Jeffrey Ding
UID: 104928991
CEE M20
April 6th, 2018

Homework 1

1. Polyhedron Properties

1.1 Introduction

The goal in this problem is to develop a method that calculates the inradius, outradius, and edge length of 5 pre-determined nested Platonic solids with the first solid nested within the unit sphere. Then, these values will be printed into a table.

1.2 Model and Methods

The script assume that the outradius of the Tetrahedron *R4* is 1 because it is nested into a unit sphere. Under that assumption, the edge length *E4* can be calculated using a reworked version of the given formula for the Solid. In this case, the original formula is $R4 = \frac{\sqrt{6}}{4}E4$ and the reworked version of the given formula is simply $E4 = R4 \div \left(\frac{\sqrt{6}}{4}\right)$. Once calculated, this allows the inradius r4 to be calculated using the given formula for the Solid. In the first case, the formula would be $r4 = \frac{\sqrt{6}}{12}E4$.

After calculating the inradius r4, the next solid's outradius R# will be the previous solid's inradius r#. This is expressed in the second shape by:

```
R6 = r4;
```

This script follows this design until the most inward object's calculations are made. After all calculations, a table is printed using the fprintf function. The following is the first 3 lines of this table, with the rest of the lines being nearly identical replicas of the third line:

```
fprintf('Solid | inradius | outradius | edge length |\n');
fprintf('------------------------------------------\n');
fprintf('Tetra | %1.6f | %1.6f  | %1.6f    |\n', r4, R4, E4);
```

1.3 Calculations and Results

When the program is executed, the following output is printed to the screen:

```
Solid | inradius | outradius | edge length |
------------------------------------------
Tetra | 0.333333 | 1.000000  | 1.632993    |
Cube  | 0.192450 | 0.333333  | 0.384900    |
Octah | 0.111111 | 0.192450  | 0.272166    |
Dodec | 0.088295 | 0.111111  | 0.079294    |
Icosa | 0.070164 | 0.088295  | 0.092839    |
```

The outradius of a Solid is the same as the inradius of the Solid above it.

1.4 Discussion

As we go down the columns, the values decrease, which makes sense because each subsequent Platonic solid is inside the one before. This can help us check our work because the value should continue going down as we move down the columns. If a inradius or outradius value was found to be larger than the previous Solid's same value, then we can conclude that the calculations would be wrong because a Solid within another Solid cannot have a larger inradius or outradius of the Solid it is within.

The only exception to this trend is in the edge length characteristic. This also makes sense because the Solid's have different edges and number of edges. Therefore, it is possible that the edge length may be larger, although not fairly likely.

2. Ellipse Calculations

2.1 Introduction

The goal in this problem is to develop a method that calculates the perimeter of an ellipse with user-specified semiaxes *a* and *b*. The perimeter will be approximated using 8 different approximation methods. After all calculations, all recorded values will be printed out in a way to facilitate comparison.

2.2 Model and Methods

The script initially prompts the user for *a* and *b* value inputs using the following lines of code:

```
a = input('Enter "a" value: ');
b = input('Enter "b" value: ');
```

Then, the script uses the *a* and *b* values to calculate a *h* value that will be used in future calculations. The equation for h is $h = \left(\frac{(a-b)}{(a+b)}\right)^2$. Then, the script calculates using each perimeter approximation formula. The first approximation calculation code looks like this:

```
P1 = pi*(a+b);
```

The rest of the approximation follows similar variable naming rules and equations. After calculating all values, the script prints out the data in a way to facilitate comparison. The following is the first, second, and last line printed's code:

```
fprintf('a = %10.5f, b = %10.5f\n', a, b);
fprintf('P1 = %10.15f\n', P1);
fprintf('h = %10.15f\n', h);
```

The lines of code between the second and last line are nearly identical to the second line.

2.3 Calculations and Results

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

```
Enter "a" value: 1
Enter "b" value: 1
a =    1.00000, b =    1.00000
P1 = 6.283185307179586
P2 = 6.283185307179586
P3 = 6.283185307179586
P4 = 6.283185307179586
P5 = 6.283185307179586
P6 = 6.283185307179586
P7 = 6.283185307179586
P8 = 6.283185307179586
h = 0.000000000000000
```

The P# values all output the same value. However, when the user inputs *a* value 1 and *b* value 0.1, the following output is printed:

```
Enter "a" value: 1
Enter "b" value: 0.1
a =    1.00000, b =    0.10000
P1 = 3.455751918948773
P2 = 4.465042092769171
P3 = 3.992419204913146
P4 = 4.058287586404561
P5 = 4.063927210018872
P6 = 4.063151007270351
P7 = 4.063793684013894
P8 = 4.190169051843549
h = 0.669421487603306
```

In this case, P# value are not the same output.

