it is increasingly important for engineers and scientists to be able to work with colleagues in other parts of the world. Correct conversion of data from one system of units to another (for example, from the metric system to the American system or vice versa) is critically important.

8) Create a variable *pounds* to store a weight in pounds. Convert this to kilograms and assign the result to a variable *kilos*. The conversion factor is 1 kilogram = 2.2 lb.

```
>> pounds = 30;
>> kilos = pounds / 2.2
kilos =
   13.6364
```

9) Create a variable *ftemp* to store a temperature in degrees Fahrenheit (F). Convert this to degrees Celsius (C) and store the result in a variable *ctemp*. The conversion factor is C = (F – 32) * 5/9.

```
>> ftemp = 75;
>> ctemp = (ftemp - 32) * 5/9
ctemp =
   23.8889
```

10) The following assignment statements either contain at least one error, or could be improved in some way. Assume that *radius* is a variable that has been initialized. First, identify the problem, and then fix and/or improve them:

```
     33 = number

          The variable is always on the left
          number = 33

     my variable =  11.11;
```

```
          Spaces are not allowed in variable names
          my_variable = 11.11;

     area = 3.14 * radius^2;

          Using pi is more accurate than 3.14
          area = pi * radius^2;

     x = 2 * 3.14 * radius;

          x is not a descriptive variable name
          circumference  =  2 * pi * radius;
```

11) Experiment with the functional form of some operators such as **plus**, **minus**, and **times**.

```
>> plus(4, 8)
ans =
    12
>> plus(3, -2)
ans =
     1
>> minus(5, 7)
ans =
    -2
>> minus(7, 5)
ans =
     2
>> times(2, 8)
ans =
    16
```

12) Generate a random

- real number in the range (0, 20)

```
rand * 20
```

- real number in the range (20, 50)

```
rand*(50-20)+20
```

- integer in the inclusive range from 1 to 10

```
randi(10)
```

- integer in the inclusive range from 0 to 10

```
randi([0, 10])
```

- integer in the inclusive range from 50 to 100

```
randi([50, 100])
```

13) Get into a new Command Window, and type **rand** to get a random real number. Make a note of the number. Then, exit MATLAB and repeat this, again making a note of the random number; it should be the same as before. Finally, exit MATLAB and again get into a new Command Window. This time, change the seed before generating a random number; it should be different.

```
>> rand
ans =
0.8147

>> rng('shuffle')
>> rand
ans =
0.4808
```

14) What is the difference between x and 'x'?

```
In an expression, the first would be interpreted as the name of
a variable, whereas 'x' is the character x.
```

15) What is the difference between 5 and '5'?

```
The first is the number 5, the second is the character 5.
(Note: int32(5) is 53.  So, 5+1 would be 6.  '5'+1 would be54.)
```

16) The combined resistance R_T of three resistors R_1, R_2, and R_3 in parallel is given by

$$R_T = \frac{1}{\frac{1}{R_1}+\frac{1}{R_2}+\frac{1}{R_3}}$$

Create variables for the three resistors and store values in each, and then calculate the combined resistance.

```
>> r1 = 3;
>> r2 = 2.2;
>> r3 = 1.5;
>> rt = 1/(1/r1 + 1/r2 + 1/r3)
rt =
    0.6875
```

17) Explain the difference between constants and variables.

```
Constants store values that are known and do not change.
Variables are used when the value will change, or when the value
is not known to begin with (e.g., the user will provide the
value).
```

18) What would be the result of the following expressions?

```
     'b' >= 'c' - 1           1

     3 == 2 + 1               1

     (3 == 2) + 1             1

     xor(5 < 6, 8 > 4)        0

     10 > 5 > 2

     0    Evaluated from left to right: 10>5 is 1,
          then 1 > 2 is 0

     result = 3^2 - 20;
     0 <= result <= 10

          1    Evaluated left to right: 0 <= result is 0,
          then 0 <= 10 is 1
```

19) Create two variables *x* and *y* and store numbers in them. Write an expression that would be **true** if the value of *x* is greater than five or if the value of *y* is less than ten, but not if both of those are **true**.

```
  >> x = 3;
  >> y = 12;
  >> xor(x > 5, y < 10)
  ans =
       0
```

20) Use the equality operator to verify that 3*10^5 is equal to 3e5.

```
  >> 3*10^5 == 3e5
  ans =
       1
```

