Simple Programming in MATLAB

Plotting Graphs:

We will plot the graph of the function

$$y = f(x) = e^{-1.5x} \sin(8\pi x), \quad 0 \le x \le 1$$

Plotting a graph using MATLAB involves three steps:

- Create points $0 = x_1 < x_2 < \cdots < x_n = 1$.
- Compute $y_i = f(x_i)$, $i = 1, 2, \cdots, n$.
- Draw a polygonal line that connects the points $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$.

(1) Create x-values: We first create a vector $x = (x_1, x_2, \cdots, x_n)$ for $n = 5$. It could be done in several ways:

(a) We can create the vector manually:

```
>> x = [0 .25 .5 .75 1] <ret>

 x =

     0 0.2500 0.5000 0.7500 1.0000
```

Here `x(1) = 0, x(2) = .25, x(3) = .5, x(4) = .75` and x(5) = 1.

(b) We may also use the "do - loop".

```
>> n = 5;      (ret)
>> h = 1/(n-1);      (ret)
>> for k=1:n    (shift-ret)
     x(k)=(k-1)*h;     (shift-ret)
   end    (shift-ret)
>> x    (ret)

 x =

     0 0.2500 0.5000 0.7500 1.0000
```

(c) We may also use MATLAB's `linspace` command:

```
>> x = linspace(0,1,5)    (ret)

 x =

     0 0.2500 0.5000 0.7500 1.0000
```

Thus we have created a row vector x of size n ($n = 5$ in this case).

(2) Compute y-values:

```
>> y = exp(-1.5*x) .* sin(8*pi*x)
```

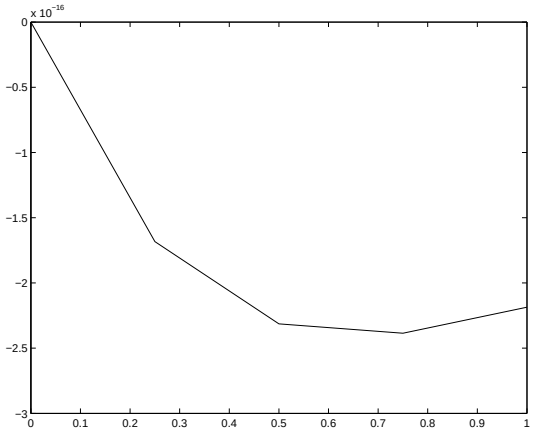
y is a vector, where $y_i = e^{-1.5x_i} \sin(8\pi x_i)$. This step requires some explanation. Here `x = [x(1) x(2) ⋯ x(5)]` is a row-vector, i.e., it is a matrix of size 1×5.

- `-1.5*x` is a vector `[-1.5*x(1), -1.5x(2), ⋯, -1.5x(5)]`.
- `pi` is the usual π in MATLAB.
- `sin(8*pi*x)` is a vector `[sin(8*pi*x(1)), sin(8*pi*x(2)), ⋯, sin(8*pi*x(5))]`
- The operation "`.*`" is the *vector multiplication* operation, which is "component-wise" multiplication. If $a = (a_1, a_2, \cdots, a_n)$ and $b = (b_1, b_2, \cdots, b_n)$. Then `a .* b` is the vector $(a_1b_1, a_2b_2, \cdots, a_nb_n)$. Other standard vector operations are similar, e.g., `a./b` is the vector $(a_1/b_1, a_2/b_2, \cdots, a_n/b_n)$.
- Based on the discussion, it is clear that `y` is the vector $(y_1, y_2, \cdots, y_5)$ where $y_i = f(x_i)$.

(3) Plotting the points `( x(i), y(i) )`:

```
>> plot(x,y)
```

This command will produce a graph.



The graph is disappointing. The reason for this strange graph is that we we took a low value of n. We can take a higher value of n, and repeat all the commands, which will require a lot of typing. To avoid such situations, we will *create* and *execute* a MATLAB *script* file, in the next section.

Writing MATLAB scripts

A script file is an ASCII file, created by the user, which contains a sequence of MATLAB commands. This file must be saved with an extention ".m". They are also referred to as *M files*. By typing the name of the script file (without the '.m' extension) at the command-prompt (`>>`), we execute the script, i.e., execute all the commands sequentially.

Current Directory: The files created in MATLAB are usually stored in the *current directory* (folder) of MATLAB. The current directory is displayed in a window just above the command window. The *default* current directory of MATLAB may not be the best place to save a file for proper organization. For example, we may create a new directory, called "MatlabTutorial", make MatlabTutorial as the current directory, and save files in that directory. One can change the current directory to MatlabTutorial by navigating through the *browse button*, located next to the current directory display. A script file may be executed if it is in the current directory (by changing the MATLAB path, one can access also other directories).

