M20 - Introduction to Computer Programming with MATLAB
Instructor: Prof. Enrique López Droguett, Ph.D.
Teacher Assistants: M. Fidansoy, G. San Martín, M. Pishahang, V. Vela.
Fall 2023 – UCLA
Student: *Alex Lie*
UCLA ID: *905901892*

HOMEWORK 3

Task 1: Re-implementing matrix operations

Introduction

I am using MATLAB to create two different functions for matrix operations. The first function will calculate the trace (the sum of the diagonal elements) of a square matrix. The second function will compute the transpose of a matrix. Although there is already a trace() and transpose() function defined in MATLAB already, creating these functions is good practice for beginners who are learning MATLAB. We can use these functions in other programs that we write.

Model and Theory

1) $Tr(A) = \sum_{i=1}^{n} a_{ii}$
2) $If\ A = [a_{ij}]_{m\times n}\ ,\ then\ A' = [a_{ij}]_{n\times m}$

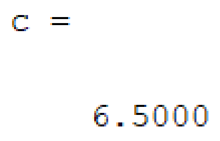
Methodology

First, I created a .m file for Task 1, cleared the workspace and command window, declared the matrix m, and called the matrix_trace() and matrix_transpose() functions using m as an input. I then created a new .m file called matrix_trace. This function will have an output c, which is a number, and an input m, which is a matrix. I first checked if the input is a matrix. If it is, I then check if it is a square one. I did this by using conditional statements. Then using a for-loop, I accessed the diagonal values of the matrix and added them up in the variable c, following equation 1. I then ended the function. For the matrix_transpose function, I created a new file called matrix_transpose. This function will have an output mt, which is a matrix, and an input m, which is also a matrix. I first checked if the input is a matrix using a conditional statement. If it is a square matrix, I then created a new matrix called mt. The width of mt is the height of m, and the height of mt is the width of m. Using a nested for-loop, iterating through the widths and heights of mt, I set values for each position in mt to the corresponding value in m, following equation 2. I then ended the function. I ran my code in the Task1.m file and my output worked the way I intended it to! Lastly, I added comments to make my code easier to understand for other users.

Calculations and results

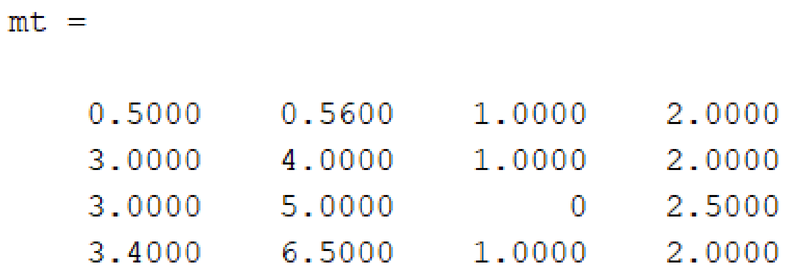
See next page.

3)



```
c =

    6.5000
```



```
mt =

    0.5000    0.5600    1.0000    2.0000
    3.0000    4.0000    1.0000    2.0000
    3.0000    5.0000         0    2.5000
    3.4000    6.5000    1.0000    2.0000
```

c is the value of matrix_trace(m) and mt is the value of matrix_transpose(m).

<u>Discussions and Conclusions</u>

Testing the functions using matrix m, my output (3) is correct and my code works as intended!

I added additional conditional statements in my code in case the user who uses my functions attempts to use an array with multiple dimensions as an argument. In that case, the output will be a string that says, "Input is not a matrix". For the trace_matrix() function, I added another conditional statement to check if the input is a square matrix by comparing the width and height. If it is not a square matrix, then the output will be a string that says "Input is not a square matrix". Otherwise, the code will continue as intended. Although I did not include these test cases in my code, you are welcome to try running non-matrices and non-square matrices as arguments to my functions.

Since the trace() and transpose() functions already exist in MATLAB, there is no reason for me to use my function (other than coding practice). Calling trace(m) and transpose(m) will give the same results as my output using my own functions.

<u>Task 2: Recursive Functions</u>

<u>Introduction</u>

I am using MATLAB to create two recursive functions. The first function calculates the factorial of a number. The second function generates the nth value of the Fibonacci sequence. Although there are already factorial() and fibonacci() functions defined in MATLAB, creating these recursive functions is good practice for beginners learning MATLAB. We can use these functions in other programs that we write.