21) In the ASCII character encoding, the letters of the alphabet are in order: ‘a’ comes before ‘b’ and also ‘A’ comes before ‘B’. However, which comes first - lower or uppercase letters?

```
>> int32('a')
ans =
          97
>> int32('A')
ans =
          65

The upper case letters
```

22) Are there equivalents to **intmin** and **intmax** for real number types? Use **help** to find out.

```
>> realmin
ans =
  2.2251e-308
>> realmin('double')
ans =
  2.2251e-308
>> realmin('single')
ans =
  1.1755e-38
>> realmax
ans =
  1.7977e+308
```

23) Use **intmin** and **intmax** to determine the range of values that can be stored in the types **uint32** and **uint64**.

```
>> intmin('uint32')
ans =
           0
>> intmax('uint32')
ans =
  4294967295
>> intmin('uint64')
ans =
                    0
>> intmax('uint64')
ans =
 18446744073709551615
```

24) Use the **cast** function to cast a variable to be the same type as another variable.

```
    >> vara = uint16(3 + 5)
    vara =
          8
    >> varb = 4*5;
```

```
>> class(varb)
ans =
double
>> varb = cast(varb, 'like', vara)
varb =
     20
>> class(varb)
ans =
uint16
```

25) Use **help elfun** or experiment to answer the following questions:

- Is **fix(3.5)** the same as **floor(3.5)**?

```
>> fix(3.5)
ans =
     3
>> floor(3.5)
ans =
     3
```

- Is **fix(3.4)** the same as **fix(-3.4)**?

```
>> fix(3.4)
ans =
     3
>> fix(-3.4)
ans =
    -3
```

- Is **fix(3.2)** the same as **floor(3.2)**?

```
>> fix(3.2)
ans =
     3
>> floor(3.2)
ans =
     3
```

- Is **fix(-3.2)** the same as **floor(-3.2)**?

```
>> fix(-3.2)
ans =
    -3
>> floor(-3.2)
ans =
    -4
```

- Is **fix(-3.2)** the same as **ceil(-3.2)**?

```
>> fix(-3.2)
ans =
    -3
>> ceil(-3.2)
ans =
    -3
```

26) For what range of values is the function **round** equivalent to the function **floor**?

For positive numbers: when the decimal part is less than .5

For negative numbers: when the decimal part is greater than or equal to .5

For what range of values is the function **round** equivalent to the function **ceil**?

For positive numbers: when the decimal part is greater than or equal to .5

For negative numbers: when the decimal part is less than .5

27) Use **help** to determine the difference between the **rem** and **mod** functions.

```
>> help rem
 rem    Remainder after division.
    rem(x,y) is x - n.*y where n = fix(x./y) if y ~= 0.
    By convention:
       rem(x,0) is NaN.
       rem(x,x), for x~=0, is 0.
       rem(x,y), for x~=y and y~=0, has the same sign as x.

rem(x,y) and MOD(x,y) are equal if x and y have the same sign,
butdiffer by y if x and y have different signs.

>> help mod
 mod    Modulus after division.
    mod(x,y) is x - n.*y where n = floor(x./y) if y ~= 0.

    By convention:
       mod(x,0) is x.
       mod(x,x) is 0.
       mod(x,y), for x~=y and y~=0, has the same sign as y.
```

28) Find MATLAB expressions for the following

$\sqrt{19}$

```
sqrt(19)
```

$3^{1.2}$

```
3^1.2

tan(π)

tan(pi)
```

29) Using only the integers 2 and 3, write as many expressions as you can that result in 9. Try to come up with at least 10 different expressions (e.g., don't just change the order). Be creative! Make sure that you write them as MATLAB expressions. Use operators and/or built-in functions.

```
3 ^ 2

2 ^ 3 + (3 - 2)

3 * 3

3 ^ 3 - 3 * 3 * 2

2^3 + abs(2-3)

2^3 + sign(3)

3/2*2*3

2\3*2*3

sqrt(3^(2+2))

nthroot(3^(2+2),2)
```

30) A vector can be represented by its rectangular coordinates x and y or by its polar coordinates r and θ. Theta is measured in radians. The relationship between them is given by the equations:

```
x = r * cos(θ)
y = r * sin(θ)
```

Assign values for the polar coordinates to variables *r* and *theta*. Then, using these values, assign the corresponding rectangular coordinates to variables *x* and *y*.

```
>> r = 5;
>> theta = 0.5;
>> x = r * cos(theta)
```

```
x =
    4.3879
>> y = r * sin(theta)
y =
    2.3971
```

