QUIZ 3 – MATLAB 2

Directions:

1. This is a limited open book quiz, meaning that the ***only*** sources of information allowed are *lecture notes, tutorial problems, textbook and your handwritten notes* reiterating/reinforcing the material presented in class. This information can be accessed in soft or hard copy form.
2. Please note that the access of files, documents, or information ***other than*** that explicitly listed at Point 1 will be treated as cheating and will be dealt with accordingly.
3. The access of social media, text messages, email or any other form of electronic communication will be treated as cheating and will be dealt with accordingly.
4. "Teamwork" and/or discussions are not permitted during the quiz.
5. The use of cell phones during the quiz is prohibited.
6. Only your owl submission will be marked.
7. **Quiz duration: 90 minutes (9:30 am – 11:00 am) + 10 minutes upload time**
8. **Total Duration: 100 minutes (9:30 am – 11:10 am)**

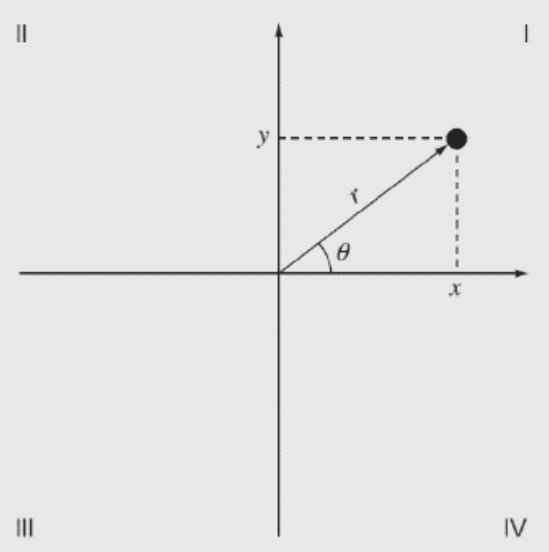
Submission Directions:

9. Code all your solutions in MATLAB. Save each problem as a separate '.m' file.
10. Ensure your upload to OWL is successful and that all files are functional. If there is a problem with your upload you must let the professor or TA know **BEFORE** leaving the zoom room. After leaving the quiz missing or corrupt files will not be accommodated for.

Note: While performing problems you can use the help function built into MATLAB to help you with syntax, and the proper use of commands.

Problem 1 [10 marks]

Two methods to represent a 2D location include Cartesian (x,y) and polar (r,Θ) coordinate systems.



$$r = \sqrt{x^2 + y^2}$$

x	y	Θ
>0		$\tan^{-1}(y/x)$
<0	>0	$\tan^{-1}(y/x) + \pi$
<0	<0	$\tan^{-1}(y/x) - \pi$
<0	=0	π
=0	>0	$\pi/2$
=0	<0	$-\pi/2$
=0	=0	0

Given the x and y coordinates of any point, the value of r can be calculated as shown above, and theta can be determined depending on the x and y values of the point. The various equations for calculating theta are given above along with the conditions when each should be applied.

We wish to create a function called `cart2polar` that converts the Cartesian definition of a point into polar coordinates. The function definition must be as follows:

```
function [r, theta] = cart2polar(x,y)
```

Where `r` and `theta` are the returned polar coordinates, and `x` and `y` are the Cartesian coordinates that are passed in to be converted.

a) Define the function & write code to calculate r using the x & y values passed in. **(2 marks)**

b) Using if/elseif/else structures, write code which will apply the appropriate formula to calculate the value of theta based on the conditions in the chart provided above. **(7 marks)**

c) The above equations calculate theta in radians, convert the resulting theta value to degrees at the end of your function so it returns the value in degrees. **(1 mark)**

You may test your function with the following known values in the format [(x,y) = (r, Θ [º])]:

(2,0) = (2,0) (2,1) = (2.23,26.6) (0,3) = (3,90) (-1,-2) = (2.23,-116.6)

Upload your appropriately named '.m' file(s) to OWL when completed. This is likely best done using a single '.m' file named P1.m with the `cart2polar` function embedded for simplicity.

Problem 2 [10 marks]

You have been transported back in time to Sept. 14th 1752 … or some might say it is Sept. 3rd … it's complicated. Yesterday the British parliament voted to switch from the Julian calendar to the Gregorian calendar resulting in 11 days being skipped. The issue with the Julian calendar is that it approximates the real year of 365.2422 days with one of 365.25 days, an error of 11 minutes per year. The peasantry has revolted following the passing of the bill as they believe the Gregorian calendar is literally stealing 11 days out of their lives and taking them to- their graves sooner.

