Benjamin Asdell
CEE/MAE M20
UID: 005114992 **Homework 2** October 15, 2020

1. Day of the Week Calculator

1. Introduction

   The objective of this problem is to write a script that can read a specifically formatted date (DD/MMM/YYYY) inputted by a user. The program will output the date to the command window along with its day of the week. In addition, a special message will indicate if the date is a weekend day.

2. Models and Methods

   The program begins with asking for user inputs for each part of the date. The prompt tells the user to type in a certain format (YYYY, MMM, DD). If the user fails to do so, several input verification steps stop the program from executing further. The various invalid inputs include: dates that do not exist except during leap year, dates that never exist, years before 0000 or after 9999, any parts of the date that do not follow the prompted format. If the input passes all these tests, it continues to the calculation step.
   The calculation is based on the formula given in the assignment and found at https://en.wikipedia.org/wiki/Determination_of_the_day_of_the_week. The full formula is as follows:

$$w = (d + \lfloor 2.6m - 0.2 \rfloor + y + \lfloor \frac{y}{4} \rfloor + \lfloor \frac{c}{4} \rfloor - 2c) mod 7$$

   - w: day of the week (0-6, Sunday-Saturday)
   - d: numeric value of day
   - m: shifted month $((month + 9) mod 12) + 1)$
   - c: first two digits of year
   - y: last two digits of year

   If the month is January or February, the numeric value for the year used in the calculations of y and c is decremented by one to adjust for the formula. $\lfloor \rfloor$ brackets signify the `floor` function in MATLAB.
   As a sidenote, I opted not to hard-code the months in the input validation step nor the days of the week when printing the final date to the command window. Instead, arrays are manipulated and array indexes are extracted in order to verify that the month inputted by the user is an actual month (in the array of valid months) and to assign the name of the day to the numeric result from the final calculation.

3. Calculations and Results

When the program is executed, the user is prompted for the three parts of the selected date. Inputting the date suggested by the prompt gives the following output.

```
Please enter the year as YYYY (e.g. 2008):
2008
Please enter the month as MMM (e.g. FEB):
FEB
Please enter the day as DD (e.g. 06):
06
FEB 06 2008 is a Wednesday.
```

Checking a variety of sources online will verify that this output is consistent with the true day of the week. Next is an example of an invalid input.

```
Please enter the year as YYYY (e.g. 2008):
2019
Please enter the month as MMM (e.g. FEB):
FEB
Please enter the day as DD (e.g. 06):
29
Error using hw2_005114992_p1 (line 33)
Invalid input.
```

This input fails the verification relating to nonexistent dates, specifically the fact that the selected year 2019 is not a leap year and therefore February only has 28 days. The error message tells the user that the input is invalid and exits the program. Here is another example of an input with some variation.

```
Please enter the year as YYYY (e.g. 2008):
1776
Please enter the month as MMM (e.g. FEB):
jul
Please enter the day as DD (e.g. 06):
04
JUL 04 1776 is a Thursday.
```

Checking online, this output correctly displays that the first American Independence Day was in fact a Thursday. Note that the month input is case insensitive and therefore the lowercase 'jul' still allows the program to execute as normal and output the date in uppercase.

4. Discussion

   This problem is an extension or variation of the elapsed time problem done in lecture, as it incorporate date inputs and calculations based on those inputs to output the desired information. Interestingly, the formula for calculating the day of the week requires manipulation of the input in order to execute properly, as mentioned in Section 3. The program is fairly functional for the purposes of the assignment, but could be improved by allowing the user to fix their invalid inputs without having to run the program again. Also, while the program does allow for years in the range 0000 to 9999, it is not accurate for dates preceding OCT 1582, as briefly mentioned in the assignment.

2. Neighbor Identification

1. Introduction

   The goal of this problem is to examine a matrix of varying dimensions and relay some information about the cells adjacent to a user selected cell. The program depends on linear indexing of the matrix within MATLAB to evaluate and print the indexes of those 'neighbor' cells to the command window.

