Aaron Kuo
UID: 305-170-284
CEE / MAE M20
July 3rd 2020

HOMEWORK 01

1 Calculating Surface Area of Oblate Spheroid

1.1 Introduction

The objective of this problem is to develop a script that calculates both the actual surface area of an oblate spheroid, as well as an approximation of the surface area using user-specified radii. The results will be calculated using the built in MATLAB π value as well as the built in trigonometry functions (radian form). After calculations, the results will be printed to the screen with 10 digits for each answer to allow for any discussion of differences.

1.2 Model and Theory

The applicable equations for this particular problem are:

$$Area(r_1, r_2) = (2\pi\ (r_1^2 + \frac{r_2^2}{sin(\gamma)} ln(\frac{cos(\gamma)}{1 - sin(\gamma)}))$$

$$Approximation(r_1, r_2) = 4\pi\ (\frac{r_1 + r_2}{2})^2$$

$$\gamma = arccos(\frac{r_2}{r_1})$$

Where

- r_1 - Equatorial Radius
- r_2 - Polar Radius
- π - A constant, either 3.14 or via MATLAB built-in value

1.3 Methods and Pseudo-code

The calculation process goes as follows:

1. Prompt user to input radii (r_1, r_2)
    - Error check user input for viability (value of r_2 must not be greater than r_1)

2. Calculate Surface Area using the first equation listed above with given variables
3. Calculate Approximation Surface Area using the second equation listed above with given variables
4. Print out both results

1.4 Calculations and Results

With given example radii of $r_1 = 6378.137$ and $r_2 = 6356.752$, the resulting output from the script is:

```
Enter value for r1:
6378.137
Enter value for r2:
6356.752

The surface area of an oblate spheroid is: 510065604.9
The approximation of the surface area of an oblate spheroid is:
509495321.6
```

With an example radius of $r_1 = 6000$ and $r_2 = 7000$, the resulting output from the script is:

```
Enter value for r1:
6000
Enter value for r2:
7000

Error using HW1 (line 21)
Error. r1 must be greater than r2
```

1.5 Discussions and Conclusions

It can be shown that the approximated value for the surface area of the Earth is less than the actual value but very similar, proving both equations are valid for solving for surface areas of oblate spheroids. This code is able to run on any two variable radii that fit the given specifications, namely r_2 having a smaller value than r_1.

2. Neighbor Identification

2.1 Introduction

The objective of this problem is to utilize the concept of linear indexing to find neighboring cells in a M x N matrix given a user-specified cell. The results will be calculated using a script that will classify the cell's position in a matrix and use its position to identify its neighbors. Following the calculation, the script will print out the Cell ID that was chosen and the associated neighboring cells in ascending order.

2.2 Model and Theory

The concept behind this problem is largely based on linear indexing, where a single number is used to identify a cell in a 2D matrix. For this problem, all matrices created are assumed to having a cell numbering order where the top left cell is *always* the start with Cell ID #1 and increases as you go down the column. For example, if we have a 3 x 3 matrix and our chosen cell is 3, its neighbors would be 2, 5, and 6. If the chosen cell was 5, its neighbors would be all 8 surrounding numbers. For cells not touching the borders of the matrix, the script would identify all 8 surrounding members. For cells touching walls, the script would calculate its neighbors in a different fashion since those cells would not have all 8 neighbors surrounding them. The detailed process will be discussed below.

2.3 Methods and Pseudo-code

1. Prompt user to input specific cell and matrix specifications
    - Error checks user input for viability (e.g. M and N must have a value greater than 2, the specified cell must fall within the range of the matrix)
2. Identify Specified Matrix Location
    - The script will identify if the cell is located at a "wall" of the matrix or if it is in the center. For example, if the specified cell as P = M, it will know that the specified cell is at the bottom-left corner of the matrix
3. Calculate neighbors based on position
    - Based on the location the script identified in the previous step, it will calculate its neighbors. Using the same example of P = M

```
elseif P == M
    neighbors = [P_Up P_Right P_UpperRight];
```

    - The script will know that the specified cell is at the bottom-left corner of the matrix and will have 3 neighbors, the cell above, the cell on the right, and the cell at the upper-right corner of the cell.
    - The neighbors are identified using a pseudo-code that uses the number of the specified matrix and adds or subtracts from it.

```
P_Down      = P + 1;
P_Left      = P - M;
```

    - For example, the cell below an identified cell would be its number plus one. The cell to the left would subtract the number of the size of the matrix column.
4. Print the Cell ID and its neighbors in ascending order

2.4 Calculations and Results

With and example matrix of M = 4, N = 6 and a specified cell of P = 11, the resulting output from the script is:

```
Enter value for M:
4
Enter value for N:
6
Enter value for P:
11

Cell ID:      11

Neighbors:     6     7     8    10    12    14    15    16
```

With an example matrix of M = 3, N = 3 and a specified cell of P = 11, the resulting output from the script is:

```
Enter value for M:
3
Enter value for N:
3
Enter value for P:
11

Error using HW1 (line 48)
Error. Please enter a value of P within the range of 1 to M*N
```

2.5 Discussions and Conclusions

It can be shown that linear indexing is a successful method to identify cell neighbors in this matrix under the proper circumstances. In regards to a linear indexing scheme used for a 3D grid, a similar system can be used to identify its neighbors. On a 2D grid, there are 3 types of classification we use:

Corner - 3 neighbors

Edge - 5 neighbors

Interior - 8 neighbors

For a 3D grid, we can classify as such:

Corner - 7 neighbors

Edge -11 neighbors

Face -17 neighbors

Interior -26 neighbors

Similar to our script for a 2D grid, the script would identify the position of the specified cell. We are assuming the matrix follows the same numbering convention so a 3 x 3 matrix would start with 1 at the top left corner in the first layer, 10 on the second, and 19 on the third. For simplicity sake, we will use a 3 x 3 matrix and choose 1 as the identified cell. The neighbors on the first layer can be identified the same way as a 2D grid, with:

P_Right = P + M

P_Down = P + 1

P_LowerRight = P - M + 1

To identify the neighbors on the second corner, we can simply add the number of cells there are in one layer. For example

P_Back = P + (M*N)

P_Back_Right = P + (M*N) + M

P_Back_Down = P + (M*N) + 1

P_Back_LowerRight = (P + (M*N)) - M + 1