EGR 141: Loops

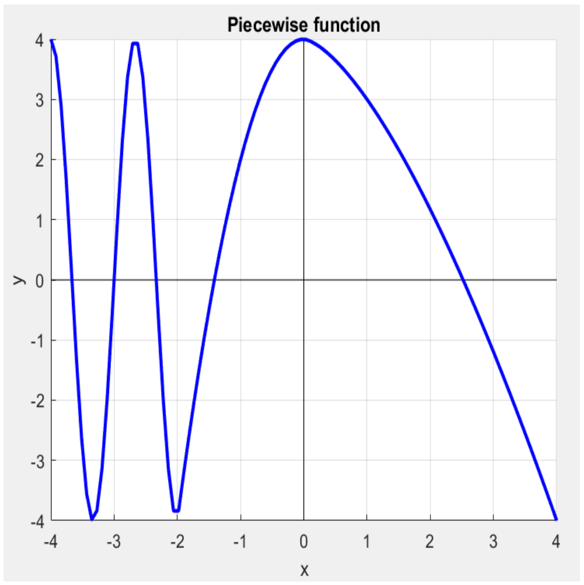
Summary: The goal of this lab is to help understand repetition statements, specifically for and while loops, along with vectorization. You should not use any MATLAB commands or concepts that are discussed in future chapters and sections.

- Each of the following problems should have a script and, possibly, a function associated with them.
- For each problem, the script file should be called something appropriate, such as *Lab6_1_yourName.m.*
- Include any functions that you needed to create in order to complete the problem. Name them whatever is indicated in the problem.
- Inside your script, solve each of the given problems. In between each problem, type *pause;* Clearly indicate where the code for each problem begins by using a comment block. Start each new problem with a *clear* .
- If my example output "lines up nicely" then your output should as well.
- All output statements involving variables should output variable values, not pre-computed constants. For example, if I ask you to output $r/2$ when $r = 3$, then you should set r to be three then output as *fprintf(' r/2 = %f ',r/2);* and not *fprintf('r/2 =1.5')* or *fprintf(r/2 = %f',3/2).*
- Note that example output for each problem is not necessarily correct output (I intentionally do different output than what you will do).

1. Create a function, *aPiecewise*, that takes in a vector x and returns the corresponding output of the below piecewise function. In your script file, plot the function on the domain $[-4, 4]$. Label the axis, give a reasonable title, and include the standard coordinate axis ($x = 0$ and $y = 0$) as black lines on the figure.

$$y = f(x) = \begin{cases} |x+2| - 4 & x < -2 \\ 4 - 2x^2 & -2 \le x \le 0 \\ 4 - \sqrt{x} & x > 0 \end{cases}$$

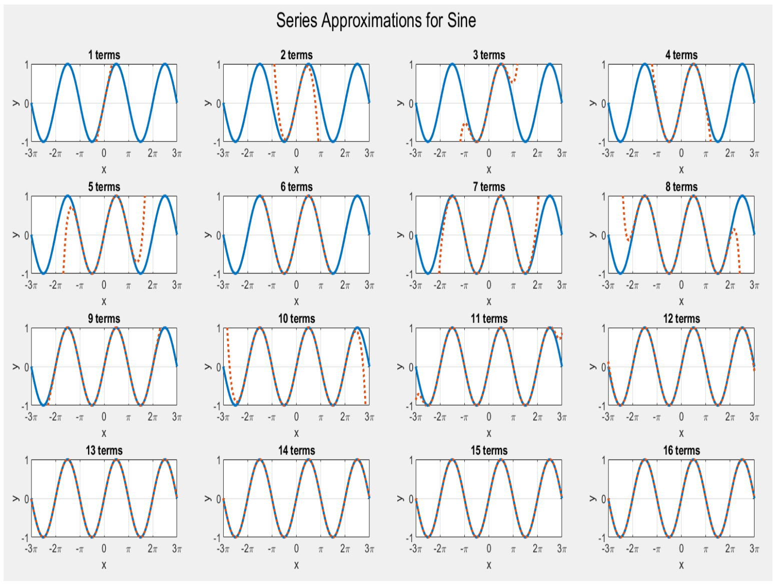
   Use a loop with if statements to obtain the desired output, do not use logical vector operations.



2. The Maclaurin Series for $\cos(x)$ is given by

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \dots$$

Note that we have an equals sign (yes, cosine is *equal* to an infinite polynomial. Isn't math fun!). Use subplot and for loops to create 16 plots on one figure. Each figure should contain a plot of $\cos x$ for $-3\pi \le x \le 3\pi$ and the Taylor Series approximation with n terms (for n from 1 to 16). For example, your first plot should have simply $\cos x$ versus 1. The second plot has $\cos x$ and $1 - \frac{x^2}{2!}$. Restrict the axis to the given x coordinates and $y \in [-1, 1]$. Label each axis, label each $n\pi$ on the x-axis, and label the entire figure appropriately (take a look at *sgtitle*). Do not use any built-in MATLAB functions to either generate or evaluate the series. You need to code the series "from scratch."



