Problem Set 08 · While Loops

Instructions

1. Use the answer sheet provided in the Assignment Files to complete this problem set. Fill out the header information. Follow any additional instructions that appear in the answer sheet. Submit your finished answer sheet with the rest of your deliverables.
2. You will need your PS07_academic_integrity function for this assignment. You should make any fixes necessary to make it work.
3. Read each problem carefully before starting your work. You are responsible for following all instructions within each problem. Remember that all code submissions must follow the course programming standards.
4. Below are the expected deliverables for each problem.
    - Name your files to match the format in the table below.
    - Publish your code for each problem. See PS06 for more information.
    - Do not forget to include any data files loaded into your code.

Item	Type	Deliverable to include in Submission
Problem 1: Taylor Series for cos(x)	Paired	☐ PS08_taylor_*login1*_*login2*.m ☐ PS08_taylor_*login1*_*login2*_report.pdf
Problem 2: Infinite Fin Model	Individual	☐ PS08_fin_length_*yourlogin*.m ☐ PS08_fin_length_*yourlogin*_report.pdf
Problem 3: Simplex Optimization	Individual	☐ PS08_simplex_*yourlogin*.m ☐ PS08_simplex_*yourlogin*_report.pdf ☐ PS08_funceval.p ☐ Your sub-function to evaluate a known function
PS08 Answer Sheet	Individual	☐ PS08_AnswerSheet_*yourlogin*.docx
Academic Integrity Statement	Individual	☐ PS07_academic_integrity_*yourlogin*.m

5. Save all files to your Purdue career account in a folder specific to PS08.
6. When you are ready to submit your assignment,
    - Compress all the deliverables into one zip file and name it **PS08_yourlogin.zip**. Be sure that you
        i. Only compress files using **.zip** format. No other compression format will be accepted.
        ii. Only include deliverables. Do **not** include the problem document, blank templates, etc.
    - Submit the zip file to the Blackboard drop box for PS08 before the due date.
7. After grades are released for this assignment, access your feedback via the assignment rubric in the My Grades section of Blackboard.

Helpful MATLAB Commands

Learn about the following built-in MATLAB commands, which might be useful in your solutions:

`round, factorial, any, all`

Problem 1: Taylor Series for $\cos x$

Paired

Learning Objectives

Calculations	01.00 Perform and evaluate algebraic and trigonometric operations
Variables	02.00 Assign and manage variables
Text Display	05.00 Manage text output
Relational & Logical Operators	14.00 Perform and evaluate relational and logical operations
User-Defined Functions	11.03 Create a user-defined function that adheres to programming standards
	11.04 Construct an appropriate function definition line
	11.05 Match the variables names used in the function definition line to those used in the function code
	11.06 Execute a user-defined function
Flowcharts	15.03 Construct a flowchart for an indefinite looping structure using standard symbols and pseudocode
	15.04 Track a flowchart with an indefinite looping structure
	15.09 Create test cases to evaluate a flowchart
	15.10 Construct a flowchart using standard symbols and pseudocode
Repetition Structures	17.02 Convert between these indefinite looping structure representations: English, a flowchart, and code
	17.03 Code an indefinite looping structure

Problem Setup

Many complicated functions used in science and engineering applications are made easier to understand when represented as Taylor series. Taylor series are also used in science and engineering applications to make approximations. For instance, the cosine function can be approximated with the Taylor series as:

$$\cos x = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k}}{(2k)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots$$

In this problem, your task is to create a user-defined function that calculates the approximate value of the cosine of a given number by summing an unknown number of terms in the series. The number of terms will be determined by establishing a "tolerance", which is the maximum allowable value of the final computed term in the series.

For example, the table below shows the number of terms, values of each Taylor series term, and the approximate value of cosine of 2 when the tolerance is 0.01.

