Kepler’s Third Law Homework:

Algorithm

The starting point of this program were the following three Kepler Laws of planetary motion:

1. All planets move in an elliptical with the Sun on the focus of the Ellipse.
2. The line joining a planet to the sun sweeps out equal areas in equal time.
3. If T is the period, and A the semi-major axis, then T^2/A^3 is constant.

The program was to investigate the third law. The program to model the orbits was similar to the one for circular motion:

Where:  
 – Acceleration

– Constant, around

– Components of the radius

– Radius, or distance from the sun

– Velocity after the time interval

– Velocity before time interval

– Radius after the time interval

– Time interval

The difference in this program is that the radius is not constant rather it changes so that the path becomes elliptic. The formula for this phenomena is:

Where:

– Angle

– Semi-major axis

– Eccentricity of the ellipse

Since the planets each take different time to complete the orbit, the program will run until the coordinates are back to where it started. The program will save this time and find the following value:

Code

// Here are all the libraries that will be used.

#include <iostream>

#include <iomanip>

#include <cmath>

#include <fstream>

#include <stdio.h>

using namespace std;

// Here is the function that will run the planetary

void KeplersLaw(long double e, long double a, long double vx, long double vy)

{

ofstream Kepler;

Kepler.open("OrbitFile.txt");

// Here are all the constants that will be used.

long double r, rx, ry = 0, ax, ay, tt = 0, T = 0, r\_0 = 0, mu,

dt = .0001, Period = 0, Pi = 3.1415926535898;

mu = 4 \* pow(Pi, 2);

// Here is the title for all the values that will be outputted.

Kepler << "Radius (AU) X position (AU) Y position (AU) Time (Yr)" << endl;

// Setting the precision for stylistic reasons.

Kepler << showpoint << setprecision(8);

// Setting the radius depending on the particular values of each planet.

r = (a \* (1 - pow(e, 2))) / (1 - (e \* cos(2 \* Pi \* T)));

rx = r;

r\_0 = r;

// Output the first value.

Kepler << r << " " << rx << " " << ry << " " << T << endl;

// Loop which will use the Euler Method to calculate the positions at each time period.

while (Period == 0)

{

// Start the time index at 1.

T = tt \* dt + dt;

// recalculate the radius.

r = (a \* (1 - pow(e, 2))) / (1 - (e \* cos(2 \* Pi \* T)));

// Find the x components of the acceleration and velocity to find it's position.

ax = - (mu \* rx) / pow(r, 3);

vx = vx + ax \* dt;

rx = rx + vx \* dt;

// Find the y components of the acceleration and velocity to find it's position.

ay = -(mu \* ry) / pow(r, 3);

vy = vy + ay \* dt;

ry = ry + vy \* dt;

//Have a condition which ends the orbit close to where it started, with some error.

if (tt > 100)

{

if (rx >= r\_0 - 0.01 && rx <= r\_0 + .01)

{

if (ry >= -0.01 && ry <= .01)

{

Period = 1;

}

}

}

// increase the counter, different from the index, but related.

tt++;

// Display the values.

Kepler << r << " " << rx << " " << ry << " " << T << endl;

}

// Calculate T^2/A^3.

long double KeplerValue = pow(T, 2) / pow(a, 3);

// Display to the user the final T^2/A^3 value.

Kepler << "\n The Value for T^2/A^3 is: " << KeplerValue << endl;

cout << KeplerValue;

// Close the file.

Kepler.close();

};

int main()

{

// Redefine Pi since it needs to be redefined.

long double Pi = 3.1415926535898;

// For Venus.

cout << "The Value T^2/A^3 for Venus is: ";

KeplersLaw(0.0067, 0.7233, 0, 1.188 \* 2 \* Pi);

rename("Orbitfile.txt", "Venus.txt");

cout << endl;

//For Earth

cout << "The Value T^2/A^3 for Earth is: ";

KeplersLaw(.0167, 1, 0, 2 \* Pi);

rename("Orbitfile.txt", "Earth.txt");

cout << endl;

// For Mars, although it does not work.

/\*

cout << "The Value T^2/A^3 for Mars is: ";

KeplersLaw(0.09341233, 1.52366231, 0, 0.875 \* 2 \* Pi);

rename("Orbitfile.txt", "Mars.txt");

cout << endl;

\*/

// For Jupiter.

cout << "The Value T^2/A^3 for Jupiter is: ";

KeplersLaw(0.0484, 5.2034, 0, 0.453 \* 2 \* Pi);

rename("Orbitfile.txt", "Jupiter.txt");

cout << endl;

// For Saturn.

cout << "The Value T^2/A^3 for Saturn is: ";

KeplersLaw(0.05415, 9.5371, 0, 0.336 \* 2 \* Pi);

rename("Orbitfile.txt", "Saturn.txt");

cout << endl;

// End the program.

return 0;

}

How to Run the Code

This code is written in C++ so in order to run it, the g++ compiler should be used. This compiler should already be in Omega. The file extension that seemed to work best is the .C extension. Note that the file creates files but in terms of user interaction, the program only lets the user know what the program does and when it completed the run.

Results and Analysis

The following orbits were created using the program and Excel:

The Calculated Period wasand the calculated third law value was . the actual values were supposed to be and the value was .

The Earth’s calculated period was and . the actual values were and

Jupiter’s calculated period was and the value was . The actual values are and .

Saturns’s calculated period was and the value was . The actual values are and .

The difference in the values is not big at all and was within 5% of the “correct” values.

Conclusion

This program did a decent job at modeling the orbits of the planets, and although it was not perfect, it did give very close values to those standard. I still do not know why Mars did not work well, but for the rest, the program did a decent job.