31) In special relativity, the Lorentz factor is a number that describes the effect of speed on various physical properties when the speed is significant relative to the speed of light. Mathematically, the Lorentz factor is given as:

$$\gamma = \frac{1}{\sqrt{1-\frac{v^2}{c^2}}}$$

Use 3×10^8 m/s for the speed of light, *c*. Create variables for *c* and the speed *v* and from them a variable *lorentz* for the Lorentz factor.

```
>> c = 3e8;
>> v = 2.9e8;
>> lorentz = 1 / sqrt(1 - v^2/c^2)
lorentz =
    3.9057
```

32) A company manufactures a part for which there is a desired weight. There is a tolerance of N percent, meaning that the range between minus and plus N% of the desired weight is acceptable. Create a variable that stores a weight, and another variable for N (for example, set it to two). Create variables that store the minimum and maximum values in the acceptable range of weights for this part.

```
>> weight = 12.3;
>> N = 2;
>> min = weight - weight*0.01*N
min =
   12.0540
>> max = weight + weight*0.01*N
max =
   12.5460
```

33) An environmental engineer has determined that the cost C of a containment tank will be based on the radius *r* of the tank:

$$C = \frac{32430}{r} + 428\pi r$$

Create a variable for the radius, and then for the cost.

```
>> format bank
>> radius = 11;
>> cost = 32430/radius + 428*pi*radius
cost =
      17738.80
```

34) A chemical plant releases an amount A of pollutant into a stream. The maximum concentration C of the pollutant at a point which is a distance x from the plant is:

$$C = \frac{A}{x}\sqrt{\frac{2}{\Pi e}}$$

Create variables for the values of A and x, and then for C. Assume that the distance x is in meters. Experiment with different values for x.

```
>> A = 30000;
>> x = 100;
>> C = A/x * sqrt(2/(pi*exp(1)))
C =
        145.18
>> x = 1000;
>> C = A/x * sqrt(2/(pi*exp(1)))
C =
         14.52
>> x = 20000;
>> C = A/x * sqrt(2/(pi*exp(1)))
C =
          0.73
```

35) The geometric mean g of n numbers x_i is defined as the nth root of the product of x_i:

$$g = \sqrt[n]{x_1 x_2 x_3 \ldots x_n}$$

(This is useful, for example, in finding the average rate of return for an investment which is something you'd do in engineering economics). If an investment returns 15% the first year, 50% the second, and 30% the third year, the average rate of return would be `(1.15*1.50*1.30)`$^{1/3}$.) Compute this.

```
>> x1 = 1.15;
>> x2 = 1.5;
>> x3 = 1.3;
>> gmean = nthroot(x1*x2*x3, 3)
gmean =
          1.31
```

36) Use the **deg2rad** function to convert 180 degrees to radians.

```
>> deg2rad(180)
ans =
    3.1416
>>
```

Chapter 2: Vectors and Matrices

Exercises

1) If a variable has the dimensions 3 x 4, could it be considered to be (**bold** all that apply):
a matrix
a row vector
a column vector
a scalar

2) If a variable has the dimensions 1 x 5, could it be considered to be (**bold** all that apply):
a matrix
a row vector
a column vector
a scalar

3) If a variable has the dimensions 5 x 1, could it be considered to be (**bold** all that apply):
a matrix
a row vector
a column vector
a scalar

4) If a variable has the dimensions 1 x 1, could it be considered to be (**bold** all that apply):
a matrix
a row vector
a column vector
a scalar

5) Using the colon operator, create the following row vectors

```
     2     3     4     5     6     7

1.1000    1.3000    1.5000    1.7000

     8     6     4     2

>> 2:7
ans =
     2     3     4     5     6     7
>> 1.1:0.2:1.7
ans =
    1.1000    1.3000    1.5000    1.7000
>> 8:-2:2
ans =
     8     6     4     2
```

6) Using a built-in function, create a vector *vec* which consists of 20 equally spaced points in the range from –pi to +pi.

```
vec = linspace(0,2*pi,50);
```

7) Write an expression using **linspace** that will result in the same as 2: 0.2: 3

```
linspace(2,3,6)
```

8) Using the colon operator and also the **linspace** function, create the following row vectors:

```
    -5    -4    -3    -2    -1

     5     7     9

     8     6     4

>> -5:-1
ans =
    -5    -4    -3    -2    -1
>> linspace(-5,-1,5)
ans =
    -5    -4    -3    -2    -1
>> 5:2:9
ans =
     5     7     9
>> linspace(5,9,3)
ans =
     5     7     9
>> 8:-2:4
ans =
     8     6     4
>> linspace(8,4,3)
ans =
     8     6     4
```

