University of Cyprus
Department of Civil and Environmental Engineering

CEE221/CEE325 – Introduction to Matlab

Computer Labs, Saturdays 21/1/23 & 28/1/23

Exercise - 1:

- Start Matlab and, then, close and open the various windows (Workspace, Command-History, Current Directory, etc.).
- Use the command window as a calculator performing different kinds of computations, using the +, -, *, /, ^ operators, e.g.: 5.6+7.8, 7.5/1.2, $6.5*2.5^3$, etc..
- Define various variables and store some (scalar) values.
- Perform some computations, executing mixed expressions, in order to realize the order of precedence in which computations are performed.
- Use the commands *who* and *whos* to display the variables that are currently stored in the workspace of Matlab.
- Clear the workspace using the *clear* command and clear the command window using the *clc* command.
- Use the command *format* to switch between *long* and *short* format, compact and loose format and execute some computations to notice the differences.
- Use the help and doc commands to display help information and documentation, respectively, about Matlab commands and functions.
- Use the provided Matlab functions (trigonometric, in both radians and degrees, *sqrt*, logarithmic, rounding, etc.).
- Use the colon operator, with the default and with a specified step, to automatically create vectors of values.

Exercise - 2:

Use the commands that are necessary to request and get from the user (i.e. you) the width, b, and height, h, of a beam with a *bxh* rectangular cross-section, and compute the area ($A=b*h$) and the moment of inertia ($I=b*h^3/12$) of the specific cross-section.

Print the computed area and moment of inertia with all 3 different ways (while being computed, using the function *disp()* and using the function *fprintf()* with certain number of digits overall and after the decimal point.

Exercise - 3:

Define a row vector and then multiply it with a scalar.

Define a column vector and then multiply it with a scalar.

Define vectors using the colon (:) operator with various steps.

Access certain elements of the vectors you have defined.

Open the Workspace and see the representation of the defined variables and the various options provided by the Workspace window.

Define several matrices (arrays), using both the semicolon and a new line to define their rows and access certain elements of the matrices you have defined.

Define a few matrices with ones (containing elements with values of 1), with sevens (values 7), with different sizes.

Define identity matrices of various sizes.

Define diagonal matrices.

Use the function rand() to define arrays of certain dimensions with random numbers.

Determine the transposes of some of your matrices.

Define a 4x4 matrix with random numbers between 5 and 8 and then compute its transpose, its determinant, its inverse, and its rank. Subsequently, multiply that matrix with its inverse to check whether the result is an identity matrix.

Define some matrices with proper dimensions (sizes) and perform matrix multiplications and element-wise multiplication, division and raising to a power.

Use the functions *length()* and *size()* to obtain the dimensions of vectors and arrays.

Define some arrays and concatenate them.

Exercise - 4:

Define the proper arrays (matrices) and solve a system of algebraic equations, such as the following, to determine the unknown x, y and z, using **both** the \ operator and the inversion of the matrix of coefficients.