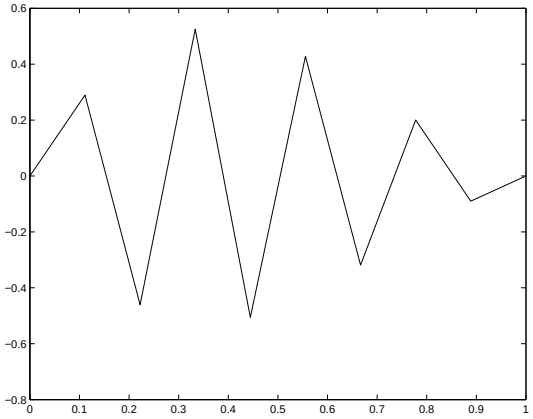
Creating script file: Select File - New - M-File from the File menu of MATLAB window. An edit window will appear. Type the appropriate MATLAB commands in the file. In our case, we type the following:

```
% Script 1; Script to plot a function
n=10;
x=linspace(0,1,n);
y=exp(-1.5*x).*sin(8*pi*x);
plot(x,y)
```

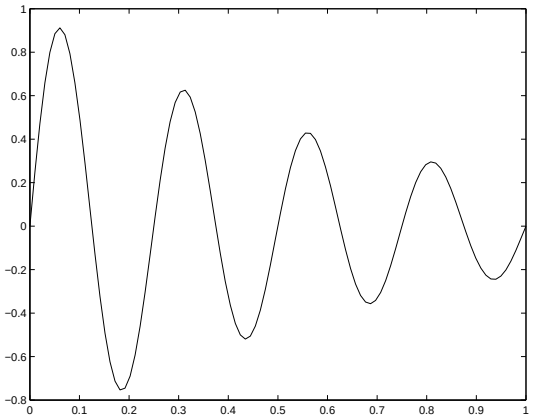
Now save this script file by selecting File - Save As... frome the File menu of the edit window. For example, save it as "sc1.m". Note that we have used `n=10`, and have suppressed the output of each command by putting a '`;`' at the end of the commands. To execute this script file, we type at the command prompt

```
>> sc1
```  (ret)

We get the graph



The graph is still not good. We have to increase the value of n. If you have closed the edit window, open the file `sc1.m` by selecting File - Open... and the file `sc1.m`. Change `n=100`, save the file, and execute again. We get the following graph.



This looks realistic.

If we want to change the function, we can also do that by editing the same script file.

Creating Function files

A function file is just like a script file, but it has well-defined input and output variables. It is also an *M-File*. These functions can be used in other scripts. For example, we used the function

$$f(x) = e^{-1.5x} \sin(8\pi x)$$

in the script `sc1.m`, and typed it explicitly. We can define this function in a *function file* as follows: First open an edit window and then type

```
function y = func1(x);
y = exp(-1.5*x) .* sin(8*pi*x);
```

Save this small file as 'func1.m' (use the same name as the name of the function while saving). Note that the first line of this *function file* is the *function definition line*, which clearly indicates the name of the function and the input/output variables. In this function, `x` is the input variable which could be either a scalar or vector. The output variable is `y`; it is scalar if `x` is scalar, and it is a vector if `x` is a vector. The *size* of `y` is same as the size of `x`.

The function `func1.m` can be used in the script `sc1.m` by slightly changing the script as follows:

```
% Script sc2.m; Script to plot a function
% We plot a function in the interval [a,b]
a = 0;
b = 1;
n=100;
x=linspace(a,b,n);
y=func1(x);
plot(x,y)
```

Save the script as `sc2.m`. Note that we have used `y=func1(x);`. *Make sure that* `func1.m` *and* `sc2.m` *are both in the current directory*. Also note that we can assign other values to the variables `a` and `b` (other that 0 and 1 respectively).

In this particular case, we really did not need to define the function `func1.m`, as it was a simple function. But functions, used in MATLAB, can be complicated and it is useful to define functions separately.

Taylor Polynomials of e^x:

We will write a function that is the n^{th}-degree Taylor polynomial of the function $f(x) = e^x$. It is well known that this polynomial is given by

$$\sum_{k=0}^{n} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$$

The function is defined as follows:

```
function y = TaylPol(x,n);
l = size(x);
y = ones(l);
term = ones(l);
for k=1:n
   term = (term.*x)/k;
   y = y + term;
end
```

Some comments of this function:

- TaylPol is a function of two variables, `x` and `n`. Here `n` is the degree of the Taylor polynomial. `y` is the value of the Taylor polynomial at `x`; `x` could be a scalar or a vector and `y` is a vector of same length.
- The command `size(x)` gives the *size* of `x` when `x` is considered as a matrix. In this case, if `x` is a row-vector with 10 components, then `size(x) = [ 1 10]`. If `x` is a scalar (a row-vector with one component), then `size(x) = [1 1]`.
- The command `ones(l)` is a row vector of size `l` whose components are 1.

