1. Problems from Gilat, Ch. 1.10

Open a terminal window, change to your home directory and create the directory `octave`. Change to that directory and start `octave`.

```
% cd
% mkdir octave
% cd octave
% octave
```

Open another terminal window, change to directory `~/octave`, and using your text editor, create the file `probs1.m` that contains `octave/MATLAB` commands to perform calculations as enumerated below.

As you type the commands to answer each problem, execute the commands (in the entire file) by typing `probs1` at the `octave` prompt:

```
octave:1> probs1
```

Note: The output from `octave` is piped through `more` as necessary (i.e. if the output will not fit within your terminal window). The `more` prompt in this case is a colon at the bottom left edge of the screen: as usual type 'space' to advance, 'b' to back up, and 'q' to quit.

1.2 a) Calculate

$$23\left(-8+\frac{\sqrt{607}}{3}\right)+\left(\frac{40}{8}+4.7^2\right)^2$$

1.4 a) Calculate

$$\cos\left(\frac{5\pi}{6}\right)\sin^2\left(\frac{7\pi}{8}\right)+\frac{\tan\left(\frac{\pi}{6}\ln 8\right)}{\sqrt{7}+2}$$

1.6 a) Define the variables x and z as $x = 5.3$, and $z = 7.8$, then evaluate:

$$\frac{xz}{(x/z)^2}+14x^2-0.8z^2$$

1.10 a) The following is a trignonometric identity:

$$\sin(3x) = 3\sin x - 4\sin^3 x$$

Verify that the identity is correct by calculating each side of the equation, substituting $x = 7\pi/20$.

1.16) The distance d from a point (x_0, y_0) to a line $Ax + By + C = 0$ is given by:

$$d = \frac{|Ax_0 + By_0 + C|}{\sqrt{A^2+B^2}}$$

Determine the distance of the point $(-3, 4)$ from the line $2x - 7y - 10 = 0$. First define the variables A, B, C, x_0 and y_0, and then calculate d. (Use the `abs` and `sqrt` functions).

2. Problems from Gilat, Ch. 2.11

Again, working in your $\sim$ /octave directory, and using your text editor, create the file `probs2.m` that contains `octave/MATLAB` commands to perform calculations as enumerated below.

Once more, as you type the commands to answer each problem, execute the commands (in the entire file) by typing `probs2` at the `octave` prompt:

```
octave:1> probs2
```

2.1 Create a row vector that has the elements 6, $8 \cdot 3$, 81, $e^{2.5}$, $\sqrt{65}$, $\sin(\pi/3)$ and 23.05.

2.2 Create a column vector that has the elements 44, 9, $\ln(51)$, 2^3, 0.1 and $5\tan(25°)$.

2.4 Create a column vector in which the first element is 18, the elements decrease with increments of -4, and the last element is -22. (Recall that a column vector can be created by the transpose of a row vector.)

2.8 Create a vector, name it `Afirst`, that has 13 elements in which the first is 3, the increment is 4 and the last element is 51. Then, using the colon symbol, create a new vector, call it `Asecond`, that has seven elements. The first four elements are the the first four elements of the vector `Afirst`, and the last three are the last three elements of the vector `Afirst`.

2.9 Create the matrix shown below by using the vector notation for creating vectors with constant spacing and/or the `linspace` command when entering the rows.

$$B = \begin{bmatrix} 0 & 4 & 8 & 12 & 16 & 20 & 24 & 28 \\ 69 & 68 & 67 & 66 & 65 & 64 & 63 & 62 \\ 1.4 & 1.1 & 0.8 & 0.5 & 0.2 & -0.1 & -0.4 & -0.7 \end{bmatrix}$$

2.10 Using the colon symbol, create a 3×5 matrix (assign to a variable named `msame`) in which all of the elements are the number 7.