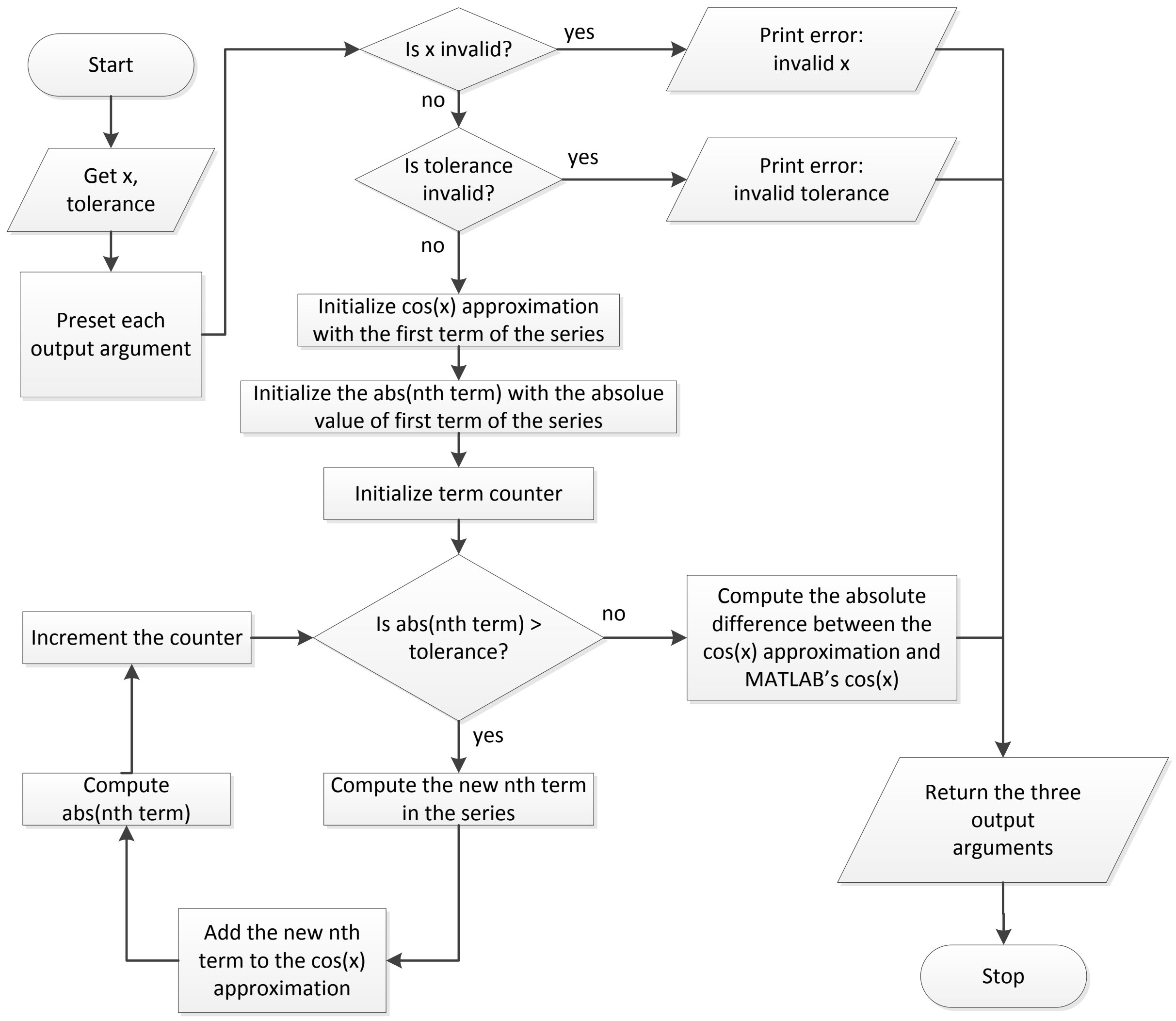
Number of Terms , *n*	Value of (*k*)	Value of *nth* Term	Taylor Series Value of cos (2)
1	0	1	1
2	1	-2	-1
3	2	0.6667	-0.3333
4	3	-0.0889	-0.4222
5	4	0.0063 (|0.0063| ≤ 0.01, so final computed term)	-0.4159
	MATLAB's built-in cosine function: cos(2) = -0.4161		

After determining the approximate value of the cosine function, your UDF needs to calculate the absolute difference between that value and MATLAB's value for the cosine. In the table above, the absolute difference is |(-0.4159) – (-0.4161)|.