9) How many elements would be in the vectors created by the following expressions?

```
linspace(3,2000)
```

```
          100 (always, by default)

    logspace(3,2000)

          50 (always, by default - although these numbers
    would get very large quickly; most would be
          represented as Inf)
```

10) Create a variable *myend* which stores a random integer in the inclusive range from 5 to 9. Using the colon operator, create a vector that iterates from 1 to *myend* in steps of 3.

```
>>myend = randi([5, 9])
myend =
     8
>> vec = 1:3:myend
vec =
     1     4     7
```

11) Using the colon operator and the transpose operator, create a column vector *myvec* that has the values -1 to 1 in steps of 0.5.

```
>> rowVec = -1: 0.5: 1;
>> rowVec'
ans =
   -1.0000
   -0.5000
         0
    0.5000
    1.0000
```

12) Write an expression that refers to only the elements that have odd-numbered subscripts in a vector, regardless of the length of the vector. Test your expression on vectors that have both an odd and even number of elements.

```
>> vec = 1:8;
>> vec(1:2:end)
ans =
     1     3     5     7

>> vec = 4:12
vec =
     4     5     6     7     8     9    10    11    12
>> vec(1:2:end)
ans =
     4     6     8    10    12
```

13) Generate a 2 x 4 matrix variable *mat*. Replace the first row with 1:4. Replace the third column (you decide with which values).

```
>> mat = [2:5; 1 4 11 3]
mat =
     2     3     4     5
     1     4    11     3
>> mat(1,:) = 1:4
mat =
     1     2     3     4
     1     4    11     3
>> mat(:,3) = [4;3]
mat =
     1     2     4     4
     1     4     3     3
```

14) Generate a 2 x 4 matrix variable *mat*. Verify that the number of elements is the product of the number of rows and columns.

```
>> mat = randi(20,2,4)
mat =
     1    19    17     9
    13    15    20    16
>> [r c] = size(mat);
>> numel(mat) == r * c
ans =
     1
```

15) Which would you normally use for a matrix: **length** or **size**? Why?

Definitely **size**, because it tells you both the number of rows and columns.

16) When would you use **length** vs. **size** for a vector?

If you want to know the number of elements, you'd use **length**. If you want to figure out whether it's a row or column vector, you'd use **size**.

17) Generate a *2 x 3* matrix of random

- real numbers, each in the range (0, 1)

```
>> rand(2,3)
ans =
    0.0215    0.7369    0.7125
    0.7208    0.4168    0.1865
```

- real numbers, each in the range (0, 10)

```
>> rand(2,3)*10
ans =
    8.0863    2.2456    8.3067
    2.9409    4.0221    5.0677
```

- integers, each in the inclusive range from 5 to 20

```
>> randi([5, 20],2,3)
ans =
    18    17     5
    11    11     7
```

18) Create a variable *rows* that is a random integer in the inclusive range from 1 to 5. Create a variable *cols* that is a random integer in the inclusive range from 1 to 5. Create a matrix of all zeros with the dimensions given by the values of *rows* and *cols*.

```
>> rows = randi([1,5])
rows =
     3
>> cols = randi([1,5])
cols =
     2
>> zeros(rows,cols)
ans =
     0     0
     0     0
     0     0
```

19) Create a matrix variable *mat*. Find as many expressions as you can that would refer to the last element in the matrix, without assuming that you know how many elements or rows or columns it has (i.e., make your expressions general).

```
>> mat = [12:15; 6:-1:3]
mat =
    12    13    14    15
     6     5     4     3
>> mat(end,end)
ans =
     3
>> mat(end)
ans =
     3
>> [r c] = size(mat);
>> mat(r,c)
ans =
     3
```

20) Create a vector variable *vec*. Find as many expressions as you can that would refer to the last element in the vector, without assuming that you know how many elements it has (i.e., make your expressions general).

```
>> vec = 1:2:9
vec =
     1     3     5     7     9
>> vec(end)
ans =
     9
>> vec(numel(vec))
ans =
     9
>> vec(length(vec))
ans =
     9
>> v = fliplr(vec);
>> v(1)
ans =
     9
```

