A Comparison of Numerical Solutions to the General Neutron Point Reactor Kinetics Equations

New Mexico AiS Challenge
Final Report
April 7, 2004

Team 65
Santa Fe High School

Team Members:
Leif Hopkins

Teacher:
Anita Gerlach

Project Mentor:
H. Omar Wooten

Table of Contents

Executive Summary

The problem that is the force behind this project is the general neutron point reactor kinetics equations. This system of two differential equations describe the rate of change, with respect to time, of the neutron population and what is called the *precursor density*, or the amount of precursor neutrons, in a nuclear reactor. Though solvable by several methods, I chose two numerical integration methods to solve these differential equations and make plots of the data. From these data, I was able to compare to analytical and experimental results in order to determine the overall effectiveness and convergence of each said numerical method.

Written in the C++ computer language, two programs were constructed in order to utilize two numerical methods in solving this system. These two methods were the Euler method of integration and the Runge Kutta method of integration. Both of these methods are ideal for use in programming and solving many physical systems; the neutron point reactor kinetic equations are no different.

Of course, the solvability of this system by the use of numerical methods does not guarantee the accuracy of other more analytical results. With each program constructed, a user interface was programmed into each that would allow the user to configure the interval of integration, the time step, and so forth. Several different time steps were used, varying in accuracy between a tenth and a ten thousandth of a decimal, with accuracy of data approaching up to 16%.

As the programs were utilized and the results were attained, we became enlightened to several points regarding the numerical methods. We learned both about their ease and relative effectiveness and their inherent downfalls, both of which we found important for any person wishing to use such methods.

Report

a. *Problem Statement*

In nuclear physics, for purposes of optimizing the performance and regulating the safety of a nuclear reactor, it is important that the nuclear reactor run at a level that is described as *critical.* The maintaining of this criticality is vital both for the maximum power output of the nuclear reactor, and the minimizing of the radiation given out by the reactor.

To describe the state of criticality, we must first understand the nature of nuclear power. Nuclear power is based upon a process called *fission*, a process in which a neutron approaches a fissile isotope, and its very proximity, as the neutron slows near the atom, causes it to split into two or more pieces, generating *fission products* and generating even more neutrons called *prompt* and *delayed* neutrons, named aptly for the time interval between the initial split and their appearance. These neutrons collide with hydrogen in the water surrounding the fuel pins, depositing their energy and increasing the temperature of the water causing it to boil. The heat of the water, or rather the steam, is then used to power turbines and generate power.

This process would seem hap hazardous at first. With the flurry of neutrons scattering everywhere and splitting atoms at all points, it is very easy to attain a state that is called *supercritical,* or an out of control reactor (an event that occurred at Chernobyl). Thus said, it is important to maintain the critical state, which can now be defined as a steady state system, or, one where the average number of neutrons remains constant in time.

The population of the neutrons in the system is, therefore, quite important knowledge when analyzing this factor. Likewise is the *precursor* density, or the population of fission products that result in delayed neutrons. These equations have been found, by methods beyond the scope of this explanation, to be:

$$dn(t)/dt = (\rho(t) - \beta)/\Lambda)\, n(t) - \Sigma(\lambda_i C_i, i = 1, m)$$

and

$$dC_i(t)/dt = (\beta_i/\Lambda) n(t) - \lambda_i C_i(t))$$

where n(t) is the neutron population at time t, and $C_i(t)$ is the ith precursor density. The other constants involved in this equation describe more quantities in nuclear physics, all of which are beyond the scope of this explanation. Briefly though, they relate to probabilities of neutron interactions and yields (fractions) that particular precursors appear per fission. The second equation requires particular explanation; it is the combination of the six precursor groups into one equation. Typically the equations number seven, being the initial neutron population density and the six precursor groups (characterized by their rates of radioactive decay), but the precursor groups can be defined and approximated using only one equation by determining effective quantities averaged over the six groups.[1]

Though the above equations are used for these purposes, their use in my project was primarily their solvability via numerical integration methods. These equations have been analytically solved previously, so by employing numerical integration methods to our solving of these systems I was able to compare my accuracy and the accuracy of the numerical methods to these results. The goal is to determine how closely brute force Runge-Kutta and Euler methods compare to analytical results.

b. *Method*

The method, or rather methods, utilized in numerically integrating these equations were the Euler method and the fourth order Runge Kutta method. These methods are based upon different approaches[2], and are used, along with the differential equation and an initial value, to approximate the solution of the anti-derivative of the differential equation.