Your user-defined function must:

- accept as input arguments x and the tolerance value for the last term of the Taylor series
- test for invalid inputs
  - x must be a scalar;
  - the tolerance must be between 0 and 1, not inclusive
  - print appropriate, helpful error messages for invalid inputs
- return as outputs:
  - the number of terms used in the Taylor series computation,
  - the computed value of $\cos x$ using the Taylor series, and
  - the absolute difference between the Taylor series computed value of $\cos x$ and the value returned by MATLAB's built-in cosine function
  - if an input argument is invalid, then return each output argument above as -99

A flowchart is provided for this problem and shows a method to code the Taylor series of the cosine function. Translate this specific flowchart to MATLAB code.


Start
Get x, tolerance
Preset each output argument
Is x invalid?
yes
Print error: invalid x
no
Is tolerance invalid?
yes
Print error: invalid tolerance
no
Initialize cos(x) approximation with the first term of the series
Initialize the abs(nth term) with the absolue value of first term of the series
Initialize term counter
Increment the counter
Is abs(nth term) > tolerance?
no
Compute the absolute difference between the cos(x) approximation and MATLAB's cos(x)
yes
Compute abs(nth term)
Compute the new nth term in the series
Return the three output arguments
Add the new nth term to the cos(x) approximation
Stop

Problem Steps

1. **Before you start to code**: Review the flowchart to understand the process for using the Taylor series to compute $\cos x$. Note that error messages are printed to the MATLAB Command Window.
2. In your Word answer sheet:
    a. Add a series of test cases to thoroughly test all the possible paths (valid and invalid paths) in the flowchart. One test case has been provided on your answer sheet.
    b. Record the corresponding flowchart outputs for each test case.
    c. Complete the variable tracking table by hand for the execution of the loop for the test case provided.
3. Translate the flowchart above to a MATLAB user-defined function named **PS08_taylor_cos_*yourlogin1_yourlogin2*.m**. Comment your code appropriately and follow the ENGR132 Programming Standards.

   Hint: learn about MATLAB's `factorial` command

4. Test your function by calling it from the Command Window with the test cases you created in step 2a. Do not suppress the output when you call your function. Paste the function call and results displayed in the Command Window as comments under the `COMMAND WINDOW OUTPUTS` section of your function file.
5. Call your academic integrity function in the `ACADEMIC INTEGRITY` section.
6. Publish your function as a PDF file using any valid test case and name the file as directed in the deliverables list.

Problem 2: Infinite Fin Model

Individual

Learning Objectives

Below are learning objectives that may be used to assess your work on this problem. Learning objectives from past assignments may also be used to assess your work. Use the links to find the full evidence lists for each topic.

Calculations	01.00 Perform and evaluate algebraic and trigonometric operations
Variables	02.00 Assign and manage variables
Text Display	05.00 Manage text output
Relational & Logical Operators	14.00 Perform and evaluate relational and logical operations
User-Defined Functions	11.00 Create and execute a user-defined function
Flowcharts	15.03 Construct a flowchart for an indefinite looping structure using standard symbols and pseudocode
	15.04 Track a flowchart with an indefinite looping structure
	15.09 Create test cases to evaluate a flowchart
	15.10 Construct a flowchart using standard symbols and pseudocode
Selection Structures	16.01 Convert between these selection structure representations: English, a flowchart, and code
	16.02 Code a selection structure
Repetition Structures	17.02 Convert between these indefinite looping structure representations: English, a flowchart, and code
	17.03 Code an indefinite looping structure

Problem Setup

Electronics generate a lot of heat while operating. Computers have cooling systems embedded in their motherboards to dissipate the heat. One method of cooling is to use extended surfaces, commonly referred to as fins. The fins are a means to enhance heat transfer between the motherboard and the surrounding air by adding to the surface area over which convection cooling occurs. You can approximate the temperature along the fin using a technique called an infinite fin model. This method allows you to simplify the temperature calculations by assuming that the temperature of the fin at the end farthest away from the heat source is equal to the ambient air temperature.

