Final Exam

Algorithm

The purpose of this program is to investigate the population equation using the Euler method. The starting point is:

Where:

– Rate of change of the population

– Constants

– Population at a given time

Solving the differential equation via Wolfram Alpha:

Where:

– Constant of integration

Since this was a nonlinear differential equation, dealing with it analytically would be unpleasant. Luckily there is a simpler way to find the solution. This program will compute the differential equations numerically using the following:

Where:

– Population after the time interval

– Population before the time interval

– Time interval

If is sufficiently small enough, it will approximate the solution to the differential equation with the particular constant values with precision. The time interval will be since this is not too big or not too small.

This program will investigate a couple different cases and will create a set of data points which will be graphed in Excel. Unfortunately Excel will not be able to fit most of these graphs with equations since the general solution is of quotients of exponentials and Excel cannot handle these forms.

Code

// These are all the libraries that will be used in this program.

#include <iostream>

#include <iomanip>

#include <cmath>

#include <fstream>

#include <stdio.h>

using namespace std;

// Here is the function that will be used to calculate all the population problems.

// The general form of the equation is dN/dt = a\*N-b\*N^p.

// The function will work with different a, N, b, p, dt, and total time of the program.

void PopulationGrowth(double a, double N, double b, double p, double dt, int Time)

{

// A file will be created where all the data will be stored.

ofstream Population;

Population.open("File.txt");

// This is where the values will be initialized.

double N\_t, T = 0, dndt, TrueTime;

//since the total time will be dependent on the dt, the "True time" must be considered.

// In this case it is the total time divided by dt.

TrueTime = Time / dt;

// The initial value is the value that is inputted.

N\_t = N;

// Here is where the header and the starting value will be displayed.

Population << "Time Population" << endl;

Population << T << " " << N\_t << endl;

// This is where the Euler method will be performed.

// The iterations will be done via a for loop.

for (int tt = 1; tt <= TrueTime; tt++)

{

// Using the dN/dt and the time interval, the differential equations will be solved.

dndt = a \* N\_t - b \* pow(N\_t, p);

N\_t = N\_t + (dndt) \* dt;

// The time counter will be updated.

T = T + dt;

//Displaying the values.

Population << T << " " << N\_t << endl;

}

// Closing the file.

Population.close();

}

int main()

{

// The time interval that will be used will be .01, since it is sufficiently small but not excessive.

// All the functions will rename the file in accordance to their paticular values.

// Part A

// Letting a = 10, b = 0, and N\_0 = 30.

PopulationGrowth(10, 30, 0, 2, .01, 10);

rename("File.txt", "Part A.txt");

// Part B

// Letting a = 10, b = 0.1, and N\_0 = 30.

PopulationGrowth(10, 30, .1, 2, .01, 10);

rename("File.txt", "Part B.txt");

// Part C

// Letting a = 10, b = 10, and N\_0 = 30.

PopulationGrowth(10, 30, 10, 1.5, .01, 10);

rename("File.txt", "Part C.txt");

// Part D 1/2

// Assume that the power p in the N^p component is greater than 2.

PopulationGrowth(10, 30, .1, 2.5, .01, 10);

rename("File.txt", "Part D, 1 of 2.txt");

// Part D 2/2

// Assume that the power p in the N^p component is lesser than 2.

PopulationGrowth(10, 30, .1, 1.5, .01, 10);

rename("File.txt", "Part D, 2 of 2.txt");

// This is the cases for bacteria.

// Part E 1/2

// a = 2, b = 0, N\_0 = 1, and the power of N^p will be 2.

PopulationGrowth(2, 1, 0, 2, .01, 10);

rename("File.txt", "Part E, 1 of 2.txt");

// Part E 2/2

// a = 2, b = .1, N\_0 = 1, and the power of N^p will be 2.

PopulationGrowth(2, 1, .1, 2, .01, 10);

rename("File.txt", "Part E, 2 of 2.txt");

// Let the user know that the program was successful.

cout << "The program has completed and all the data points have been calculated." << endl << 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

# Part A

Solve the equation by Euler method by taking . Also, take and .

Since , the solution gives a differential equation that is unbounded, and that is exactly what happens with the graph. Without the term, this solution diverges to infinity as time goes on.

Note that since the nonlinear term in the differential equation is zero in this case, this becomes a linear differential equation and the solution is a simple exponential.

# Part B

Solve again by Euler method with same values but now take .

# Compare the Results of Part A and B

Since there now is a term, the solution will now be restricted by it and in this case allows it to reach an “equilibrium state”. Now recalling the general solution, it can be seen that at a certain time, the term is dominant, but after some time, the term has an effect and cancels with the term to reach “equilibrium”. Now this is only for this particular set of values, and as will be seen, different values give different results.

# Part C

Consider the situation when .

From the rate equation, it can be seen that the term has a product with it so if both and terms are equal, the term will “overpower” the term and there will be a decrease in the rate thus a decrease in population. Now physically speaking a negative population cannot occur so it makes sense that the graph does not cross the x-axis, but it would have been thought that the population would have reached zero at some point.

This was not the case and in fact this graph had a horizontal asymptote at . This is due to the exponential terms in the general equation which cancel out when the population reaches .

Lastly, if this was human population, after the population reached , it would end with that one.

# Part D

In a community where food and other resources are abundant, do you think the form of the second term is good enough (i.e. the power of N)? What logical choice you think the power of N (in the second term) should be?

To solve this problem, consider the following modification:

Now if conditions are good, then the population should grow more and have a higher “equilibrium” population.

Either or have to lead to this happening.

Both cases will be computed and the graphs are shown below. Note that

The reason for this difference in power is because if it were too big, the results would be ridiculous, and if the difference in power was too small, then the two would not have a significant difference.

It can be seen that the case where gave the results that were desired. Thus if life is peachy, the in will be less than 2.

# Compare the Results with A and B

Since there is a nonzero term, the graph will be bounded, unlike that of A. since the value is less than that of part B, the “equilibrium” point is much higher.

# Part E

In case of bacterial growth, where every bacterial is divided into two to increase its population, the first term will be represented by , and lets’ assume that the second term remains the same, the modified equation will then look like,

Solve the equation with the same values of b as A and B, and take in this case.

# Question: Will there be a negative growth rate?

In the previous cases, the only times that there was a negative growth rate was when the term “overpowered” the term. This makes sense if the original rate equation is considered.

Consider:

In this case the term does not exceed the term thus the growth will be positive until “equilibrium”.

# Part E 1/2

Without the term, the graph becomes unbounded. Notice that the growth is not that great at the beginning, but after a while, the growth goes crazy. This is most likely due to the fact that since the starting amount was only 1, it would take a while for it to start going crazy.

# Part E 2/2

Now with the term, the graph does eventually reach “equilibrium”.

Conclusion

The Population equation, as simple as it looks, is not a simple equation to handle analytically since it is nonlinear, but with the Euler method, particular numerical solutions can easily be achieved.

One this that was surprising was that for all the cases, the graphs never reached zero and tended to sit at some positive value. This is probably due to the continuous nature of the differential equation and it in the case where the population hit 1, it is obvious that that population has reached its end point, assuming it is not self-reproducing.

Also interesting was the fact that depending on how big or small the values of the coefficients and , the solution could end up being quite different, like in the cases where and .

In the end, this project demonstrates the power of the Euler method with nonlinear differential equations, which are not the tamest of beast. The Euler method really is a powerful tool.