2. Models and Methods

   The program starts by asking the user to input values for the number of rows and columns of the matrix. Note that the matrix is never actually constructed and does not need to be in order to find the indexes of the neighbor cells. The dimensions of the matrix are validated according to the parameters that the matrix must be at least 3x3 and M,N must be integer values.
   Several nested conditional statements are required to test the position of the selected cell relative to the selected matrix. For example, the cell at position 21 is a corner node for a 4x6 matrix, but is an interior cell for many other matrices.

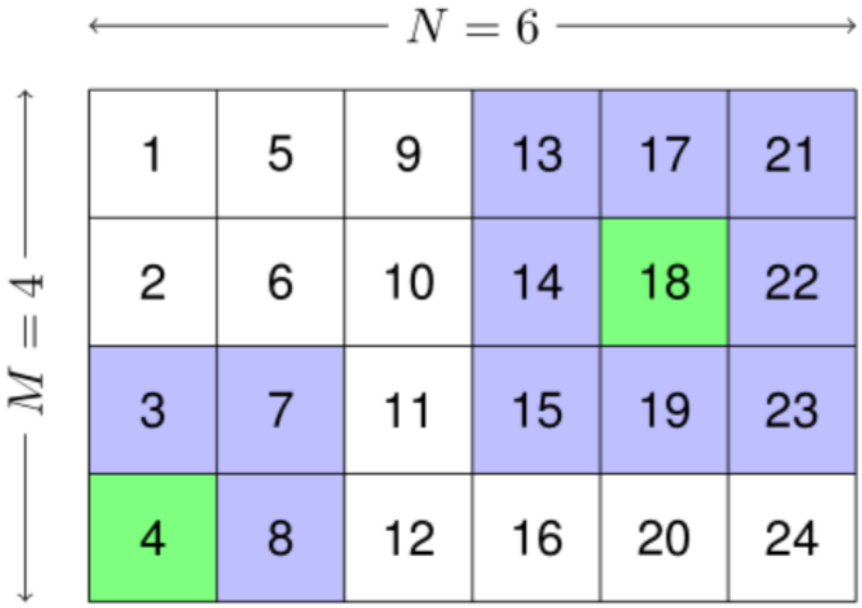

Figure 1: This diagram facilitates understanding of 'neighbor' cells and shows how MATLAB linearly indexes a matrix. Note that corner nodes (in this case, 4) only have 3 valid neighbor cells while interior cells (like 18) have all 8 valid neighbor cells.

3. Calculations and Results

When the program is run, the following is printed to the command window. For the first example, the inputs corresponding to the assignment page and Figure 1 above will be used.

```
Select the number of rows:
4
Select the number of columns:
6
Select a specific cell:
4
Corner node
     8     3     7
```

Checking with Figure 1, the program prints the correct output, indicating that the selected cell is a corner node and printing the indexes of its neighbor cells. The order in which the neighbors are displayed is irrelevant (according to the assignment). The next input is from the same 4x6 matrix, but the other example cell.

```
Select the number of rows:
4
Select the number of columns:
6
Select a specific cell:
18
    13    14    15    17    19    21    22    23
```

The output correctly displays all 8 neighbor cells since the selected cell is an interior cell. The final input is an example of an invalid input, except there is an added functionality.

```
Select the number of rows:
3
Select the number of columns:
2
M or N must be greater than or equal to 3.
Select the number of rows:
2
```

```
Select the number of columns:
3
M or N must be greater than or equal to 3.
Select the number of rows:
3
Select the number of columns:
3
Select a specific cell:
5
     1     2     3     4     6     7     8     9
```

As the assignment specifies, an error message is displayed if the user inputs a row or column dimension less than 3. However, with TA approval, the program does not immediately stop when there is an invalid input. Instead, the program tells the user the parameters for the matrix dimensions and again prompts for new dimensions. This functionality handles invalid inputs for both row and column lengths in addition to continuing until a valid input is entered. Lastly, the program correctly displays all 8 neighbors for the interior cell of the 3x3 selected matrix.