2.4 Discussion

The change in output due to changes in *a* and *b* values is due to the ellipse no longer being a regular circle. This is indicated by the non-zero h value that is outputted. In fact, the larger the h value is, the more and more varied the P# values will become.

Basically, as the ellipse becomes “flatter”, the predictions becomes deviate further and further from each other. This is as previously stated because the *h* value is becoming larger and larger. Due to this variance, it is inevitable that all perimeter values are not guaranteed to agree with each other.

The only method I can think of to determine the most accurate approximation would be to take the integral of the ellipse formula in respect to x and find the the approximation with the smallest difference.

3. Circle-Circle Intersections

3.1 Introduction

The goal in this problem is to develop a method that calculates the area formed by two intersecting circles. Circle 1's center is located at user-specified *x1* and *y1*, which represent the x and y location of the circle, respectively. Circle 1's radius is also a user-specified *r1*. Circle 2's center is located at user-specified *x2* and *y2*, which represent the x and y location of the circle, respectively. Circle 2's radius is also a user-specified *r2*. The area will be calculated using a given set of formulas. After all calculations, all recorded values will be printed out.

3.2 Model and Methods

The script initially prompts the user for *x1, y1, r1, x2, y2* and *r2* value inputs using the following lines of code:

```
x1 = input('Enter x1 value: ');
y1 = input('Enter y1 value: ');
r1 = input('Enter r1 value: ');
x2 = input('Enter x2 value: ');
y2 = input('Enter y2 value: ');
r2 = input('Enter r2 value: ');
```

Then, the script uses the *x* and *y* values to calculate a *d* value that will be used in future calculations. The *d* value represents the separation distance between the two center of the circle. The equation for *d* is $d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$. Then, the script calculates a *c* value that represents the chord length of the intersection of the two circles. The equation for *c* is $c = \frac{1}{d}\sqrt{(-d + r1 + r2)(d - r1 + r2)(d + r1 - r2)(d + r1 + r2)}$. With the *c* and *d* values calculated, the script calculates the area of the intersection, whose equation is:

$$Area = r_1^2 cos^{-1}\left(\frac{d^2 + r_1^2 - r_2^2}{2dr_1}\right) + r_2^2 cos^{-1}\left(\frac{d^2 - r_1^2 + r_2^2}{2dr_2}\right) - \left(\frac{d}{2}\right)c$$

After calculating all values, the script prints out the user-inputted values and the area of the intersection using the following code:

```
fprintf('\nx1 = %5.2f\n', x1);
fprintf('y1 = %5.2f\n', y1);
fprintf('r1 = %5.2f\n', r1);
fprintf('x2 = %5.2f\n', x2);
fprintf('y2 = %5.2f\n', y2);
fprintf('r2 = %5.2f\n', r2);
fprintf('\nArea = %5.4f\n', Area);
```

3.3 Calculations and Results

When the program is executed, and the user inputs *x1, y1, r1, x2, y2,* and *r2* values 0, 0, 4, 3, 4, and 3, respectively, the following output is printed to the screen:

```
Enter x1 value: 0
Enter y1 value: 0
Enter r1 value: 4
Enter x2 value: 3
Enter y2 value: 4
Enter r2 value: 3

x1 =  0.00
y1 =  0.00
r1 =  4.00
x2 =  3.00
y2 =  4.00
r2 =  3.00

Area = 6.6417
```

3.4 Discussion

The theoretical area of the intersection for two circles with non-zero radii separated by a distance d = r1 + r2 would be 0. This is due to the circles not intersecting and creating a lens. The edge of the circles are touching but not overlapping. My code recovers this result exactly because it calculates that c and any other calculation depending on d to become 0. With d = r1 + r2, the c value will result to a value of 0. With these values, area will be calculated to be 0.

When d + r1 < r2, the area becomes a complex number consisting of an imaginary number. Under this inequality, the geometrical interpretation of it would be that there are two circles that are not intersecting at all. The correct area calculation in this case would be zero.