21) Create a 2 x 3 matrix variable *mat*. Pass this matrix variable to each of the following functions and make sure you understand the result: **flip**, **fliplr**, **flipud**, and **rot90**. In how many different ways can you **reshape** it?

```
>> mat = randi([1,20], 2,3)
mat =
    16     5     8
    15    18     1
>> flip(mat)
ans =
    15    18     1
    16     5     8
>>fliplr(mat)
ans =
     8     5    16
     1    18    15
>> flipud(mat)
ans =
    15    18     1
    16     5     8
>> rot90(mat)
ans =
     8     1
     5    18
    16    15
```

```
>> rot90(rot90(mat))
ans =
     1    18    15
     8     5    16
>> reshape(mat,3,2)
ans =
    16    18
    15     8
     5     1
>> reshape(mat,1,6)
ans =
    16    15     5    18     8     1
>> reshape(mat,6,1)
ans =
    16
    15
     5
    18
     8
     1
```

22) What is the difference between `fliplr(mat)` and `mat = fliplr(mat)`?

The first stores the result in *ans* so *mat* is not changed; the second changes *mat*.

23) Use **reshape** to reshape the row vector 1:4 into a 2x2 matrix; store this in a variable named mat. Next, make 2x3 copies of mat using both **repelem** and **repmat**.

```
>> mat = reshape(1:4,2,2)
mat =
     1     3
     2     4
>> repelem(mat,2,3)
ans =
     1     1     1     3     3     3
     1     1     1     3     3     3
     2     2     2     4     4     4
     2     2     2     4     4     4
>> repmat(mat,2,3)
ans =
     1     3     1     3     1     3
     2     4     2     4     2     4
     1     3     1     3     1     3
     2     4     2     4     2     4
```

24) Create a *3 x 5* matrix of random real numbers. Delete the third row.

```
>> mat = rand(3,5)
mat =
    0.5226    0.9797    0.8757    0.0118    0.2987
    0.8801    0.2714    0.7373    0.8939    0.6614
    0.1730    0.2523    0.1365    0.1991    0.2844

>> mat(3,:) = []
mat =
    0.5226    0.9797    0.8757    0.0118    0.2987
    0.8801    0.2714    0.7373    0.8939    0.6614
```

25) Given the matrix:

```
>> mat = randi([1 20], 3,5)
mat =
     6    17     7    13    17
    17     5     4    10    12
     6    19     6     8    11
```

Why wouldn't this work:

```
mat(2:3, 1:3) = ones(2)

Because the left and right sides are not the same dimensions.
```

26) Create a three-dimensional matrix with dimensions 2 x 4 x 3 in which the first "layer" is all 0s, the second is all 1s and the third is all 5s. Use **size** to verify the dimensions.

```
>> mat3d = zeros(2,4,3);
>> mat3d(:,:,2) = 1;
>> mat3d(:,:,3) = 5;
>> mat3d
mat3d(:,:,1) =
     0     0     0     0
     0     0     0     0
mat3d(:,:,2) =
     1     1     1     1
     1     1     1     1
mat3d(:,:,3) =
     5     5     5     5
     5     5     5     5
```

27) Create a vector x which consists of 20 equally spaced points in the range from $-\pi$ to $+\pi$. Create a y vector which is **sin(x)**.

```
>> x = linspace(-pi,pi,20);
>> y = sin(x);
```

28) Create a *3 x 5* matrix of random integers, each in the inclusive range from -5 to 5. Get the **sign** of every element.

```
>> mat = randi([-5,5], 3,5)
mat =
     5     4     1    -1    -5
     4     4    -1    -3     0
     5    -2     1     0     4
>> sign(mat)
ans =
     1     1     1    -1    -1
     1     1    -1    -1     0
     1    -1     1     0     1
```

29) Find the sum 3+5+7+9+11.

```
>> sum(3:2:11)
ans =
    35
```

30) Find the sum of the first n terms of the harmonic series where n is an integer variable greater than one.

$$1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}+\cdots$$

```
>> n = 4;
>> sum(1./(1:n))
ans =
    2.0833
```

31) Find the following sum by first creating vectors for the numerators and denominators:

$$\frac{3}{1}+\frac{5}{2}+\frac{7}{3}+\frac{9}{4}$$

```
>> num = 3:2:9
num =
     3     5     7     9
>> denom = 1:4
denom =
     1     2     3     4
>> fracs = num ./ denom
fracs =
    3.0000    2.5000    2.3333    2.2500
>> sum(fracs)
ans =
   10.0833
```

32) Create a matrix and find the product of each row and column using **prod**.

```
>> mat = randi([1, 30], 2,3)
mat =
    11    24    16
     5    10     5

>> prod(mat)
ans =
    55   240    80

>> prod(mat,2)
ans =
        4224
         250
```