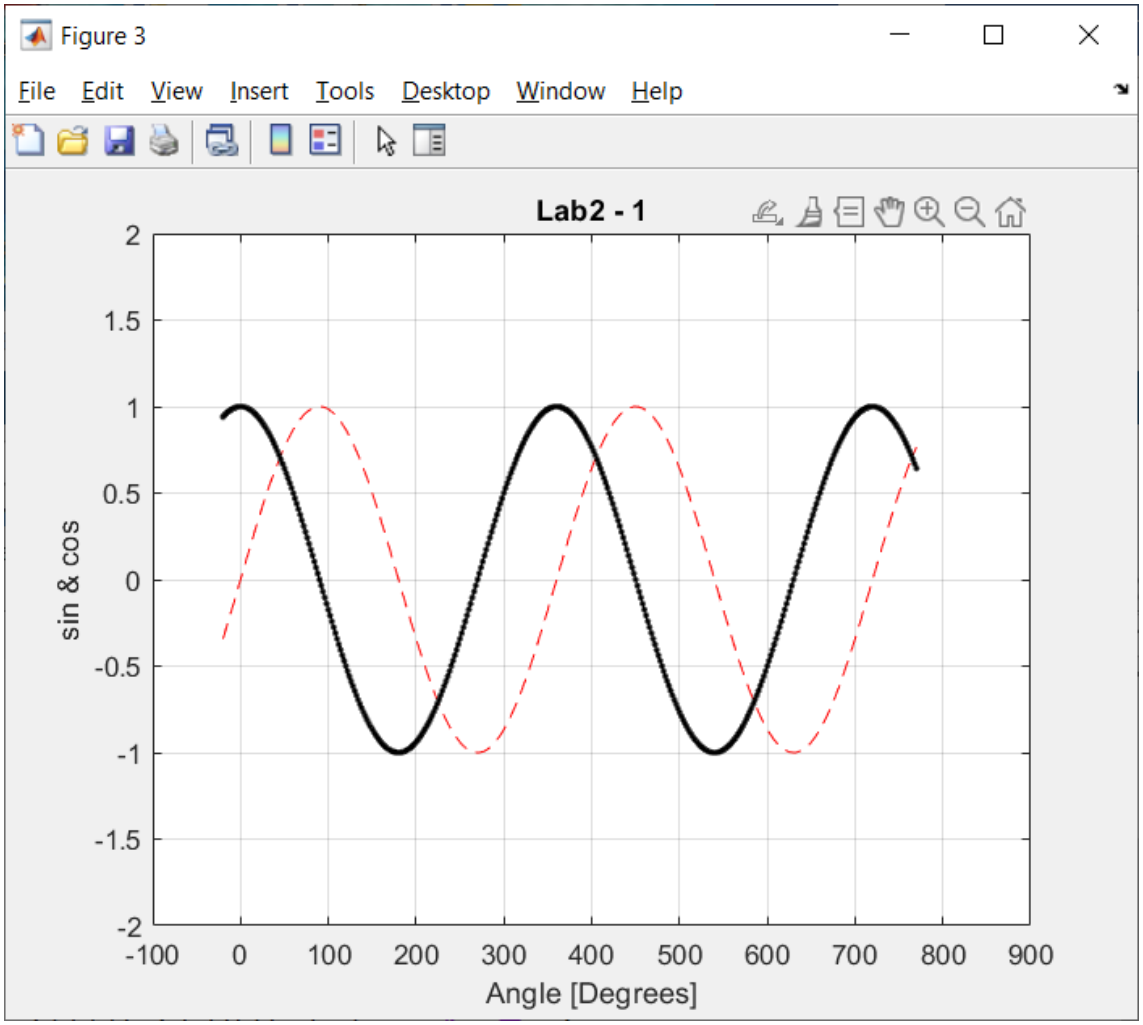
$$13x + 2y - 0.5z = 24.5$$

$$-2.5x - 3.5y + 5.7z = 5.6$$

$$7.5x + 9.1y – 12.5z = 6.1$$

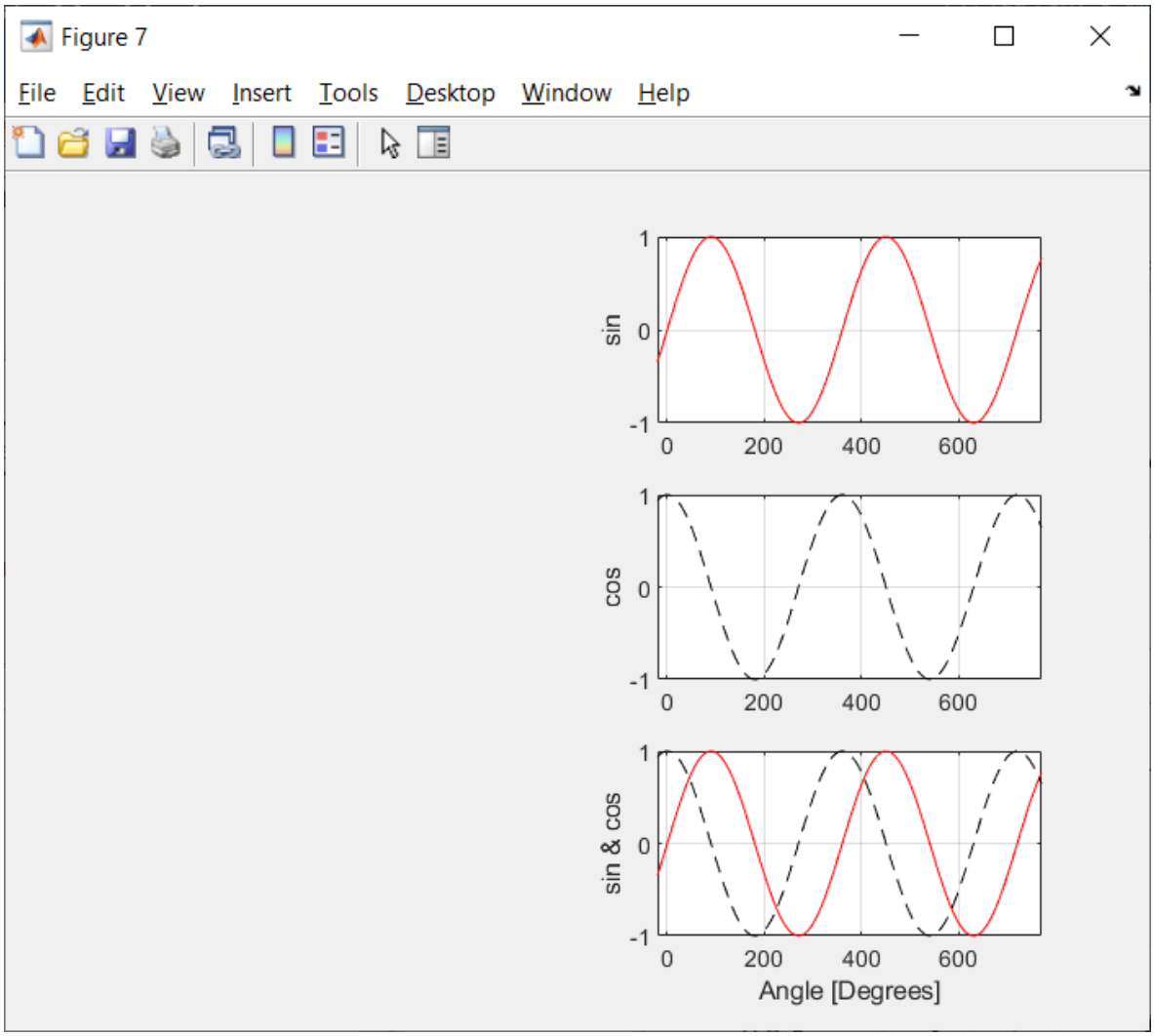
Exercise - 5:

- Provide the necessary Matlab commands to draw in the Figure 3, the sine(θ) with red dashed line '- -' and the cos(θ) with black '.-' line, with the angle θ, varying between -20 and 770 degrees with a step equal to 2.
- Add a title, e.g. *'Lab5',* to the figure and titles are the axes, specifically *'Angle [Degrees]*' on the horizontal axis and *'sin & cos'* on the vertical axis.
- Add a grid in the figure and set the horizontal axis to vary between -100 and 900 degrees and between -2 and 2 the vertical axis, as shown in the following figure.