Nevertheless, Parliament has tasked you with creating a program to convert Julian Date Numbers (JDN) to both Gregorian and Julian calendar dates. The JDN numerically counts each day from noon on January 1, 4713 BC (for the Julian calendar) or November 24, 4714 BC (for the Gregorian calendar). For example, the JDN of 2361222 is either Sept 14th (Gregorian) or Sept 3rd (Julian) in 1752, which is where the '11 days lost' comes from.

a) Using the table below, define each variable with its appropriate value. **(1 mark)**

Variable:	y	j	m	n	r	p	v	u	s	w	B	C
Value:	4716	1401	2	12	4	1461	3	5	153	2	274277	-38

b) Now, define the following equations for e, f_{julian}, $f_{gregorian}$, g , & h in MATLAB which will be used to convert JDN to both Gregorian and Julian calendar dates. **(2 marks)**

Note: all division operations are floor division, which means that the result is rounded down to the nearest integer. To accomplish this, use the built-in `floor` function on the division result. For example, if `x/y = 5/2 = 2.5`, then floor division of `5/2 = floor(5/2) = 2`.

$$f_{julian} = JDN + j \qquad f_{gregorian} = JDN + j + \frac{\left(\frac{4 \times JDN + B}{146097}\right) \times 3}{4} + C$$

$$e = r \times f + v \qquad g = \frac{mod(e,p)}{r} \qquad h = u \times g + w$$

c) We can now calculate the Day (D), Month (M), and Year (Y) of the JDN using the formulas below. Implement these in your script to convert the JDN to the D,M,Y format for both Julian and Gregorian calendars. **(5 marks)**

$$\boldsymbol{D} = (mod(h,s))/u + 1 \qquad \boldsymbol{M} = mod\left(\frac{h}{s} + m, n\right) + 1 \qquad \boldsymbol{Y} = \frac{e}{p} - y + \frac{n+m-\boldsymbol{M}}{n}$$

Note: again, division operations are floor division, use the `floor` function as described above.

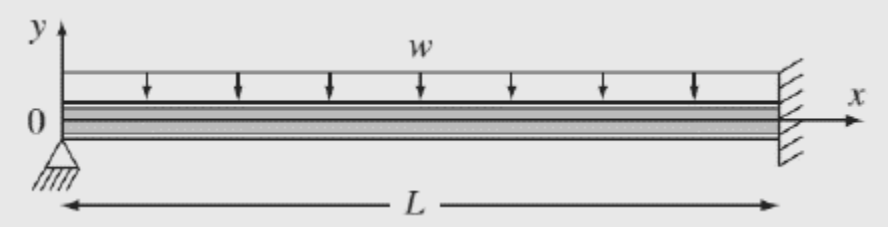
Note: the variables e, g, and h are different for the two calendars, so you may wish to have two sets of these values with names corresponding to each calendar.

d) Test that your script works using a JDN of 2361222 which corresponds to Sept 14th (Gregorian) or Sept 3rd (Julian) in the year 1752. Make sure that your script outputs your resulting dates in both formats in an appropriate way using `fprintf`. **(2 marks)**

Upload your appropriately named '.m' file containing your script to OWL when completed.

Problem 3 [10 marks]

Below is a figure showing a pinned-fixed beam subject to a uniform load (like something you may have seen in MME 2212).



The resulting deflection of the beam under this load can be expressed with the following equation:

$$y = -\frac{w}{48EI}(2x^4 - 3Lx^3 + L^3x)$$

The parameters of the beam are as follows: $L = 400\ cm$, $E = 52{,}000\ kN/cm^2$, $I = 32{,}000\ cm^4$, and $w = 4\ kN/cm$.

a) Define the above function as an anonymous function named `deflection`. **(1 mark)**

b) Plot the deflection of the beam using `fplot` from x = 0 to 400 cm. Label the vertical axis as '`deflection (cm)`' and the horizontal axis as '`x (cm)`'. **(2 marks)**

c) Use the built-in `fminbnd` function to determine the location where the deflection of the beam is the largest (in the negative y direction). Feel free to refer to the example we did in Chapter 2 that used this function. You can use an interval of x = 0 to x = L. **(4 marks)**

d) Have your script use `fprintf` to output the location and magnitude of the maximum deflection of the beam in an appropriate manner. **(1 mark)**

e) Repeat part (c) using the `goldmin` function we used in class. **(2 marks)**

Upload your appropriately named '.m' file(s) to OWL when completed. This is likely best done using a single '.m' file named P3.m.