33) Create a *1 x 6* vector of random integers, each in the inclusive range from 1 to 20. Use built-in functions to find the minimum and maximum values in the vector. Also create a vector of cumulative sums using **cumsum**.

```
vec = randi([1,20], 1,6)
min(vec)
max(vec)
cvec = cumsum(vec)
```

34) Write a relational expression for a vector variable that will verify that the last value in a vector created by **cumsum** is the same as the result returned by **sum**.

```
>> vec = 2:3:17
vec =
     2     5     8    11    14    17
>> cv = cumsum(vec)
cv =
     2     7    15    26    40    57
>> sum(vec) == cv(end)
ans =
     1
```

35) Create a vector of five random integers, each in the inclusive range from -10 to 10. Perform each of the following:

```
>> vec = randi([-10, 10], 1,5)
```

- subtract 3 from each element

```
>> vec-3
```

- count how many are positive

```
>> sum(vec > 0)
```

- get the cumulative minimum

36) Create a *3 x 5* matrix. Perform each of the following:

```
>> mat = randi([-10 10], 3,5)
```

- Find the maximum value in each column.

```
>> max(mat)
```

- Find the maximum value in each row.

```
>> max(mat, [], 2)
>> max(mat')
```

- Find the maximum value in the entire matrix.

```
>> max(max(mat))
```

- Find the cumulative maxima.

```
>> cummax(mat)
```

37) Find two ways to create a *3 x 5* matrix of all 100s (Hint: use **ones** and **zeros**).

```
ones(3,5)*100


zeros(3,5)+100
```

38) Given the two matrices:

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

Perform the following operations:

`A + B`

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

`A - B`

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

`A .* B`

$$\begin{bmatrix} 2 & 8 & 3 \\ 4 & -3 & 0 \end{bmatrix}$$

39) The built-in function **clock** returns a vector that contains 6 elements: the first three are the current date (year, month, day) and the last three represent the current time in hours, minutes, and seconds. The seconds is a real number, but all others are integers. Store the result from clock in a variable called *myc*. Then, store the first three elements from this variable in a variable *today* and the last three elements in a variable *now*. Use the fix function on the vector variable *now* to get just the integer part of the current time.

```
>> myc = clock
myc =
   1.0e+03 *
    2.0130    0.0010    0.0080    0.0120    0.0060    0.0014
>> today = myc(1:3)
today =
        2013           1           8
>> now = myc(4:end)
```

```
now =
   12.0000    6.0000    1.4268
>> fix(now)
ans =
    12     6     1
```

40) A vector *v* stores for several employees of the Green Fuel Cells Corporation their hours worked one week followed for each by the hourly pay rate. For example, if the variable stores

```
>> v
v =
33.0000  10.5000  40.0000  18.0000  20.0000  7.5000
```

that means the first employee worked 33 hours at $10.50 per hour, the second worked 40 hours at $18 an hour, and so on. Write code that will separate this into two vectors, one that stores the hours worked and another that stores the hourly rates. Then, use the array multiplication operator to create a vector, storing in the new vector the total pay for every employee.

```
>> hours = v(1:2:length(v))
hours =
    33    40    20

>> payrate = v(2:2:length(v))
payrate =
   10.5000   18.0000    7.5000

>> totpay = hours .* payrate
totpay =
  346.5000  720.0000  150.0000
```

41) A company is calibrating some measuring instrumentation and has measured the radius and height of one cylinder 10 separate times; they are in vector variables *r* and *h*. Find the volume from each trial, which is given by $\Pi r^2 h$. Also use logical indexing first to make sure that all measurements were valid (> 0).

```
>> r = [5.501  5.5  5.499 5.498 5.5 5.5 5.52 5.51 5.5 5.48];
>> h = [11.11 11.1 11.1 11.12 11.09 11.11 11.11 11.1 11.08 11.11];
>> all(r>0 & h>0)
ans =
     1
>> vol = pi * r.^2 .* h
```

42) For the following matrices A, B, and C:

$$A = \begin{bmatrix} 1 & 4 \\ 3 & 2 \end{bmatrix} \quad B = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 5 & 6 \\ 3 & 6 & 0 \end{bmatrix} \quad C = \begin{bmatrix} 3 & 2 & 5 \\ 4 & 1 & 2 \end{bmatrix}$$

- Give the result of 3*A.

$$\begin{bmatrix} 3 & 12 \\ 9 & 6 \end{bmatrix}$$

- Give the result of A*C.

$$\begin{bmatrix} 19 & 6 & 13 \\ 17 & 8 & 19 \end{bmatrix}$$

- Are there any other matrix multiplications that can be performed? If so, list them.

