Function m-files:

1. Create a function m-file, **cho.m** to represent coupled harmonic oscillations:

```
% cho.m
% coupled harmonic oscillator
function z = cho(x)
z = cos(8*x) + cos(9*x);
```

At the MATLAB prompt, type

```
>> cho(pi/2)
>> cho(pi/3)
>> 7 + cho(2^(1/5))/cho(3)
```

It works in the same way as other MATLAB built-in functions (**cos, exp, log** etc.) You can even plot the function, e.g.

```
>> t = linspace(0 , 6*pi , 400);
>> plot(t, cho(t)), grid
```

2. Write a function m-file **myf.m** that returns the values of the function

$$w = y^2\, sin(10y) + e^{-y}\, cos(y).$$

At the MATLAB prompt, plot this function on a grid, over the range $0 \leq y \leq 3\pi$ using 360 points. Using a single plot command, plot on the same diagram (over the range $0 \leq w \leq 3\pi$ using 360 points) the graphs of $x = \text{myf}(w)$, $y = w^2$ and $z = -w^2$. *Note the use (and re-use) of dummy variables!*

3. **multiple input / output** Write a function m-file **Q3f.m** which inputs two integers **nc** & **nr** and returns the following three outputs:

   - a row vector of random numbers {each in the interval $[0, 1)$} with **nc** elements,
   - a column vector of random numbers {each in the interval $[0, 1)$} with **nr** elements,
   - an **nr** $\times$ **nc** matrix each of whose elements is a random number in the interval $[0, 1)$.

   Use the variable names **r, c & M** for the row vector, column vector and matrix respectively. On the command window try the following:

```
>> Q3f(7, 9)
>> r = Q3f(7, 9)
>> c = Q3f(7, 9)
>> M = Q3f(7, 9)
>> [r c M] = Q3f(7, 9)
>> [M, r, c] = Q3f(7, 9)
```

Script files calling function files

4. Write a function file **Q4f.m** with two inputs (t and M) and three outputs (N, V and P) where:

$$V(t) = 1 + \sum_{n=1}^{M} (-1)^{n-1} \frac{4}{(2n-1)\pi} \cos\left(\frac{(2n-1)\pi}{2} t\right),$$

$$P(t) = \pi + \sum_{k=1}^{N} \frac{(-1)^{k-1}}{\exp(k/5)} \sin(k\,t).$$

Here **N** is the smallest integer such that the addition of any extra term to the sum {in $P(t)$} will not change the MATLAB value of the sum (i.e. the smallest number of terms after which the sum becomes constant). *Use a single **for** loop to compute V and a single **while** loop to compute P. Absolutely NO **if** statements should appear in **Q4f.m**!* For $V(t)$ possible choices of M are: $100, 200, 300, 400$.

Write a script file **Q4s.m** which:

- creates the vector t with 500 elements starting at $t = 0$ and ending at $t = 25$,
- inputs (using the input command) the value of M,
- calls the function **Q4f.m**; uses subplot commands to plot the graphs of V versus t (top graph) and that of P versus t (bottom graph). For each element in the time vector, display the time *(1 decimal place, fixed point)* and number of terms (N) using **fprintf** commands as follows:
  At time ...., P requires .... terms to converge.

5. Write a function file **Q5f.m** with a single input x and two outputs y_1 and y_2 where

$$y_1 = x \qquad \text{and} \qquad y_2 = \frac{1}{2}\, x\,(4 - x)\, \sin\left(\pi \sqrt{\frac{x}{2}}\right).$$

Write a script file **Q5s.m** which inputs a positive integer N and computes the iterates $x_0, x_1, x_2 \ldots x_N$, where

$$x_0 = 0.1,$$

$$x_{i+1} = \frac{1}{2}\, x_i\,(4 - x_i)\, \sin\left(\pi \sqrt{\frac{x_i}{2}}\right), \qquad \text{for} \qquad 0 \le i \le N-1,$$

and then plots (on the same diagram) using **hold on** and **hold off** commands (as well as the function **Q5f.m**):

- the line y_1, in the interval $0 \le x \le 1.5$ using 100 points for both vectors in the plot,
- the curve y_2, also in $0 \le x \le 1.5$ and using 100 points for both vectors in the plot,
- the series of horizontal lines joining the points $(x_i, f(x_i))$ and $(x_{i+1}, f(x_i))$,
- the series of vertical lines joining the points $(x_{i+1}, f(x_i))$, $(x_{i+1}, f(x_{i+1}))$.

Test your code with several values of N (say $N = 5, 8, 15, 20, 50, 100$) and each time display the last element in the vector x using 8 decimal places (fixed point) via the **fprintf** command.

fprintf:

6. If the number **doesn't fit** the format string you specified, MATLAB will expand the field width so it can print out a sensible number. Type the following at the prompt, using ↑ to pull down previous commands for editing.

   Of course you would *normally* put spaces in your messages next to the numbers, but examples below show you exactly how much space the numbers take up. Enter each of the following and check if you agree with the output: Note that in MATLAB, 093 is the same as 93, i.e. leading zeros do not really matter!

   - MATLAB will not give you less than the decimal places you specify! Notice the short width specification:

     ```
     >> fprintf('Left->%7.5f<-Right \n', 01234.56789)
     ```

   - Again, try specifying a much shorter width than expected:

     ```
     >> fprintf('Left->%7.2f<-Right \n', -0123456789)
     ```

   - You can't force MATLAB tu put **no** numbers in front of the decimal point:

     ```
     >> fprintf('Left->%7.6f<-Right \n', .123456789)
     ```

   - Try specifying more decimal places than the width:

     ```
     >> fprintf('Left->%7.8f<-Right \n', 0.123456789)
     ```

   - MATLAB never gives you less than the width you asked for, filling up with spaces in front:

     ```
     >> fprintf('Left->%12.8f<-Right \n', 0.123456789)
     >> fprintf('Left->%22.1e<-Right \n', 0123456789)
     >> fprintf('Left->%6.2f<-Right \n', 0.001)
     >> fprintf('Left->%6.2f<-Right \n', 1)
     ```

   - You can include leading zeros by typing 0 in front of the width:

     ```
     >> fprintf('Left->%06.2f<-Right \n', 1)
     ```

   - Specifying zero decimal places:

     ```
     >> fprintf('Left->%6.0f<-Right \n', 0.123456789)
     >> fprintf('Left->%6.0f<-Right \n', 01234.56789)
     >> fprintf('Left->%11.0e<-Right \n', 0.123456789)
     ```