The equation for the infinite fin model is

$$T = T_\infty + (T_b - T_\infty)e^{-mx}$$

where T is the temperature of the rod at a given length from the heat source, T_b is the temperature at the end of the rod that is attached to the heatsource, T_∞ is the ambient air temperature, x is the distance from the heat source along the rod, and m is a constant associated with the rod and is defined as

$$m = \sqrt{\frac{hP}{kA}}$$

where h is the heat transfer coefficient, P is the perimeter around the cross-sectional area of the rod, A is the cross-sectional area of the rod, and k is the thermal conductivity of the rod. You can assume constant thermal conductivity for this method.

You are testing fins of different materials. You plan to use rods, each 0.005 m in diameter, of copper, aluminium, and stainless steel. One end of the rods will be maintained at 373 K. The surface of the rod is exposed to ambient air at 298 K, with a convection heat transfer coefficient of 100 W/m^{2}K. The thermal conductivity of the three metals are as follows:

Metal	**Thermal Conductivity (W/(m·K))**
Aluminium	205
Copper	400
Stainless Steel	16

Your task is to determine the minimum rod length, to the nearest centimeter, necessary to use the infinite fin model. You must create a user-defined function that will calculate the temperature along the rod. This function must

- Accept four (4) valid inputs:
  - Rod diameter
  - the metal's thermal conductivity
  - the heat source temperature, and
  - the ambient air temperature.
- Return one output: the minimum length of the rod.
- Check that all inputs are greater than or equal to zero and prints an appropriate and useful warning message. If one of the inputs is invalid, then return a rod length of -1.
- Round the modelled temperature, T, to the nearest whole number to be in line with the accuracy of your temperature measuring device.
- Display the minimum rod length to the Command Window using professional formatting.

Problem Steps

1. **Before you start to code**: Create a flowchart to outline how information should move through the code. You can draw the flowchart using any means that result in a clear image for the answer sheet. Make sure your flowchart is legible. Options include:
   - Drawing it by hand and taking a <u>clear</u> photo
   - Drawing it directly in the Word answer sheet using Word's drawing tools
   - Drawing it in Microsoft's Powerpoint, Publisher, or Visio
   - Using another flowchart tool, such as Lucidchart

2. In your Word answer sheet:
    a. Complete the variable tracking table by hand for the test case provided.
3. Translate your flowchart to a MATLAB user-defined function. Comment your code appropriately and follow the ENGR132 Programming Standards.
4. Test your function by calling it for a copper rod and then an aluminium rod.
5. For the two tests, paste the function call and results displayed in the Command Window as comments under the `COMMAND WINDOW OUTPUTS` section of your function file.
6. Call your academic integrity function in the `ACADEMIC INTEGRITY` section
7. Publish your function as a PDF file using **stainless steel as the inputs** and name the file as directed in the deliverables list.

Problem 3: Simplex Optimization

Individual

Learning Objectives

Below are learning objectives that may be used to assess your work on this problem. Learning objectives from past assignments may also be used to assess your work. Use the links to find the full evidence lists for each topic.

