Jeffrey Ding
UID: 104928991
CEE M20
April 13th, 2018

Homework 2

1. The Lunar Phase Calculator

1.1 Introduction

The goal of this problem is to develop a method that calculates the lunar phase for a user-specified date. Then, the user-specified date, illumination, and lunar phase will be printed.

1.2 Model and Methods

The script prompts the user to input the month, day, and year that will be evaluated. The user-specified month is used to calculate a monthNum that will be used in future calculations and error handling. The default value for monthNum is zero. However, the script uses an if-else ladder to set the monthNum to the corresponding month, for example, “JAN” sets monthNum to 1 to “DEC” sets monthNum to 12. The user-specified day and year is converted to a dayNum and yearNum value using the str2num function for use in future calculations and error handling.

For error handling, the script first checks if dayNum or yearNum is an empty array. If not, the script checks if the month isn’t valid, being that monthNum stays at the default value of 0. If not, check if the day isn’t valid, being that the input is of incorrect format or if the dayNum is negative or greater than 31. If not, check if the year isn’t valid, being that the input is of incorrect format or if yearNum is less than 1900 or if yearNum is greater than 9999. If not, check if the date isn’t valid, being that some of the months have only 30 days or the year is a leap year and the inputted month is February.

If no errors are caught, that means that the inputted date is valid. The script then sets an a offset that is 0 for all cases except for January and February, where it is 1. Then, the script calculates the y and m values using the following two lines of code:

```
y = yearNum - a + 4800;
m = monthNum + 12*a - 3;
```

Then, the script calculates the J and deltaJ values using the following two lines of code:

```
J = dayNum + floor((153*m + 2)/5) + 365*y + floor(y/4) - floor(y/100) + floor(y/400) -
32045;
```

```
deltaJ = J - 2415021;
```

This allows the script to calculate for phaseCalc and use that to calculate the L value. The following lines of code calculates these values:

```
T = 29.530588853;
phaseCalc = (rem(deltaJ, T))/T;
L = (sin(pi*phaseCalc))^2;
```

The script that prints the user-specified date and Illumination percentage using these lines of code:

```
fprintf('%s %s %s:\n', month, day, year);
fprintf('Illumination = %.1f percent\n', L*100);
```

A final if-else statement determines the printing of the lunar phase likes this:

```
if phaseCalc < 0.5
   fprintf('Waxing\n');
else
   fprintf('Waning\n');
end
```

1.3 Calculations and Results

When the program is executed, and the user inputs *month* value JAN, *day* value 15, and the *year* value 1900, the following output is printed to the screen:

```
Please enter the month as MMM (e.g. JAN): JAN
Please enter the day as DD (e.g. 01): 15
Please enter the year as YYYY (e.g. 2000): 1900
JAN 15 1900:
Illumination = 99.3 percent
Waxing
```

The output states that on the user-specified date, the phase was waxing and the illumination was 99.3 percent.

1.4 Discussion

The output generally displaying a full moon and a new moon alternating every 2 weeks. This is realistic because a lunar phase cycle is essentially one month. Basically, it takes

2 weeks to go from a new moon to a full moon and another 2 weeks to go from a full moon to a new moon.

According to my model, the date of the next full moon is April 30th, 2018. This is only one day off from the actual date, which is April 29th, 2018.

The assumption we are making in this problem is that we are using January 1st, 1900’s full moon as the baseline for all lunar phase calculations. Therefore, all calculations estimations based on an event in the distant past. A way to make this method better would be to calculate the user-specified date using the most recent new-moon date relative to the user-specified date.

2. Neighbor Identification

2.1 Introduction

The goal of this problem is to develop a method that identifies the neighbors of a user-specified cell in a user-specified dimensional rectangular array. Then, print the user-specified cell and its neighbors.

2.2 Model and Methods

The script prompts the user to enter 3 values: the number of rows (M), the number of columns (N), and the identification cell (P). Then, the script error handles the inputs. If either M or N is less than 2, than the inputs are invalid. If not, if P is less than 1 or greater than the product of M and N, then P is invalid. If not, then finally if M, N, or P are not integers, than all inputs are invalid.

If the inputs pass error handling, then print the cell Id and the beginning of the neighbors using the following code:

```
fprintf('\nCell ID: %.0f\n', P);
fprintf('Neighbors: ');
```

Then, print the neighbors based on where the cell is. If the cell is a corner piece, there will be 3 neighbors. The cell is a corner piece if P is either 1, M*N, M, or M*N-M+1. If the cell is a side piece, there will be 5 neighbors. The cell is a side piece if P is less than M, great than M*N-M+1, or remainder of P when divided by M is 0 or 1. Else, the cell is an interior piece and there will be 8 neighbors.

2.3 Calculations and Results

When the program is executed, and the user inputs *M* value 4, *N* value 6, and *P* value 4, the following output is printed to the screen:

```
Input rows: 4
Input columns: 6
Input identification cell: 4

Cell ID: 4
Neighbors: 3 7 8
```

However, when the program is executed, and the user inputs *M* value 4, *N* value 6, and *P* value 5, the following output is printed to the screen:

```
Cell ID: 5
Neighbors: 1 2 6 9 10
```

Finally, when the program is executed, and the user inputs *M* value 4, *N* value 6, and *P* value 18, the following output is printed to the screen:

```
Cell ID: 18
Neighbors: 13 14 15 17 19 21 22 23
```

With the first example being a corner, 2nd being a side, and third being an interior, these results are in line with the expected number of neighbors.

2.4 Discussion

The printed results correctly reflect the expected number of neighbors and display the correct neighbor cells as well. This makes sense because linear indexing is an effective method to identifying neighboring cells in a 2 dimensional array.

Linear indexing can also be used for a 3 dimensional grid through similar methods. Essentially, we follow the same numbering method of a 2 dimensional grid, but for each L layer, we continuing the number, but where the 1st cell would be, the ID would be M*N+1 for the 2nd layer. Then, continue this numbering principle for the rest of the layers.

Therefore, in this 3-D grid, there would be four classifications: corners, edges, sides, and interiors. For a corner piece, there would be 7 neighbors. For an edge piece, there would be 11 neighbors. For a side piece, there would be 17 neighbors. For an interior piece, there would be 26 neighbors.