3. Create a function that takes in a single vector

$$v = \begin{bmatrix} v_1 & v_2 & v_3 & \cdots & v_{n-1} & v_n \end{bmatrix}$$

and computes the continued fraction

$$\text{continued fraction} = v_1 + \cfrac{1}{v_2 + \cfrac{1}{v_3 + \cfrac{1}{v_4 + \ddots}}}$$

with n terms. Some examples:

$$v = [2\,4] \rightarrow \text{continued fraction} = 2 + \frac{1}{4} = \frac{9}{4}.$$

$$v = [1\,8\,6\,2] \rightarrow \text{continued fraction} = 1 + \cfrac{1}{8 + \cfrac{1}{6 + \frac{1}{2}}} = \frac{119}{106}.$$

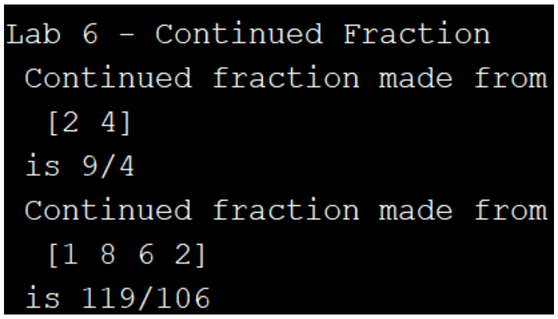
$$v = [6\,0\,1] \rightarrow \text{continued fraction} = 6 + \cfrac{1}{0 + \frac{1}{1}} = 7.$$

Look up and learn how to use *rat* to display your answer. You must use loops (you may not use recursion). Output the continued fraction for the following three vectors. All vectors should be coded dynamically (should work with *any* size), not hard-coded. In other words, part (b) should be coded for the first N Fibonacci numbers, just test on the first 20.

(a) Odd integers from 1 to 21 (including endpoints)

(b) The first 20 Fibonacci numbers (Starting at one, not at zero)

(c) A vector of size 15 where each element contains a digit of π, in order ([3 1 4 1 5 ...])



```
Lab 6 - Continued Fraction
 Continued fraction made from
  [2 4]
 is 9/4
 Continued fraction made from
  [1 8 6 2]
 is 119/106
```

4. Create a function, called *rootApprox* which takes in:

   - A function handle, f
   - A number in the domain, a
   - A number in the domain, b
   - Some tolerance, tol

   rootApprox should return an approximation to a root of $f(x)$ between $x = a$ and $x = b$.

   - Check to see if $f(a)$ and $f(b)$ have opposite sign (one is positive, the other is negative). If they have the same sign, return an empty array and quit the function (if they are both positive or negative, then $f(x)$ doesn't necessarily pass through the x-axis between a and b)
   - Set $m = \frac{a+b}{2}$ (the midpoint of our interval)
   - Run the following algorithm until $|f(m)| < tol$
     - If $f(m)$ has the same sign as $f(a)$, set $a = m$. (Set the new interval to be [midpoint, b])
     - If $f(m)$ has the same sign as $f(b)$, set $b = m$. (Set the new interval to be [a, midpoint])
     - Set $m = \frac{a+b}{2}$ again and repeat. (Find a new midpoint)
   - Return m.

   In your main script, test on each of the following functions with $a = 0$, $b = 4$ and $tol = 10^{-6}$

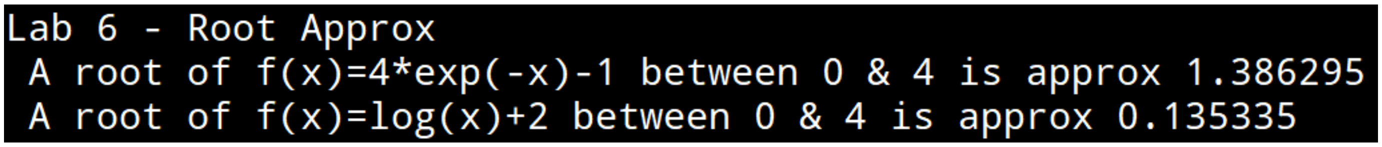
   (a) $f(x) = x^2 - 3$

   (b) $g(t) = \cos\frac{t}{2}$

   (c) $h(\theta) = \sin\frac{\theta}{2} - e^{-\theta}$

   (d) $k(x) = |x^2 - 3x + 2|$

   Print out your results nicely to the screen.



```
Lab 6 - Root Approx
 A root of f(x)=4*exp(-x)-1 between 0 & 4 is approx 1.386295
 A root of f(x)=log(x)+2 between 0 & 4 is approx 0.135335
```