2.14 Create the following matrix, A:

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

Usee the matrix A to:

a) Create a five-element row vector named `va` that contains the elements of the first row of A.

b) Create a three-element row vector named `vb` that contains the elements of the third column of A.

c) Create an eight-element row vector names `vc` that contains the elements of the second row of A and the fourth column of A.

d) Create a six-element row vector named `vd` that contains the elements of the first and fifth columns of A.

2.18 Using the `zeros`, `ones` and `eye` commands, create the following arrays:

a)

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

b)

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

c)

$$\begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$

3. Writing simple `octave/MATLAB` functions and scripts

All of the following functions and scripts should be prepared in `.m` files within your `~/octave` directory.

3a) `hello`:

Write a `MATLAB` function as follows and save in a file called `hello.m`.

```
function [] = hello()
   printf('Hello world!\n');
end
```

Execute the function within `octave` by typing

```
octave> hello
```

and you should see the output

```
Hello world!
```

If you don't see output as above, then ensure that

1. You have typed the definition of `hello` precisely as given above.
2. You have saved the definition in the file `~/octave/hello.m` and that you are running `octave` in the directory `~/octave`.

3b) `threeoutargs`:

Create a `MATLAB` function **`threeoutargs`** which has two input arguments, x and y, and which returns *three* output arguments which are $x + y$, $x - y$ and $(x + y)/2$, respectively. Ensure that you save the definition of your function as the file **`threeoutargs.m`**.

To check your implementation of **`threeoutargs`**, create a `MATLAB` script in the file **`t_threeoutargs.m`**, with contents as follows

```
[a b c] = threeoutargs(1.0, 6.0)
[val1 val2 val3] = threeoutargs(-1.0, pi)
```

Execute the script as follows

```
octave> t_threeoutargs
```

and ensure that you get the output

```
a =  7
b = -5
c =  3.5000
val1 =  2.1416
val2 = -4.1416
val3 =  1.0708
```

3c) `sintaylor` (moderately challenging!):

The Taylor series expansion for $\sin x$ is given by

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots$$

Create a `MATLAB` function `sintaylor` with a header as follows

```
function res = sintaylor(x,nmax,epsi)
```

and which computes an approximation of $\sin(x)$ using the following truncated version of the series

$$\sin x = \sum_{n=0}^{n_{\max}} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

`sintaylor` should return as soon as either one of the conditions have been met

- All of the terms in the truncated series have been evaluated.
- An individual term in the series has an absolute value that is $\leq$ `epsi` (but include that term in the sum)

Save your code in the file `sintaylor.m`

Write a test script `t_sintaylor` in the file `t_sintaylor.m` with the contents

```
format long
epsi = 1.0e-12
for x = [0.1 0.5]
   x
   exact = sin(x)
   for nmax = 1:10
      nmax
      approx = sintaylor(x,nmax,epsi)
   end
end
```

Execute the script using

```
octave> t_sintaylor
```

and verify that you get output as follows

```
epsi =  1.00000000000000e-12
x =  0.100000000000000
exact =  0.0998334166468282
nmax =  1
approx =  0.0998333333333333
nmax =  2
approx =  0.0998334166666667
nmax =  3
approx =  0.0998334166468254
nmax =  4
approx =  0.0998334166468282
nmax =  5
approx =  0.0998334166468282
```

```
nmax =  6
approx =  0.0998334166468282
nmax =  7
approx =  0.0998334166468282
nmax =  8
approx =  0.0998334166468282
nmax =  9
approx =  0.0998334166468282
nmax =  10
approx =  0.0998334166468282
x =  0.500000000000000
exact =  0.479425538604203
nmax =  1
approx =  0.479166666666667
nmax =  2
approx =  0.479427083333333
nmax =  3
approx =  0.479425533234127
nmax =  4
approx =  0.479425538616416
nmax =  5
approx =  0.479425538604183
nmax =  6
approx =  0.479425538604203
nmax =  7
approx =  0.479425538604203
nmax =  8
approx =  0.479425538604203
nmax =  9
approx =  0.479425538604203
nmax =  10
approx =  0.479425538604203
```