4. Discussion

This program is relatively basic as it only computes neighbors based on mathematical concepts and visualization of how the selected matrix would appear and be indexed. It can be highly versatile with a few modifications, such as constructing and displaying the actual matrix to the command window. It also is scalable for higher dimensions far beyond a 4x6 matrix. Linear indexing is an interesting concept used by MATLAB that makes this type of 'neighbor' analysis easier to visualize and understand. A modification of this script would be very useful in editing 2D pixel arrays or other similar applications involving data stored in matrix configurations.

3. Quadratic Function Extrema

1. Introduction

   The purpose of this program is to identify the absolute extrema of a user selected quadratic function on over a bounded region. The program identifies the quantity of real roots (also known as zeroes) of the function, then eliminates roots outside the selected bounds, and finally evaluates the function at the bounds and remaining roots. The minimum and maximum values of the function calculated from any of these points will be printed to the command window.

2. Models and Methods

   After taking user inputs, the input validation step ensures that the function is actually quadratic ($a > 0$). The main functionality of this program is an implementation of the quadratic formula:

   $$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

   All of the variables in the quadratic formula are inputted by the user through a direct prompt. The values correspond to the coefficients of the general form for a quadratic function:

   $$ax^2 + bx + c$$

   The bounds, also inputted by the user, are the first x-values at which the function is evaluated and possibly the only values if all the real roots of the function are outside the bounds. Note that the bounds are verified to be ordered from left to right and the program will continually prompt the user for new bounds until valid values are entered.
   The discriminant, taken from the quadratic formula to be $\sqrt{b^2 - 4ac}$, is an indicator of how many real roots the given quadratic function has. If the discriminant is positive, there will be two real roots; if negative, there are zero. If the discriminat is zero, there will only be one real root. Regardless of how many roots there are, they must be located within the bounds selected by the user in order for the program to recognize them as candidates for extrema.

3. Calculations and Results

   When the program is executed with the function $x^2 - 2x + 4$ and bounds [-1,1], the following is printed to the command window:

```
Left bound:
-1
Right bound:
1
a:
1
b:
-2
c:
4
The function is 1x^2 + -2x + 4
The minimum value over [-1, 1] is: 3.000000
The maximum value over [-1, 1] is: 7.000000
```

Graphing this function on a graphing calculator will verify that the minimum and maximum values do match the true values for this function and bounds. In this case, the maximum occurs at the left bound while the minimum occurs at the root of the function, which is also a local minimum. Here is an example of the program execution when the user enters invalid inputs at first. Similar to the previous problem, the program will not exit and instead prompt the user again.

```
Left bound:
1
Right bound:
-2
Please make sure L is less than R.
Left bound:
-2
Right bound:
1
a:
0
a should not be zero for a quadratic function.
a:
1
b:
-4
c:
-12
The function is 1x^2 + -4x + -12
```

```
The minimum value over [-2, 1] is: -15.000000
The maximum value over [-2, 1] is: 0.000000
```

These extrema can be confirmed using a graphing utility, but the main functionality of this example is the input validation. The first bounds the user inputs are out of order, and the first a coefficient entered is zero. Both of these are invalid inputs for those respective fields and the program responds by asking the user to overwrite those values with new, valid inputs.

4. Discussion

This program is a classic instance of using computer programming to expedite multi-step mathematical calculations. Computing roots and evaluating functions at x-values takes a significant amount of time on paper, so having a program like this greatly improves efficiency if the user needs to compute extrema for a large quantity of functions. This type of program is similar to those commonly found on graphing calculator websites to be used by students because it vastly reduces the amount of time needed to complete trivial single variable calculus problems like this. This program still has some flexibility, as it scales for much higher quadratic coefficients and can be modified to allow plotting of the function, bounds, and extrema.