We now write a script which plots e^x and its Taylor polynomials of degree 1, 3, and 5 for $0 \leq x \leq 4$.

```
% Script sc3.m
% Plot exp(x) and its Taylor polynomials of
% degree 1, 3, and 5 for 0 <= x <= 4
n=400;
x=linspace(0,4,n);
y1=exp(x);
y2 = TaylPol(x,1);
y3 = TaylPol(x,3);
y4 = TaylPol(x,5);
plot(x,y1,'k-',x,y2,'k--',x,y3,'k-.',x,y4,'k:')
```

Comments:

- `x` is a vector of length 400 (note that `size(x) = [1 400]`). The components are uniformly distributed values of x between 0 and 400 (including 0 and 400).

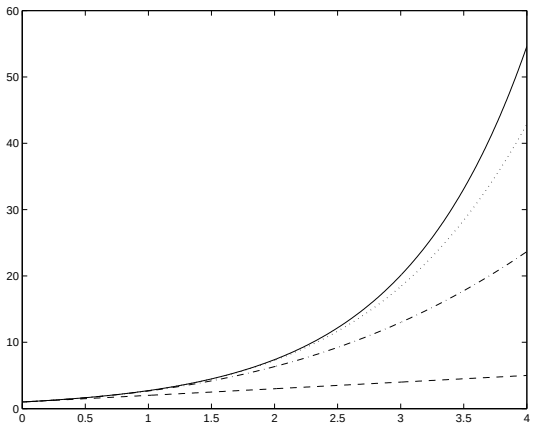
- `y1` is the row-vector of values of e^x, where x is the row-vector of length 400. `y2, y3` and `y4` are row-vectors of values of the Taylor polynomials of degree 1, 3, and 4 respectively.

- The plot command plots four plots on the same graph. It first plots `(x,y1)` with `'k-'` option where `k` represents black color, and `-` represents solid line. It then plots `(x,y2)` with `'k--'` option, i.e., black dashed line. The option `'k-.'` means black dash-dot line, and `'k:'` means black dotted line.

The script `sc3.m` is then executed by typing

```
>> sc3
```

to get



Values of $\sin x$ via its Maclaurin series:

We now present the last function of this section which evaluates $\sin x$ via its Maclaurin series;

$$\sin x = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{x^{2k-1}}{(2k-1)!}$$

This series converges for every value of x. We will find $\sin x$, for a given x, by "computing the infinite series" in the following sense: We will compute the partial sum

$$S_n = \sum_{k=1}^{n} (-1)^{k+1} \frac{x^{2k-1}}{(2k-1)!},$$

for large enough n, such that

$$|S_{n+1} - S_n| \text{ is small.}$$

Also the following observation will be useful in writing a MATLAB function for for this procedure. The $(k-1)^{th}$ term of the infinite series is

$$a_{k-1} = (-1)^k \frac{x^{2k-3}}{(2k-3)!}.$$

And thus, thus the k^{th} term can be written as

$$a_k = (-1)^{k+1} \frac{x^{2k-1}}{(2k-1)!} = a_{k-1} \left[(-1) \frac{x^2}{(2k-1)(2k-2)} \right]$$

We are now ready to write the function.

```
function y = sinmac(x)
% x could be a vector
ep = 10^(-14);
l=length(x);
for k=1:l
   sl = 0;
   sn = x(k);
   term = x(k);
   con = 1;
   pm = 1;
   while (abs(sn-sl)>ep)
      sl = sn;
      con = con+2;
      pm = -pm;
      term = (term.*(x(k)^2))/(con*(con-1));
      sn = sn + term*pm;
   end
   y(k) = sn;
end
```

We use the following script to use this function.

```
x = [.5 pi/4 1.2];
sinmac(x)
```

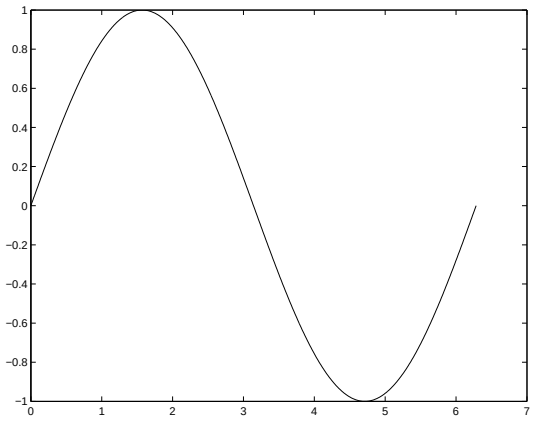
The output will be

```
ans =
   0.47942553860420 0.70710678118655 0.93203908596723
```

We can also use the script sc2.m with slight modification.

```
% Script sc2.m; Script to plot a function
% We plot a function in the interval [a,b]
a = 0;
b = 2*pi;
n=500;
x=linspace(a,b,n);
y=sinmac(x);
plot(x,y)
```

We get the graph



Caution: We will later see certain problems with this function.