

MATLAB Assignment 1

by Brent Hickman¹

In this assignment you will write some MATLAB code to graphically investigate Sterling's formula, which states that for large N

$$\log(N!) \approx N \log(N) - N.^2$$

Specifically, we will investigate how accurate Sterling's formula is for **small** N . The methods you will be instructed to use will probably seem extremely pedantic (and you would be right), but you should keep in mind that the object of this exercise is not to learn about a mathematical formula. At this point I don't even intend for you to learn about efficient programming yet. The object here is simply to familiarize you with some basic features of MATLAB programming, nothing else. The parts of this first assignment are annotated with the lesson numbers of Kermit Sigmon's tutorial (hereafter KS) which cover related topics; however, this will not be true of subsequent assignments. I advise you in the strongest terms possible to carefully read through KS in its entirety **before** beginning this assignment.

1. Write a function file to compute the factorial value of a single scalar argument and call the function "**sfactorial**". DO NOT name it "**factorial**", as MATLAB already has a built-in factorial function by that name.³ This new function should have the following 3 components:
 - A. An "if" statement which returns an error message if the argument is negative.⁴
 - B. An "elseif" statement which returns an error message if the argument is not an integer. You should use either the built-in **round**, **floor** or **ceil** functions to test for non-integers.⁵ You may have noticed that you have had only a very loose introduction to the **round**, **floor** and **ceil** functions in MATLAB yet. Thus, part of this exercise is to acquaint you with using the MATLAB documentation to find out what they do. This is in fact the most valuable skill you can acquire from this course. You can learn about these functions by either typing "**help ceil**" or "**doc ceil**" at the command line. Alternatively, you could also open up the help window and browse/search for the documentation you need. It would be a good idea to try all 3 methods, as they each prove most useful in different circumstances.
 - C. An "else" statement with an embedded "for" loop that does the actual factorial calculation. Make sure that your **sfactorial** function is able to handle any non-negative integer, including 0 (recall that the factorial of 0 is defined as 1).⁶ You should also test it out on several

¹brent-hickman@uiowa.edu

²The "!" denotes the factorial operator, which is defined only for non-negative integers and where $N!$ is the product of all non-negative integers which are less than or equal to N . But you already knew that...

³KS lesson 12

⁴KS lessons 6 & 13

⁵KS lessons 6, 7 & 13

⁶KS lesson 6

different numbers to make sure that it gives correct solutions and produces error messages rather than output when appropriate. HINT1: start by defining a variable named “result” and set it equal to 1. Then have the number of iterations depend on the input argument, redefining the result variable in each iteration as the product of itself with the iteration index. HINT2: if you instruct a “for” loop to iterate over i from a to b , if $a > b$ then the loop will execute zero times.

- D. Starting on the first or second line, you should include a few lines of comments explaining what the `sfactorial` function is and what it does. Remember that comment lines start with the % sign.⁷ Once you have properly saved your function file, type “`help sfactorial`” and see what happens.
2. Write another function file capable of accepting a vector of scalar inputs *of arbitrary length* and computing a factorial solution for each element of that vector. Call this new function “`vfactorial`”. HINT: copy your code for `sfactorial` into a new file and embed all of the tests and computations into a large “for” loop that performs the necessary operations one-by-one for each element of the vector of inputs. To do so, you will find it useful to use the built-in `length` function (see documentation) to determine the number of elements in the input vector. Also, since you will have two nested “for” loops, be sure to give the index variables different names, say i and j .
 3. Write an interactive script file that invokes your new `vfactorial` function and the built-in `log` function to graphically verify Sterling’s formula.⁸ After requesting some input from the user, this script will define a vector of domain values, being non-negative integers from 0 to 21. It will then evaluate both sides of the formula and graph the results for a visual comparison. Your file should have the following components:
 - A. An input variable called “`input`” which produces a message requesting either a 1 or 0 from the user.⁹ The value of this variable as supplied by the user will determine which of two possible methods for defining the domain vector will be employed.
 - B. By use of “if” and “else” statements, the script should then decide what to do next.¹⁰ if `input = 1`, it should read in a data matrix from the file `MATLAB1.dat` and assign the first column to the variable “`domain`”.¹¹ If `input = 0`, it should use some sort of matrix-building operation (left to your discretion) to construct a vector of ascending integers from 0 to 21 and assign it to the variable “`domain`”.¹² Once you are done coding the solution, try running

⁷see examples in KS lesson 12

⁸KS lesson 12

⁹KS lesson 13

¹⁰KS lesson 6

¹¹See MATLAB documentation on the `load` function and KS lesson 11 for help on selecting only a single column of a matrix.

¹²See MATLAB documentation on the built-in `linspace` function or the colon operator (*i.e.* do a keyword search on “colon operator” in the help window).

your program and entering some value other than 0 or 1 and see what happens. Can you explain why?

- C. By invoking your `vfactorial` function and MATLAB's built-in `log` function, it should then compute the left-hand and right-hand sides of Sterling's formula, assigning the result of the former to the variable "LHS" and the latter to the variable "RHS".¹³ While computing RHS, you should make use of the special "period operator" to allow for coordinate-wise multiplication.¹⁴
- D. Finally, you should graph LHS and RHS on the same axes for comparison. In doing so, use the `hold` utility in MATLAB.¹⁵ Graph LHS in blue and RHS in red. Your graph should have a title, a legend displayed in the upper right-hand corner within the plot, and labels for both the horizontal and vertical axes.¹⁶

¹³KS lesson 7

¹⁴KS lesson 3

¹⁵`help/doc hold`

¹⁶lesson 18 and see also the following MATLAB documentation: `help/doc plot`; `help/doc legend`; `help/doc figure`