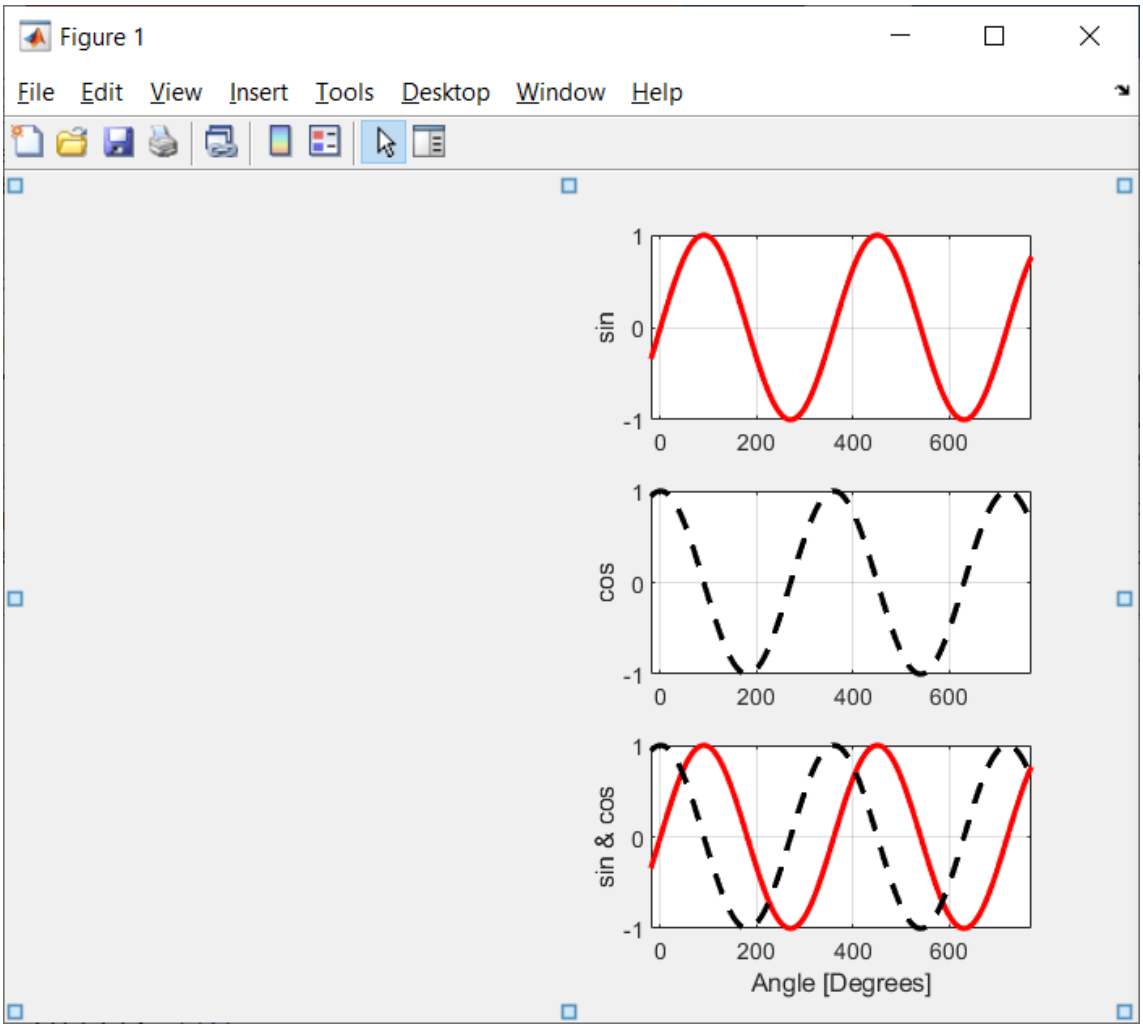
Exercise - 6:

- Create a new figure, the figure 7, which should be subdivided in 3 rows and 2 columns of subplots.
- Draw the sine, as computed in the previous exercise, with red continuous line, in the top right subplot, as shown below and add the world *sin* on the vertical axis.
- Draw the cosine, as computed in the previous exercise, with black dashed (- -) line in the center right subplot, as shown below and add the world *cos* on the vertical axis.
- Finally, draw in the center bottom subplot, as shown below, both the sine, , with red continuous line, and the cosine, with black dashed (- -) line, as computed in the previous exercise, and add the string '*Angle [Degrees]*' on the horizontal axis and the string *'sin & cos*' on the vertical axis.
- Save figure 7 both as a Matlab figure and as a *jpg* image, using the name *lab2Ex2*.