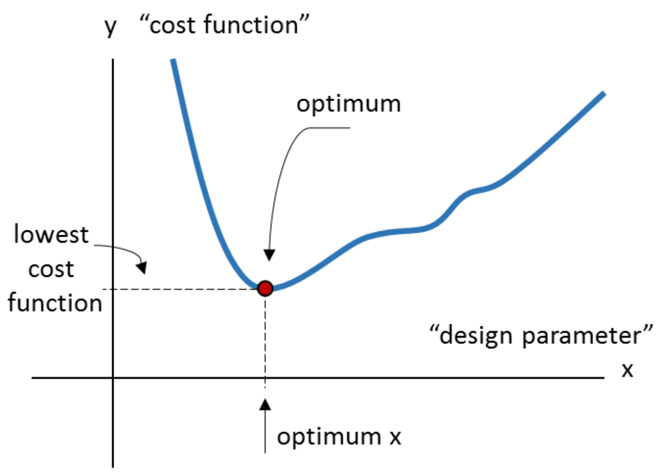
Calculations	01.00 Perform and evaluate algebraic and trigonometric operations
Variables	02.00 Assign and manage variables
Text Display	05.00 Manage text output
Relational & Logical Operators	14.00 Perform and evaluate relational and logical operations
User-Defined Functions	11.00 Create and execute a user-defined function
Flowcharts	15.03 Construct a flowchart for an indefinite looping structure using standard symbols and pseudocode
	15.04 Track a flowchart with an indefinite looping structure
	15.09 Create test cases to evaluate a flowchart
	15.10 Construct a flowchart using standard symbols and pseudocode
Selection Structures	16.00 Create and troubleshoot a selection structure
Repetition Structures	17.02 Convert between these indefinite looping structure representations: English, a flowchart, and code
	17.03 Code an indefinite looping structure

Problem Setup

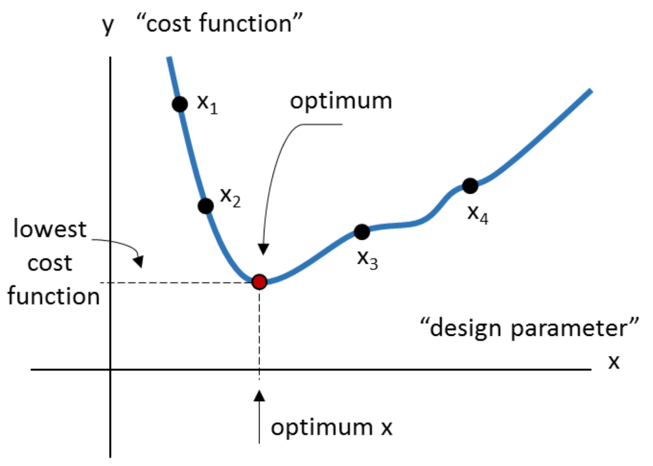
Design engineers often seek to optimize parameters in a design. For example, jet engine designers attempt to maximize the thrust-to-weight ratio of the engine, and structural designers seek to maximize the strength-to-weight ratio of a structure. The parameters to be optimized can be expressed as an objective function (also called a cost function), which is a mathematical function that captures the goal of the design. Usually engineers pose optimization problems that seek to minimize the cost function, meaning they would like to find the minimum possible value for that function.

One way to search a design space for a solution is to use a technique known as simplex searching. In a simplex method, engineers pick several points in the design space, and based upon the cost function value associated with those points, they make decisions about where else in the design space to search for the minimum function value.

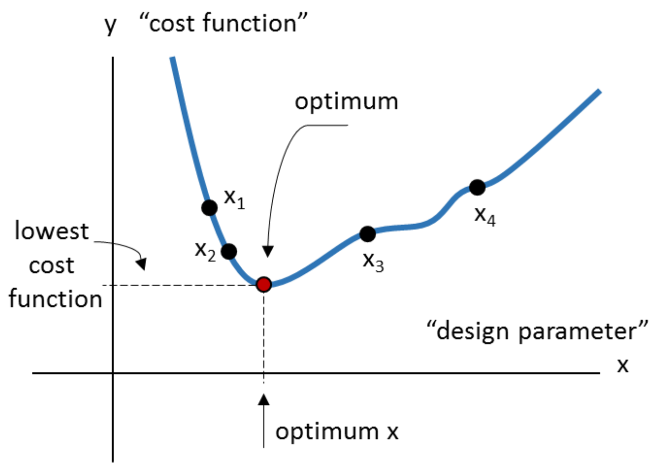
Let's say we had a cost function as shown below. The optimum value for the design variable x lies at the lowest function value, as labeled on the figure.