C*B

43) For the following vectors and matrices A, B, and C:

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

Perform the following operations, if possible. If not, just say it can't be done!

```
A * B

     No, inner dimensions do not agree

B * C

     14

C * B
```

$$\begin{bmatrix} 2 & 8 \\ 3 & 12 \end{bmatrix}$$

44) The matrix variable *rainmat* stores the total rainfall in inches for some districts for the years 2010-2013. Each row has the rainfall amounts for a given district. For example, if *rainmat* has the value:

```
>> rainmat
ans =
  25   33   29   42
  53   44   40   56
 etc.
```

district 1 had 25 inches in 2010, 33 in 2011, etc. Write expression(s) that will find the number of the district that had the highest total rainfall for the entire four year period.

```
>> rainmat = [25 33 29 42; 53 44 40 56];
>> large = max(max(rainmat))
large =
    56
>> linind = find(rainmat== large)
linind =
     8
>> floor(linind/4)
ans =
     2
```

45) Generate a vector of 20 random integers, each in the range from 50 to 100. Create a variable *evens* that stores all of the even numbers from the vector, and a variable *odds* that stores the odd numbers.

```
>> nums = randi([50, 100], 1, 20);
>> evens = nums(rem(nums,2)==0);
>> odds = nums(rem(nums,2)~=0);
```

46) Assume that the function **diff** does not exist. Write your own expression(s) to accomplish the same thing for a vector.

```
>> vec = [5 11 2 33 -4]
vec =
     5    11     2    33    -4
>> v1 = vec(2:end);
>> v2 = vec(1:end-1);
>> v1-v2
ans =
     6    -9    31   -37
```

47) Create a vector variable *vec*; it can have any length. Then, write assignment statements that would store the first half of the vector in one variable and the second half in another. Make sure that your assignment statements are general, and work whether *vec* has an even or odd number of elements (Hint: use a rounding function such as **fix**).

```
   >> vec = 1:9;
   >> fhalf = vec(1:fix(length(vec)/2))
   fhalf =
        1     2     3     4
   >> shalf = vec(fix(length(vec)/2)+1:end)
   shalf =
```

MATLAB: A Practical Introduction to Programming and Problem Solving

Fourth Edition

PRACTICE PROBLEM SOLUTIONS

Stormy Attaway

College of Engineering
Boston University

Chapter 1

Practice 1.1

Think about what the results would be for the following expressions, and then type them in to verify your answers:

```
>> 1\2
ans =
     2
>> - 5 ^ 2
ans =
   -25
>> (-5) ^ 2
ans =
    25
>> 10-6/2
ans =
     7
>> 5*4/2*3
ans =
    30
```

Practice 1.2

Generate a random

- real number in the range (0,1)

  ```
  rand
  ```

- real number in the range (0, 100)

  ```
  rand*100
  ```

- real number in the range (20, 35)

  ```
  rand*(35-20)+20
  ```

- integer in the inclusive range from 1 to 100

  ```
  randi(100)
  ```

- integer in the inclusive range from 20 to 35

  ```
  randi([20, 35])
  ```

Practice 1.3

Think about what would be produced by the following expressions, and then type them in to verify your answers.

```
>> 3 == 5 + 2
ans =
     0

>> 'b' < 'a' + 1
ans =
     0

>> 10 > 5 + 2
ans =
     1

>> (10 > 5) + 2
ans =
     3

>> 'c' == 'd' - 1 && 2 < 4
ans =
     1

>> 'c' == 'd' - 1 || 2 > 4
ans =
     1

>>xor('c' == 'd' - 1, 2 > 4)
ans =
     1

>>xor('c' == 'd' - 1, 2 < 4)
ans =
     0

>>10 > 5 > 2
ans =
     0
```

Practice 1.4

- Calculate the range of integers that can be stored in the types **int16**and **uint16**. Use **intmin** and **intmax** to verify your results.

```
>> 2^16
ans =
       65536
>> 2^15
ans =
       32768
>>intmin('int16')
ans =
 -32768
>>intmax('int16')
ans =
  32767
>>intmin('uint16')
ans =
      0
>>intmax('uint16')
ans =
 65535
```

- Enter an assignment statement and view the type of the variable in the Workspace Window. Then, change its type and view it again. View it also using **whos**.

```
>>clear
>>mynumber = 3*11;
>>whos
  Name           Size              Bytes  Class     Attributes

mynumber      1x1                    8  double

>>mynumber = int32(mynumber)
mynumber =
          33
>>whos
  Name           Size              Bytes  Class    Attributes

mynumber      1x1                    4  int32
```

Practice 1.5

- Find the numerical equivalent of the character 'x'.