Exercise - 7:

- After saving, as a Matlab figure with the name ***lab6***, the figure created in Exercise - 6, close figure 7.
- After opening the Matlab figure ***lab6.fig***, change the width of the lines of the graphs, by either using the *plotedit* command or opening of the *Property Inspector,* and setting the *LineWidth* to 2, as shown below.



Exercise - 8:

- Redo Exercise-5, using a script, with the name *ex8.m*, where you should provide all Matlab commands that have been used in Exercise-5.
- Execute the script *ex8.m*

Exercise - 9:

- Write a function, named *celciusToFahrenheit()*, with one parameter named *tempC*, which will be used to provide the temperature in Celsius and compute and return the corresponding temperature in Fahrenheit.
- Note that in order to compute the temperature in Fahrenheit, the temperature in Celsius should be multiplied by 1.8 and 32 should be added.
- Write a script, in a file named *lab9.m*, to test the *celciusToFahrenheit*() function, by asking the user for a temperature in Celsius, calling the function to get the corresponding value in Fahrenheit and printing it.

Sample execution:

```
>> celciusToFahrenheit(0)
ans =
    32
>> celciusToFahrenheit(20)
ans =
    68
>> celciusToFahrenheit(30)
ans =
    86
>> celciusToFahrenheit(100)
ans =
   212
```

Exercise - 10:

- Develop a function, named ***constructMassMatrix***, which should accept a vector with the floor masses of a building (let's name it *floorMasses* in order to be meaningful) and, using the *diag()* function, create the corresponding diagonal mass matrix, which is diagonal and has the mass of each floor as the diagonal element in the corresponding row and column.
- Execute the function for a building with floor masses 15, 20 and 18 tons (1 ton = 1000 kg) of the 1st, 2nd and 3rd floor, respectively, and for a building with floor masses 35, 30, 30 and 25 tons of the 1st, 2nd, 3rd and 4th floor, respectively.

Sample execution:

```
>> constructMassMatrix([15e3 20e3 18e3])
ans =
       15000           0           0
           0       20000           0
           0           0       18000
>> constructMassMatrix([35000 30000 30000 25000])
ans =
       35000           0           0           0
           0       30000           0           0
           0           0       30000           0
           0           0           0       25000
```

Exercise - 11:

