Jared Rivera

UID: 804603106

CEE/MAE M20

January 17th, 2017

HOMEWORK 1

1 POLYHEDRON PROPERTIES

1.1 INTRODUCTION

The goal of this problem was to write a program that gives dimensions of nested platonic solids. The platonic solids are the tetrahedron, the cube, the octahedron, the dodecahedron, and the icosahedron. The geometry underlying this problem is as follows: the icosahedron is nested within the dodecahedron which is in the octahedron which is in the cube which is in the tetrahedron which is inside a unit sphere. Each platonic solid is given the maximum possible dimensions that will stay inside the corresponding solid it is nested within. The dimensions of the solids relevant to this problem are the inner radius, the outer radius, and the edge length. The dimensions are displayed in a table to six decimal places of accuracy and are printed without MATLAB's table function.

1.2 MODEL AND METHODS

The first step in building the table was to solve for the dimensions mathematically. This was done with simple algebra using the table of relationships found in the Insight Through Computing text. The starting point for this was acknowledging that since the tetrahedron was nested within a unit sphere, its outer radius was 1. From there, edge length could be calculated and thus inner radius. A similar approach was taken for the rest of the nested solids.

The second step in this problem was constructing the table. Using the **`fprintf`** function a few times creation of a title, column headers, and data entry is done with appropriate syntax as seen below:

```
fprintf('T\t\t\t%.6f\t\t%.6f\t\t%.6f\n',rt,Rt,Et);
```

Inside the single quotes is the string. The T denotes that this row is for the nested tetrahedron. The \t marking created the spacing needed to make an aesthetically pleasing table, and the \n marking prompts the next entry to be printed directly below this one. The %.6f marking denotes that a variable will be placed in that position in the string with six decimal places of precision, that variable being one of the three to the right of the string separated by commas. These variables are the dimensions.

1.3 CALCULATIONS AND RESULTS

When the script is executed, the following is printed to the command window:

```
                Dimensions of Platonic solids

Solid           Inradius            Outradius           Edge length

T               0.333333            1.000000            1.632993

C               0.192450            0.333333            0.384900

O               0.111111            0.192450            0.272166

D               0.088295            0.111111            0.079294

I               0.070164            0.088295            0.092839
```

Each solid is represented by a letter, as done in the text. T is the tetrahedron, C is the cube, O is the octahedron, D is the dodecahedron, and I is the icosahedron.

1.4 DISCUSSION

Some basic observations that can be made are about the geometric interpretation of each dimension. In each case, the inner radius is smaller than the outer radius. This makes intuitive sense as it's a shorter distance by definition. In each case, the radii also decrease with each shape. This also makes sense as each successive shape is nested within a smaller shape than its predecessor. Finally, the edge length also decreases in each case with the exception of the icosahedron within the dodecahedron. The icosahedron has a longer edge length than the dodecahedron it is nested within. This seems counterintuitive at first glance, but due to the orientation by which the shaped are nested this is the only way to maximize the size of the icosahedron.

2 ELLIPSE CALCULATIONS

2.1 INTRODUCTION

The goal of this problem was to write a program that calculates the perimeter of an ellipse using various methods. The program prints out a value of perimeter for each of the eight methods provided and also gives the 'h-value'. The h-value is an integer that tells the user how far from being a circle the ellipse is. For example, a circle will be assigned an h-value of 0, while a very flat ellipse nearing the geometry of a line will receive an h-value close to 1.

2.2 MODEL AND METHODS

The problem starts by prompting the user for the a and b values of the ellipse using the **`input`** function. Next, the h-value is calculated as it will be necessary for the perimeter calculation that comes after. h is calculated through variable assignment and use of the user input as seen below.

```
h=((a-b)/(a+b))^2;
```

Next, perimeter is estimated using the eight methods and equations proposed in the text. The first method is shown below.

```
p1=pi*(a+b);
```

Where pi is the predetermined MATLAB estimation of π. The other methods used to calculate perimeter, p2-p8, are similar but have more going on to improve accuracy. An example is shown below.

```
p8=p1*((3-sqrt(1-h))/2);
```

p8 is defined in terms of p1 in order to make the script look less messy.

Finally, the results are printed to the command window using the **`fprintf`** function in a comparable fashion to the results in problem 1.