- Find the character equivalent of 107.

```
>>double('x')
ans =
  120
>>char(107)
ans =
k
```

Practice 1.6

Use the **help** function to find out what the rounding functions **fix**, **floor**, **ceil**, and **round** do. Experiment with them by passing different values to the functions, including some negative, some positive, some with fractions less than 0.5 and some greater. *It is very important when testing functions that you thoroughly test by trying different kinds of arguments!*

Chapter 2

Practice 2.1

Think about what would be produced by the following sequence of statements and expressions, and then type them in to verify your answers:

```
pvec = 3:2:10

pvec(2) = 15

pvec(7) = 33

pvec([2:4  7])

linspace(5,11,3)

logspace(2,4,3)


>>pvec = 3:2:10
pvec =
     3     5     7     9
>>pvec(2) = 15
pvec =
     3    15     7     9
>>pvec(7) = 33
pvec =
     3    15     7     9     0     0    33
>>pvec([2:4 7])
ans =
    15     7     9    33
>>linspace(5,11,3)
ans =
     5     8    11
>>logspace(2, 4, 3)
ans =
         100        1000       10000
```

Practice 2.2

Think about what would be produced by the following sequence of statements and expressions, and then type them in to verify your answers.

```
  mat = [1:3; 44 9  2; 5:-1:3]

  mat(3,2)

  mat(2,:)

  size(mat)

  mat(:,4) = [8;11;33]

  numel(mat)

  v = mat(3,:)

  v(v(2))

  v(1) = []

  reshape(mat,2,6)
```

```
>>mat = [1:3; 44 9  2; 5:-1:3]
mat =
     1     2     3
    44     9     2
     5     4     3
>>mat(3,2)
ans =
     4
>>mat(2,:)
ans =
    44     9     2
>>size(mat)
ans =
     3     3
>>mat(:,4) = [8;11;33]
mat =
     1     2     3     8
    44     9     2    11
```

```
     5     4     3    33
>>numel(mat)
ans =
    12
>>v = mat(3,:)
v =
     5     4     3    33
>>v(v(2))
ans =
    33
>>v(1) = []
v =
     4     3    33
>>reshape(mat,2,6)
ans =
     1     5     9     3     3    11
    44     2     4     2     8    33
```

Practice 2.3

Create a vector variable and subtract 3 from every element in it.
Create a matrix variable and divide every element by 3.
Create a matrix variable and square every element.

```
>>vec = [4 11 32 -5 0  9]
vec =
     4    11    32    -5     0     9
>>vec - 3
ans =
     1     8    29    -8    -3     6
>>
>>mat = randi(30,2,4)
mat =
    24     4    19     8
    27    27     3    16
>>mat/3
ans =
    8.0000    1.3333    6.3333    2.6667
    9.0000    9.0000    1.0000    5.3333
>>mat .^ 2
ans =
   576    16   361    64
   729   729     9   256
```

Practice 2.4

Modify the result seen in the previous Quick Question. Instead of deleting the “bad” elements, retain only the “good” ones. (Hint: Do it two ways, using **find** and using a logical vector with the expression *vec >= 0*).

```
>>vec = [11 -5 33 2 8 -4 25]
vec =
    11    -5    33     2     8    -4    25
>>pos = find(vec >= 0)
pos =
     1     3     4     5     7
>>res = vec(pos)
res =
    11    33     2     8    25

>>vec(vec>=0)
ans =
    11    33     2     8    25
```

Practice 2.5

When two matrices have the same dimensions and are square, both array and matrix multiplication can be performed on them. For the following two matrices, perform A.*B, A*B, and B*A by hand and then verify the results in MATLAB.

$$\overset{\text{A}}{\begin{bmatrix} 1 & 4 \\ 3 & 3 \end{bmatrix}}\overset{\text{B}}{\begin{bmatrix} 1 & 2 \\ -1 & 0 \end{bmatrix}}$$

```
>>A .* B
ans =
     1     8
    -3     0

>> A * B
ans =
    -3     2
     0     6

>> B * A
ans =
     7    10
    -1    -4
```

Chapter 3

Practice 3.1