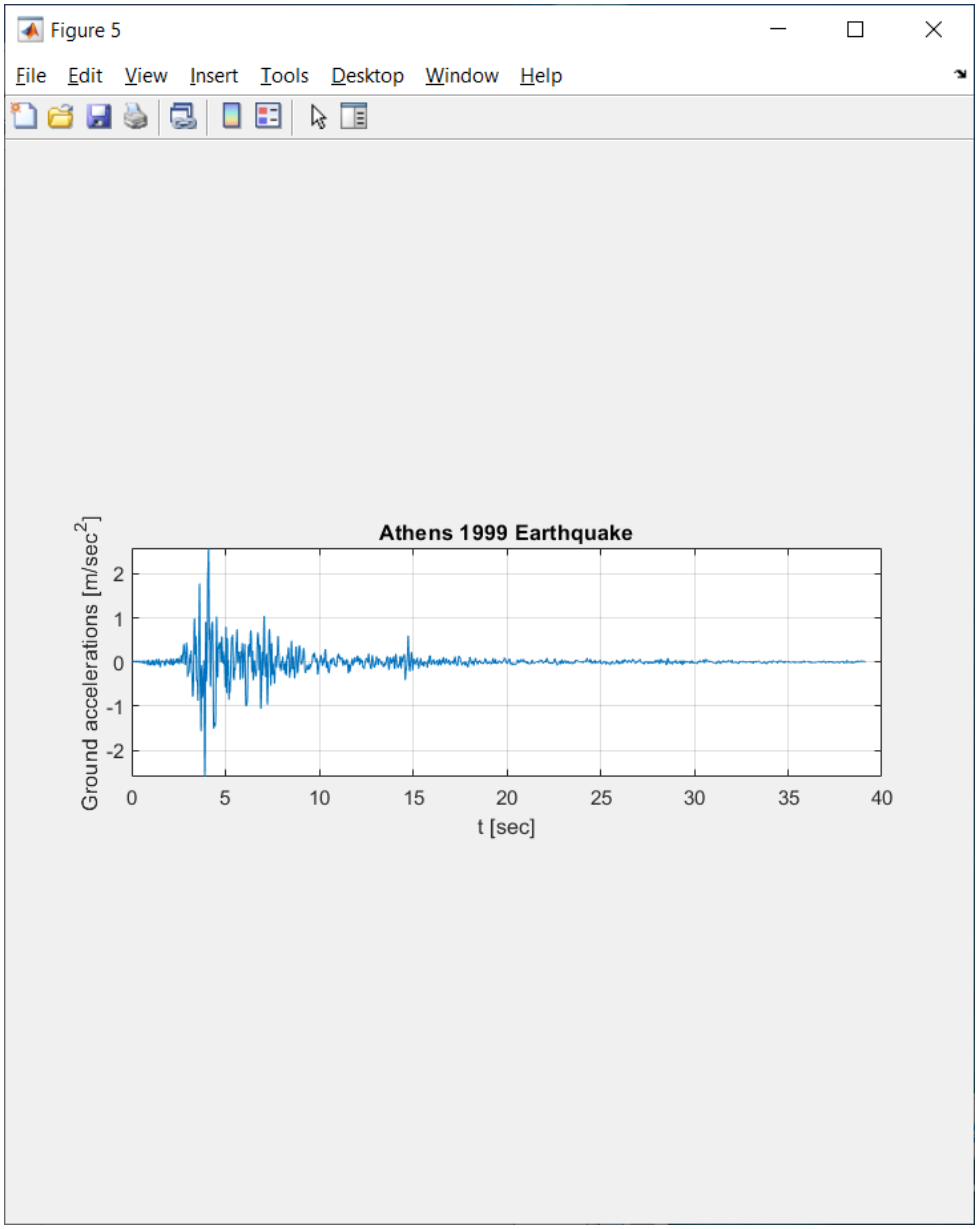
- Redo the previous exercise, but using a for loop to construct the mass matrix, i.e. do NOT use the *diag()* function.

Exercise - 12:

- An accelerogram (*επιταχυνσιογράφημα*) is a graph (plot) of the ground accelerations (εδαφικές επιταχύνσεις) in terms of the recorded time instances, as recorded by an accelerograph (*επιταχυνσιογράφος*) with a very small time-step (0.01 s or smaller).
- From the following URL (Uniform Resource Locator), you can download and save to your folder a file with the recorder ground accelerations along a specific direction, during a certain earthquake excitation.

  https://www.eng.ucy.ac.cy/petros/Earthquakes/earthquakes.htm

- The recorder ground accelerations are stored in the aforementioned file as 2 columns, the 1st one providing the times of recording and the 2nd one providing the recorded ground accelerations.
- After downloading and saving the file (e.g. *Athens_3_KallitheaN46_Accel.txt*) with the ground accelerations (directory) in a folder, which should be your active (current) Matlab folder (directory), write a Matlab script, named ***plotAccelogram.m***, to load the values and store the times (1st column) in a vector named *recordingTimes* and the ground accelerations (2nd column) in a vector named *recorderAcc*.
- Plot the accelerogram (i.e. ground accelerations vs. time instances) in the middle subplot of figure 5, which should be subdivided in 3 rows, as shown below, giving a proper title (e.g. "*Athens 1999 Earthquake*"), and axes titles (e.g. *'t [sec]*' and *'Ground accelerations [m/sec^2]*', respectively, as follows:



