MATLAB: A Practical Introduction to Programming and Problem Solving

Fourth Edition

SOLUTION MANUAL

Stormy Attaway

College of Engineering
Boston University

```
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)
```

- 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, but differ by y if x and y have different signs.

>> help mod
 mod    Modulus after division.
```

```
     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 =
```

timeoption.m

```
function choice = timeoption
% Print the menu of options and error-check
%  until the user pushes one of the buttons
% Format of call: timeoption or timeoption()
% Returns the integer value of the choice, 1-3

choice = menu('Choose a unit', 'Minutes', ...
    'Hours', 'Exit');
% If the user closes the menu box rather than
%  pushing one of the buttons, choice will be 0
while choice == 0
    disp('Error - please choose one of the options.')
    choice = menu('Choose a unit', 'Minutes', ...
    'Hours', 'Exit');
end
end
```

secsToMins.m

```
function mins = secsToMins(seconds)
% Converts a time from seconds to minutes
% Format secsToMins(seconds)
% Returns the time in minutes

mins = seconds / 60;
end
```

secsToHours.m

```
function hours = secsToHours(seconds)
% Converts a time from seconds to hours
% Format secsToHours(seconds)
% Returns the time in hours

hours = seconds / 3600;
end
```

28) Write a menu-driven program to investigate the constant π. Model it after the program that explores the constant *e*. Pi (π) is the ratio of a circle's circumference to its diameter. Many mathematicians have found ways to approximate π. For example, Machin's formula is:

$$\frac{\pi}{4} = 4 \ \text{arctan}\left(\frac{1}{5}\right) - \text{arctan}\left(\frac{1}{239}\right)$$

Leibniz found that π can be approximated by:

18) Create a data file to store information on hurricanes. Each line in the file should have the name of the hurricane, its speed in miles per hour, and the diameter of its eye in miles. Then, write a script to read this information from the file and create a vector of structures to store it. Print the name and area of the eye for each hurricane.

Ch9Ex18.m

```
% Reads hurricane information and store in vector
% of structures, print name and area for each

fid = fopen('hurricane.dat');
if fid == -1
    disp('File open not successful')
else
   i = 0;
   while feof(fid) == 0
       i = i + 1;
       aline = fgetl(fid);
       [hname, rest] = strtok(aline);
       [speed, diam] = strtok(rest);
       hstruc = struct('Name', hname, 'Speed', ...
           str2num(speed), 'Diam', str2num(diam));
       hurricane(i) = hstruc;
   end
   for i = 1:length(hurricane)
       fprintf('%s had area %.2f\n', hurricane(i).Name, ...
           pi * (hurricane(i).Diam/2)^2)
   end

   closeresult = fclose(fid);
   if closeresult == 0
       disp('File close successful')
   else
       disp('File close not successful')
   end
end
```

19) Create a file “parts_inv.dat” that stores on each line a part number, cost, and quantity in inventory, in the following format:

```
123 5.99 52
```

Use **fscanf** to read this information, and print the total dollar amount of inventory (the sum of the cost multiplied by the quantity for each part).

Ch9Ex19.m

```
% Read in parts inventory information from file
```

13) When an object with an initial temperature T is placed in a substance that has a temperature S, according to Newton's law of cooling in t minutes it will reach a temperature T_t using the formula $T_t = S + (T - S)\ e^{(-kt)}$ where k is a constant value that depends on properties of the object. For an initial temperature of 100 and k = 0.6, graphically display the resulting temperatures from 1 to 10 minutes for two different surrounding temperatures: 50 and 20. Use the **plot** function to plot two different lines for these surrounding temperatures, and store the handle in a variable. Note that two function handles are actually returned and stored in a vector. Change the line width of one of the lines.

Ch12Ex13.m
```
% Plot the cooling temperatures of an object placed in
% two different surrounding temperatures

time = linspace(1,10,100);
T1 = 50 + (100 - 50)*exp(-0.6*time);
T2 = 20 + (100 - 20)*exp(-0.6*time);
hdl = plot(time,T1,'b-',time,T2,'r-');
xlabel('Time')
ylabel('Temperature')
legend('50 degrees', '20 degrees')
set(hdl(1),'LineWidth',3)
```

14) Write a script that will draw the line y=x between x=2 and x=5, with a random line width between 1 and 10.

Ch12Ex14.m
```
% Plot line y = x with a random thickness

x = [2 5];
y = x;
hdl = plot(x,y);
title('Line with random thickness')
set(hdl,'LineWidth',randi([1 10]))
```

15) Write a script that will plot the data points from y and z data vectors, and store the handles of the two plots in variables *yhand* and *zhand*. Set the line widths to 3 and 4 respectively. Set the colors and markers to random values (create strings containing possible values and pick a random index).

Ch12Ex15.m
```
y = [33 22 17 32 11];
z = [3 7 2 9 4 6 2 3];
```