<u>Model and Theory</u>

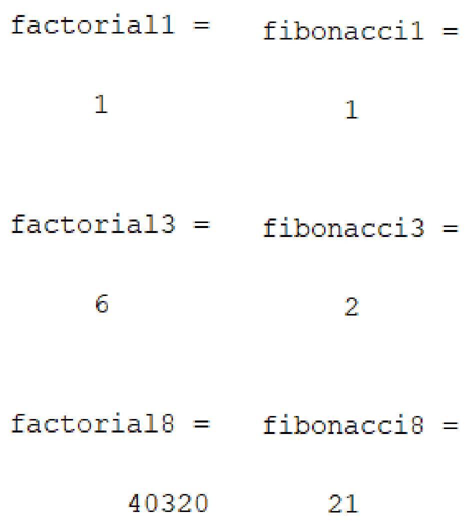
1) $n! = n \times (n - 1)!$
2) $F_n = F_{n-1} + F_{n-2}$

Methodology

First, I created a .m file for Task 2, cleared the workspace and command window, and called the calc_factorial() and calc_fibonacci() functions using 1, 3, and 8 as arguments. I then created a new .m file called calc_factorial. This function will have an output f, which is a number, and an input n, which is also a number. I first checked if the input is an integer by using the mod() function. I also checked if the input is not a negative number using a relational operator. I did this by using conditional statements. Then, once those conditions were passed, if the original input n is greater than 1, I set f equal to n times the value of the calc_factorial() using n-1 as an input, following equation 1. Then the function gets called again. When the calc_factorial() function is called with an input that is less than or equal to one, then the value of 1 will be the output. Then all of the other calc_factorial() functions can be computed and the value of f becomes the factorial of the original n input. I then ended the function. For the calc_fibonacci() function, I created a new .m file called calc_fibonacci. This function will have an output f, which is a number, and an input n, which is also a number. I first checked if the input is an integer by using the mod() function. I also checked if the input is not a negative number using a relational operator. I did this by using conditional statements. Then, once those conditions are passed, I check if n is equal to zero. If it is, then I set f equal to zero. If not, then I check if n is equal to 1. If it is, then I set f equal to 1. If not, then I set f equal to calc_fibonacci(n-1)+calc_fibonacci(n-2), following equation 2. When those calc_fibonacci() functions get passed with an input value of 2, then all of the other calc_fibonacci() functions can be computed and the value of f becomes the nth value of the Fibonacci sequence. I then ended the function. I ran my code in the Task2.m file and my output worked the way I intended it to! Lastly, I added comments to make my code easier to understand for other users.

Calculations and results

3)



```
factorial1 =    fibonacci1 =

     1                1


factorial3 =    fibonacci3 =

     6                2


factorial8 =    fibonacci8 =

       40320        21
```

calc_factorial() and calc_fibonacci() functions were called for input values of 1, 3, and 8.

Discussions and Conclusions

Testing the functions using the values 1, 3, and 8 as inputs, my output (3) is correct and my code works as intended!

I added additional conditional statements in my code in case the user who uses my functions attempts to pass a non-integer or a negative number as an argument. In that case, the output will be a string that says, either "Input must be an integer" or "Input must be non-negative. I included this because values of n that are negative or not an integer are not defined values for finding the factorial or nth term in the Fibonacci sequence. If the input is a non-negative integer, the code will continue as intended. Although I did not include these test cases in my code, you are welcome to try running non-negative and non-integer numbers as arguments to my functions.

Since the factorial() and fibonacci() functions already exist in MATLAB, there is no reason for me to use my function (other than coding practice). Calling factorial(m) and fibonacci(m) will give the same results as my output using my own functions.

Task 3: Traveling Salesman Problem

Introduction

I am using MATLAB to write a program to solve the Traveling Salesman Problem. I want to find a route to visit all locations that has a small cost, but the cost does not necessarily have to be minimized. I am creating an heuristic algorithm (sacrificing optimality for speed) called the Nearest Neighbor Algorithm to solve this Traveling Salesman problem. This function that I will be making will work with any amount of locations with a distance matrix and any starting location.