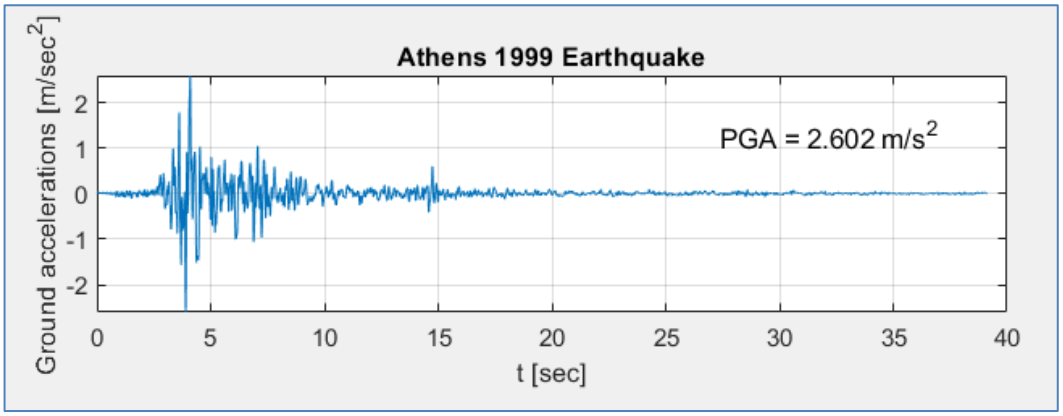
- Print the numbers of points (number of recorded instances) you have read and the time step between the points, as shown below.
- Print the maximum value, in absolute values, of the recorded ground accelerations, which is called Peak Ground Acceleration (PGA) expressed both in m/s^2 and g (where g=9.81 m/s^2), as well as the time at which the PGA has been recorded, as shown below.

```
>> plotAccelogram

 3913 pairs of values have been read
 With time step = 0.010 seconds

 PGA = 2.602400 m/s^2 = 0.265280 g at time 3.900
```

- Using the function *text()*, add a string providing the PGA at the coordinates corresponding to the 70% of the time range and the 50% of the PGA, as shown below.



Exercise - 13:

- Write a function, named *km2Miles()*,which will accept in one parameter a distance in kilometers (km) and convert and return it in miles (1 km = 0.621371192 miles).
- Write a script, in a file named *lab3_6.m*, to test the *km2Miles()* function, by asking the user for a distance in km, calling the function to get the corresponding value in miles and printing it.

  Sample execution:

```
>> test_km2Miles
Distance [km] = 100
100.0 km equals 62.137 m
>> test_km2Miles
Distance [km] = 60
60.0 km equals 37.282 m
```

Exercise – 14:

- In Exercise-3 you have developed a function, named *constructMassMatrix*(), which, after accepting a vector with the floor masses of a building, creates and returns the corresponding diagonal mass matrix, which is diagonal and has the mass of each floor as the diagonal element in the corresponding row and column.
- Write a script file, named ex7.m, in which you should prompt the user to provide the number of floors of a building and then ask for the mass of each floor, which should be added in a vector.
- Then, the function *constructMassMatrix*() should be called, sending the values of the vector with the mases of the floors, to obtain the mass matrix, which should

then be printed line by line through a loop using the function *fprintf()*, leaving 2 empty spaces between the elements of the matrix.

Sample execution:

```
>> ex7

Number of floors = 3
nFloors =
     3
Mass of floor-1 = 18e3
Mass of floor-2 = 14500
Mass of floor-3 = 14e3
     Mass Matrix
  18000.0  0.0  0.0
  0.0  14500.0  0.0
  0.0  0.0  14000.0
```