2.3 CALCULATIONS AND RESULTS

One the user inputs the a and b values, the following is printed to the command window:

```
The eight methods of calculating perimeter yield

P1=9.42477796

P2=7.69529898

P3=7.36768785

P4=9.68839540

P5=9.68844822

P6=9.68844734

P7=9.68844820

P8=9.69428400


h=      0.111
```

This specific example uses a=1, b=2. Many of the methods return similar results to the fifth decimal place, with methods 2 and 3 being exceptions to that.

2.4 DISCUSSION

After running trials with varying a and b values, a few observations can be made about the program. First, each method returns different values. This can be credited to two sources of error. First is error in the development of the equations. Each serves to approximate the perimeter of the ellipse, so some error arises as those aren't purely derived equations. Second, error may arise from the value of `pi`. This is an approximation by MATLAB so with each calculation error is propagated.

Another observation that can be made surrounds the approximated values as the ellipse becomes "flatter", meaning a and b grow further apart. The perimeters found when a=b (a circle) are all very similar, but as a and b grow apart the values begin to differ even in the first integer, as can be seen in the example above. This means that the error of approximation between equations grows as the ellipse becomes flatter.

One method to determine which method is most accurate would be to run a test where a and b start at different values, say a=1 b=2 as in this example. Next, a is increased over multiple trials with a step size of 0.1. As a approaches b, which ever method approaches 2πr quickest is most accurate.

3 CUBIC ROOTS

3.1 INTRODUCTION

The goal of this problem was to write a program that calculates the roots of a cubic function of the form $ax^3 + bx^2 + cx + d, a \neq 0$. The program prints all four coefficients (a, b, c, d) as well as the roots (r_0, r_1, r_2).

3.2 MODEL AND METHODS

The problem starts by prompting the user for the values of all four coefficients in a similar fashion to how inputs were performed in problem 2. Next, since the equation for calculating cubic roots is very long and complex the equation was broken into multiple variables to be pieced together later on.

```
rf=2*sqrt(-p/3);                        %Front of equation
rmm=(3*q)/(2*p)*sqrt(-3/p);             %Middle of middle of equation
rm0=cos((1/3)*acos(rmm));               %Middle of equation for k=0
rm1=cos((1/3)*acos(rmm)-1*2*pi/3);      %Middle of equation for k=1
rm2=cos((1/3)*acos(rmm)-2*2*pi/3);      %Middle of equation for k=2
rb=b/(3*a);                             %Back of equation
```

The equations were then pieced together for each root.

```
r0=rf*rm0-rb;
r1=rf*rm1-rb;
r2=rf*rm2-rb;
```

Finally, the roots were printed to the command window as in problems 1 and 2.

3.3 CALCULATIONS AND RESULTS

Once the user inputs the coefficients, the following is printed in the command window:

```
a=     2.00

b=    -3.00

c=    -7.00

d=     3.00


r0=   2.61803

r1=   0.38197

r2=  -1.50000
```

The results above come from using the same conditions as outlined on the problem description.

3.4 DISCUSSION

The problem description states that the specific function above has a root at exactly -1.5, which is apparently recovered by this program. However, this actually loses accuracy at the 15th decimal place. This error is most likely due to rounding errors associated with `pi`. As it is an approximation in MATLAB and occurs frequently in the calculation process, the error due to it is propagated into any final answer derived using it.

This method actually only works in a specific case, as well, that being

$$4p^3 + 27q^2 \leq 0$$

If this condition isn't satisfied, the program will likely not know what to do with the input as undefined values will appear. This condition can be checked for before running the root calculations, though. This would be done by use of "if" statements or logical expressions. The script would read similarly to the current one, but after the calculation of p and q by the machine the if statement would be implemented. If the values met the condition of the problem, the calculation would proceed exactly as it does in the current script. If not, another print option would be printed to the command window telling the user that the program cannot calculate the roots for that specific expression, and re-prompt for input of coefficients to a new cubic.

REFERENCES

F., Van Loan Charles, and K.-Y Daisy. Fan. "Chapter 1 From Formula to Program." *Insight through Computing: A MATLAB Introduction to Computational Science and Engineering*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2010. N. pag. Print.