CH2013
Computational Programming and Simulations Lab
Aug-Dec 2021

Problem Sheet #2

Aug 25 2021

We move further into our practice of MATLAB coding. Upload the codes for each question separately in the appropriate location at grader.mathworks.com, pass the pre-tests, and then submit. For this Problem Sheet, I am allowing 5 submissions/attempts. Please be cognizant of this.

1. You will recall using a "for" loop in the last class. Read up the syntax for it and make sure it's clear. Another kind of loop in MATLAB involves "conditional statements", or the ***if-elseif-end*** loops. The idea is simple, go look at the help documentation for it first.

   Write an if-elseif/else-end loop that does the following
   i. If the scalar variable is negative, it does nothing
   ii. If the scalar variable = 0, it increases its value by 1
   iii. If the scalar variable is positive, it squares it.

   Let A=[-1 0 12]. We will use conditional statements to modify A, to get a new vector B. Write a **for loop**, that picks up each element of the vector A in turn, and assigns them to the scalar variables. Now use the ***if-elseif-end*** loop you have created to update the value of this scalar variable and assign in order as elements of the vector B.

   Check to make sure you have the right answer. Note that the if-elseif/else-end loop is "nested" within the for loop. Make sure your indentation looks good.

2. Functions are very handily used in modern MATLAB versions. The script file is compact as a consequence of using functions; and the code is overall more efficient.

   a) Write a function called '**sumofelements**' that calculates the sum of the elements in a vector, C. The inputs to the function should be the vector, and the number of elements in the vector. Output of the function is the sum. Put this function at the bottom of the script file.

   b) Call the above function from a script file to find the sum of elements of vector **C'=[1 4.5 3.2 1.78 0 -9]**. The number of elements in the vector is **N** and the sum of the elements is **Sum2**. Use the in-build MATLAB function "**numel**" to determine the number of the elements in the vector – don't hard code N=6!

   c) In the same script file, define matrix **D=[1 4 5 9; -1 3 5 9; 8 2 1 6; 15 10 -2 10]**. Find the diagonal elements of D (using a single command in MATLAB) and put them in a vector **E**. Find the sum of the elements of E using the above function and called it **Sum3**.

   d) Next, find the sum of the elements in the ***first column*** of D and call it **Sum4**.

Note that for each of (b-d), you have to just call the function 'sumofelements' once and the code for that function should be at the bottom of the script file. Also keep in mind that in the function, you should initialize the sum to 0 at the start.

3. The iterative methods of solution of non-linear equations, such as the bisection, false position and Newton's method, can be easily implemented using your own code in MATLAB. Here, the if-elseif/else-end loops can come in useful to check on whether the convergence criterion is satisfied. Look up the command "break" – it will help in exiting a loop when a condition is satisfied. The algorithm for Bisection Method is illustrated on the screen for your reference. Note that we are doing two checks for convergence.

   a) Develop a function "**myfun**" which calculates the value of $f(x) = e^{0.5x} - 5 + 5x$. The input is x and output is f.
   b) Develop a function "**relativeerror**" which calculates the value of $error1 = 100 \left|\frac{x_{new} - x_{old}}{x_{new}}\right|$. The inputs are x_{new} and x_{old} and output is *error1*.
   a) Develop a function "**bisection**" which calculates $x_{new} = \frac{x_l + x_u}{2}$. The inputs are x_l and x_u, and output is x_{new}
   c) In the script file, put together the code for solution of the function shown in (a) using the Bisection Method. The initial guesses are 0 & 2. Tolerance is 10^{-6} for both x and f. Use the algorithm displayed. Make sure to call ***myfun***, ***relativeerror***, ***bisection, break*** and use nested for/if-then-else loops. The final answer should be in the variable ***xfinal1*** and final function value in ***ffinal1***. Create a vector Err1 and save all the values of the error in the vector; and create a semi-log plot of Err1 vs. Iteration number.
   d) Make a plot of f(x) vs. x and make sure that your xfinal makes sense.

4. In this question, you will repeat the above to get a solution using the false position method, with the same initial guesses. The functions myfun and relativeerror can be used as such from above, and don't need to writte again (always designate x_l as the x_{old} in error calculations, in this example).
   a) Develop a function "**falseposition**" which calculates $x_{new} = \frac{x_l f(x_u) - x_u f(x_l)}{f(x_u) - f(x_l)}$. The inputs are x_l, x_u, $f(x_l)$ and $f(x_u)$ and output is x_{new}
   b) Develop the code for solving the function in 3(a) using the false position method. Use a similar algorithm and code structure as above. Put the final answer in ***xfinal2*** and final function value in ***ffinal2***. Create a vector Err2 and save all the values of the error. Create a plot of Err2 vs. Iteration number.
   c) Analyse the error pattern – comparing the two methods you have used.

5. Use the above developed codes to find the solution at other initial guesses. Examine the converged values of x and f. (no submission)

--END--