A systematic way to search for that minimum value would be to use 4 starting points: x1, x2, x3, x4.



If we know the function values associated with each point, we can make decisions about where else to search. For example: if y(x3) is the lowest of the four cost function values, then we can re-assign x2 as the new x1, because we know that both x1 and x2 give higher cost function values than x3. Then we define a new x2 that is between the new x1 and x3, so the new situation looks like this:



After this, we evaluate the new y(x2), which looks like it's less than y(x3). This means we can re-assign x3 to be the new x4, define a new x3, and then begin the process again. Because the width of the interval on which we are searching is always decreasing (x1 and x4 are always getting closer to each other), we know that we will eventually converge on the optimum point. We can say that our search is over when (x4 – x1) is sufficiently close to 0.

The simplex method is only robust (meaning it will work, every time) if:

- The cost function has only one minimum, and
- The initial interval [x1, x4] on which we search contains that minimum.

One significant virtue of the simplex method is that we don't actually have to know what the function is (mathematically) for the method to work.

Your task is to write an executive function that will use the simplex method to find the minimum of an unknown cost function. Your function must

- Define an interval for the problem. Start by setting x1 to the minimum x value and x4 to be the maximum x value over which you will be evaluating the function. You can assume that the unknown function will have its minimum within the interval [-10, 10]
- Initialize x2 and x3 locations between the interval defined above
- Evaluate the function at the 4 x-locations
- Use a while loop to perform the optimization
- Set the tolerance of (x4 – x1) to be 0.0001
- For each iteration,
  - Update x1 and x4 similar to the figures and procedure explained above
  - Find an iterative way to determine x2 and x3 (remember that these points are always between x1 and x4)
  - Plot the locations of each guess
  - Display to the Command Window the current iteration number
  - Display to the Command Window the (x4 – x1)
- Display the final x and y coordinates of the minimum to the Command Window

While you write your simplex optimization code, you must test your simplex method with a known function, that you create, to debug your code (a quadratic would be a good choice). Once your simplex method is working, you will need to find the minimum of the unknown function in the p-code file **PS08_funceval.p**. A p-code is a protected file type that can be called within MATLAB but whose code you cannot read. P-codes do not have help lines due to the protection process, so below is the help documentation for each function.

PS08_funceval.p

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program Description
%   This p-code evaluates a specific function at a given x-location
%
% Function Call
%   y = PS08_funceval(x)
%
% Input Arguments
%   1. x = the x value of the function
%
% Output Arguments
%   1. y = the corresponding value for f(x)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Problem Steps

1. Create an executive user-defined function to perform the simplex optimization. Follow good programming standards.
    a. You may want to use a flowchart to outline your plan for the simplex optimization code.
    b. Create a sub-function UDF that works like PS08_funceval.p but uses a quadratic for which you can identify the minimum by hand. The UDF should accept one input argument, the x value of the function, and should return an output argument that is the corresponding value for f(x). Name this sub-function with a unique, meaningful name
    c. Call your known function UDF from Step 1b while you work on your simplex code.
2. Confirm that your simplex method works by using it to find the minimum of your chosen quadratic. It should display all the required information and meet the criteria stated in the problem setup.
3. Once you have your simplex function debugged and working as intended, comment out your known function and perform the simplex method on the unknown function using the p-code `PS08_funceval.p`. Note that this function should work in a similar fashion to the sub-function you created in Step 1b.
4. Paste the simplex function call and results displayed in the Command Window as comments under the `COMMAND WINDOW OUTPUTS` section of your function file.
5. Call your academic integrity function in the `ACADEMIC INTEGRITY` section.
6. Publish your executive function to a PDF and name the published file as required in the deliverables list.