Zubin Mishra
UID: 604644805
CEE/MAE M20
April 11, 2017

Homework 1

1 Polyhedron Properties

1.1 Introduction

The goal in this problem is to develop a program that will calculate the edge length, inradius, and outradius of several nested polyhedrons. These calculations are done using given inradius and outradius equations. Following the calculations, results are printed to the screen and patterns will be discussed.

1.2 Model and Methods

The script first calculates the edge length and inradius of the outermost polyhedron, using 1 as the outradius:

```
E_T = 1/(sqrt(6)/4);
r_T = sqrt(6)/12*E_T;
R_T = 1;
```

The next polyhedron's outradius is the previous one's inradius, and the calculations follow from that. This is repeated for each polyhedron. One example is shown below.

```
E_C = r_T/(sqrt(3)/2);
r_C = 1/2*E_C;
R_C = r_T;
```

After all calculations, the results are printed to the command window using MATLAB's fprintf function, using '\t' and '\n' to format a table. One line is shown below.

```
fprintf('Tetrahedron\t\t%.6f\t%.6f\t%.6f\n', r_T, R_T, E_T);
```

1.3 Calculations and Results

When the program is executed, the following output is produced:

```
                    Inradius    Outradius   Edge Length
                    ________    _________   ___________

Tetrahedron         0.333333    1.000000    1.632993
Cube                0.192450    0.333333    0.384900
Octahedron          0.111111    0.192450    0.272166
Dodecahedron        0.088295    0.111111    0.079294
Icosahedron         0.070164    0.088295    0.092839
```

1.4 Discussion

One pattern that can be seen in the table of results is that the outradius of a polyhedron is the inradius of the previous polyhedron, with the exception of the first polyhedron. This is how the script was written, so this pattern serves to check that the results were found correctly. In each column, it can also be seen that entries are decreasing, with one exception (the icosahedron edge length is larger than the dodecahedron edge length. This makes sense, as each polyhedron is nested within the previous.

2 Ellipse Calculations

2.1 Introduction

The goal in this problem is to estimate the perimeter of an ellipse using several different approximations with user-specified semiaxes. Following these calculations, the results are printed to the screen and differences in approach will be discussed.

2.2 Model and Methods

The script first gets values for a and b with two calls to the input function. Once the user inputs have been acquired, they can be used to calculate h and the 8 perimeter estimation formulas. The lines for h and one of the perimeter formulas is shown below.

```
h = ((a-b)/(a+b))^2;
P3 = pi*sqrt(2*(a^2+b^2)-(a-b)^2/2);
```

After all 8 perimeter equations have been evaluated, the results are printed to the command window using the fprintf function. One of these lines is shown below:

```
fprintf('P1 = %.16f\n', P1);
```

2.3 Results and Calculations

When the program is executed, and the user inputs 1 for a and 1 for b, the following output is printed to the screen:

```
Please enter a value for A: 1
Please enter a value for B: 1
P1 = 6.2831853071795862
P2 = 6.2831853071795862
P3 = 6.2831853071795862
P4 = 6.2831853071795862
P5 = 6.2831853071795862
P6 = 6.2831853071795862
P7 = 6.2831853071795862
P8 = 6.2831853071795862
```

```
        h = 0.0000000000000000
```
All 8 methods produce the same result. However, when b is reduced to .5, the program prints the following:

```
        P1 = 4.7123889803846897
        P2 = 4.9672941328980507
        P3 = 4.8415194363533463
        P4 = 4.8441976999363439
        P5 = 4.8442241080650428
        P6 = 4.8442236720978320
        P7 = 4.8442240985830747
        P8 = 4.8471420014978506
        h = 0.1111111111111111
```

Some of the methods still produce very similar results, but there is an appreciable difference between others, such as P1 and P2. When b is reduced further to .1, the program prints the following:

```
        P1 = 3.4557519189487729
        P2 = 4.4650420927691714
        P3 = 3.9924192049131455
        P4 = 4.0582875864045613
        P5 = 4.0639272100188721
        P6 = 4.0631510072703509
        P7 = 4.0637936840138940
        P8 = 4.1901690518435490
        h = 0.6694214876033057
```

There is now an appreciable difference between several of the methods.

2.4 Discussion

As the ellipse gets flatter, there is more and more difference between the predictions. Only a few of the predictions are similar, but even they begin to vary after the thousandths place. One possible way to determine which approximation is most accurate is to use an infinite series, and comparing the predictions to the value that the series converges to.

3 Cubic Roots

3.1 Introduction

The goal in this problem is to find the roots of a cubic equation with user-specified coefficients. Once the user input has been acquired, the roots are calculated and displayed. Discrepancies between program-generated values and known solutions will be discussed.

3.2 Model and Methods

The script first obtains values for a, b, c, and d with four calls to the input function. Once the user inputs have been acquired, the roots are calculated. The lines in MATLAB are as shown below:

```
p = (3*a*c-b^2)/(3*a^2);
q = (2*b^3-9*a*b*c+27*a^2*d)/(27*a^3);

r0 = 2*sqrt(-p/3)*cos(1/3*acos(3*q/(2*p)*sqrt(-3/p))-0*2*pi/3)-b/(3*a);
r1 = 2*sqrt(-p/3)*cos(1/3*acos(3*q/(2*p)*sqrt(-3/p))-1*2*pi/3)-b/(3*a);
r2 = 2*sqrt(-p/3)*cos(1/3*acos(3*q/(2*p)*sqrt(-3/p))-2*2*pi/3)-b/(3*a);
```

After these roots are calculated, the input coefficients and the results are printed to the command window using the fprintf function. One such line is shown below:

```
fprintf('r0 = % .5f\nr1 = % .5f\nr2 = % .5f\n', r0, r1, r2);
```

3.3 Results and Calculations

When the program is executed, and the user inputs coefficients of 2, -3, -7, and 3, the following output is printed to the screen:

```
Please enter a value for A: 2
Please enter a value for B: -3
Please enter a value for C: -7
Please enter a value for D: 3
a =  2.00
b = -3.00
c = -7.00
d =  3.00

r0 =  2.61803
r1 =  0.38197
r2 = -1.50000
```

It appears that r2 is exactly -1.5, though this is not the case.

3.4 Discussion

As mentioned above, this program does not recover the root at -1.5 exactly. Instead, we get a value of -1.500000000000001. This is because MATLAB is not infinitely precise and cannot store every digit of a never-ending decimal, so information is ‘lost’ in a sense.

It is possible to construct a test to determine if all the roots are real before proceeding by using an if statement with the condition, or a while loop with the condition to continually prompt the user.