Model and Theory

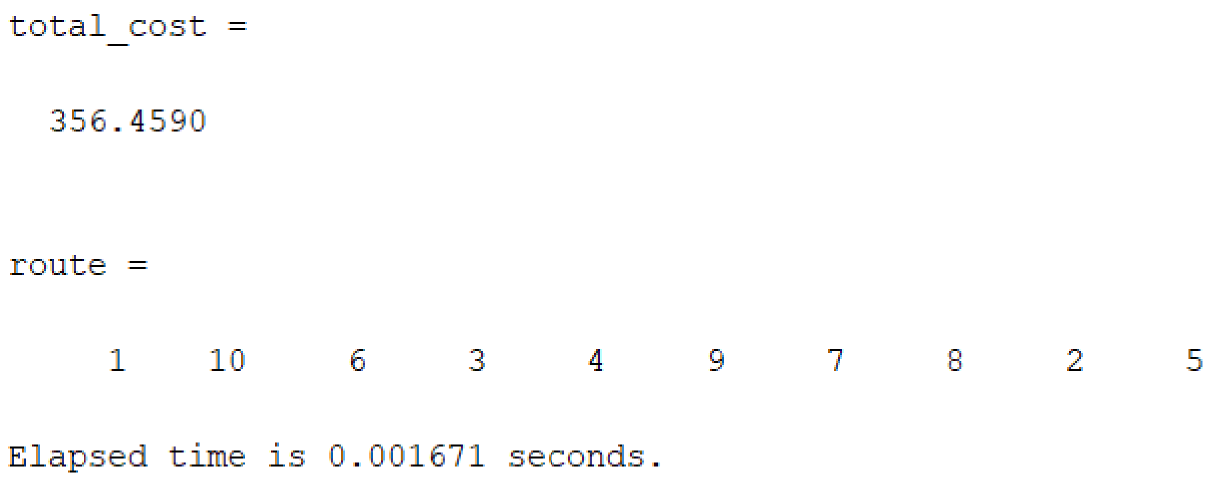
1) $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Methodology

First, I created a .m file for Task 3, cleared the workspace and command window, and declared the X and Y position vectors. I then created my d matrix, which represents the distances between two nodes. I did this by using a nested for-loop, iterating through each node twice and calculating the distance by using the distance formula (1). Once I have matrix d, I set two variables, total_cost and route, to be set to the output of the function NN_TSP(1,d), which is the nearest neighbor function that I made. I also put tic prior to calling the function, and toc after calling the function, to calculate how long it takes for the function to run. I then created a new m. file called NN_TSP. This function will have the outputs total_cost, which is a number, and route, which is a vector. This function will have the inputs start, which is a number, and D, which is a matrix. To start the function, I first set a boolean vector called notVisited to initialize all of the locations as unvisited. I then set the current location to start, and marked the start location as visited. I then created the route vector and set the first element in the route vector to the start location. I also set the total cost to zero. Then in a while loop with the condition being that not all of the locations have been visited yet, I multiplied the notVisited vector by the row of the D matrix that corresponds to the current location, to set certain distances to zero (these distances are from locations that have already been visited. Then, I set all of the values equal to zero to NaN instead, which allows me to use the min() function to figure out the location to the minimum distance, excluding the distances that are equal to 0. I then set the next location as visited, added the location to the next available index in the route vector, and increased the total cost to add in the new distance. Once every location has been visited, the while loop and function ends. I ran my code in the Task3.m file and my output worked the way I intended it to! Lastly, I added comments to make my code easier to understand for other users.

<u>Calculations and results</u>

2)



```
total_cost =

  356.4590


route =

     1    10     6     3     4     9     7     8     2     5

Elapsed time is 0.001671 seconds.
```

The route vector shows what order to visit each location in, and total_cost represents the distance traveled by taking this route. My function takes about 0.001671 seconds to run.

<u>Discussions and Conclusions</u>

Testing the functions using the locations at positions X and Y, my output (2) seens correct and my code works as intended! Each location is listed once in the route vector, and the total_cost seems like a reasonable minimal cost based on the values in matrix d. I ran through the Nearest Neighbor algorithm manually and I verified that the total cost and route order in the output is correct!

This is most likely not the smallest possible cost, but it is a pretty small cost compared to other potential costs from different routes taken.

I also tried using the Nearest Neighbor algorithm on Figure 1 in the homework #3 instructions document. I verified the output matches my answer from solving it manually.

The time it takes for my function to run with a 10x10 matrix will vary, but it is always in the ballpark of .001 seconds. If my matrix was larger, then the function would be slower. If the matrix was smaller, then the function would be faster.