I used a predetermined set of values of the constants of these equations[3], leaving the only unknowns to be n(t) and C(t). These, then, were found using the numerical integration methods. The methods were utilized by writing the formula for each method, adapted to the general neutron point reactor kinetics equations, in a C++ computer program that could iterate the formula a large number of times and determine results very quickly. As I was not analyzing the

[1] For more explanation of this combination, see Dunderstadt, J., Hamilton, L., "Nuclear Reactor Analysis," John Wiley & Sons, NY 1976

[2] The Euler method uses the slopes of tangent lines to approximate solutions, while the Runge Kutta Method uses midpoints.

[3] These constants are defined in Appendix B, within the source code. See References for source.

equations for results regarding criticality, only the sub-critical values for reactivity were used. This provides a general framework for comparing to reference data for the purpose of testing these numerical methods.

I would typically use the program, varying the length of the interval I wished it to approximate the value of the solution. The more of these approximations I attained, the better my results would become as the methods would approach greater accuracy. I was then able to analyze my results and compare them to values in literature. The simulated experiment is the introduction of a small positive reactivity to a sub-critical reactor. Reactivity is simply described as a change in the reactor composition that would increase (positive) or decrease (negative) the reactor's critical state.

c. *Results*[4]

With the numerical methods employed, it was important to use a varying set of time steps in order to gain an understanding of each method, as well as to achieve maximum accuracy when interpreting the results.

With the Euler method, grossly inaccurate figures, along with accurate figures were attained. The maximum accuracy of the Euler method attained, at t = 1 second, with a time step of .0001 seconds was 16% error (actual normalized neutron population was 2.098 at 1 second, Euler at .0001 attained 1.854). This is quite good, however, when compared with the results of a .1 time step, massively gross error was attained (here, Euler gave -4.62×10^{12} as the neutron population at 1 second).

The Runge Kutta Method proved more successful. Though neither method worked with a time step less than .01 seconds, Runge Kutta was shown to have great accuracy. At .01-second time steps, Runge Kutta showed a 16% inaccuracy. Smaller time steps, neither method showed a percent error any less than 16%.

[4] For plots and tables of these data, See Appendix A. n(t) is plotted for each method at each significant time step, 0.1 and 0.0001.

d. *Conclusions*

This project has shown that because of their relative ease and accuracy, numerical methods are excellent for the general description of mathematically defined physical systems, such as the neutron point reactor kinetics equations. However, for use in situations requiring very precise results, it would seem more efficient to develop alternative and more analytical solutions to said systems (such as those described by Kinard et. al.). If one were to describe this system, for example, the numerical methods employed here would likely provide skewed data, especially over long periods. This could result in failures to anticipate supercritical conditions in a nuclear reactor, which is, needless to say, undesirable.

e. *Original Significant Achievement*

By comparing the numerical solutions of these equations to more effective solutions, we are able to better classify the use of these methods in solving the equations. For the purposes of a nuclear plant, for example, a very fine tune must be kept on the state of the nuclear reactor. In such a case, a 16% error could result in massive damages. As such, numerical methods of this kind are not effective in situations requiring more accurate results. Nevertheless, this project has shown that because of their ease (compared to analytical solutions), they are excellent for achieving a general understanding of neutron population and precursor density within a nuclear reactor. These methods can also be implemented into more formal quick computer calculations in which neutron multiplication is desired provided that a 16% inaccuracy is acceptable.

f. *References*

The following sources were consulted while performing this project:

[1] Chapra, S., “Numerical Methods for Engineers,” McGraw Hill, 2002

[2] Dunderstat, J., Hamilton, L. “Nuclear Reactor Analysis,” John Wiley & Sons, New York, 1976

[3] Kinard, Matthew, Allen, E.J., “Efficient numerical solutions of the point kinetics equations in nuclear reactor dynamics,” *Annals of Nuclear Energy*, 31, 1039, 2004.

g. *Acknowledgments*

I would like to thank Mrs. Anita Gerlach for her assistance on this report, and also, a very special thank you goes to Mr. Omar Wooten, who even though in the middle of writing his doctoral thesis still managed to provide massive amounts of assistance on this project.

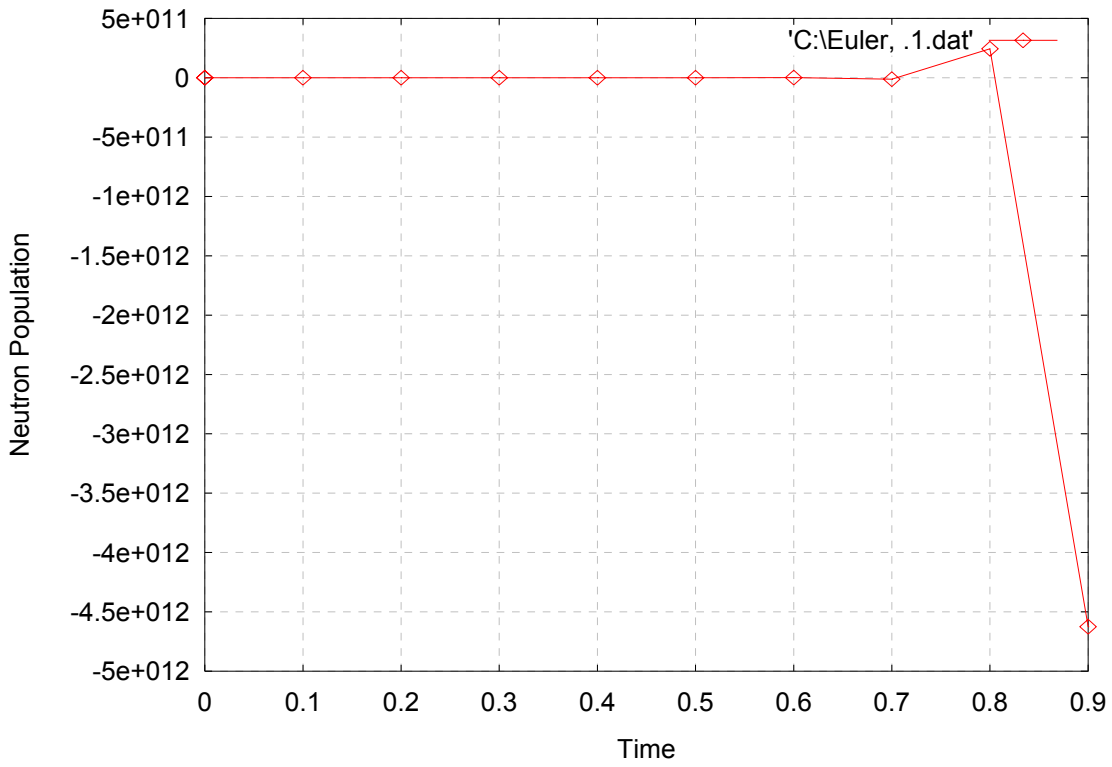
Appendices

a. *Plots of Data / Table of Results*[5]

ACTUAL NEUTRON POPULATION AT 1 SECOND = 2.2098.

Time Step (Number of Intervals)	Runge Kutta Value at 1 s	Euler Value at 1 s	Percent Error
0.1	-2.14211E+22	-4.63E+12	--------
0.01	1.854	0.99578	16% to 55%
0.001	1.854	1.85484	16% to 16%
0.0001	1.854	1.85485	16% to 16%

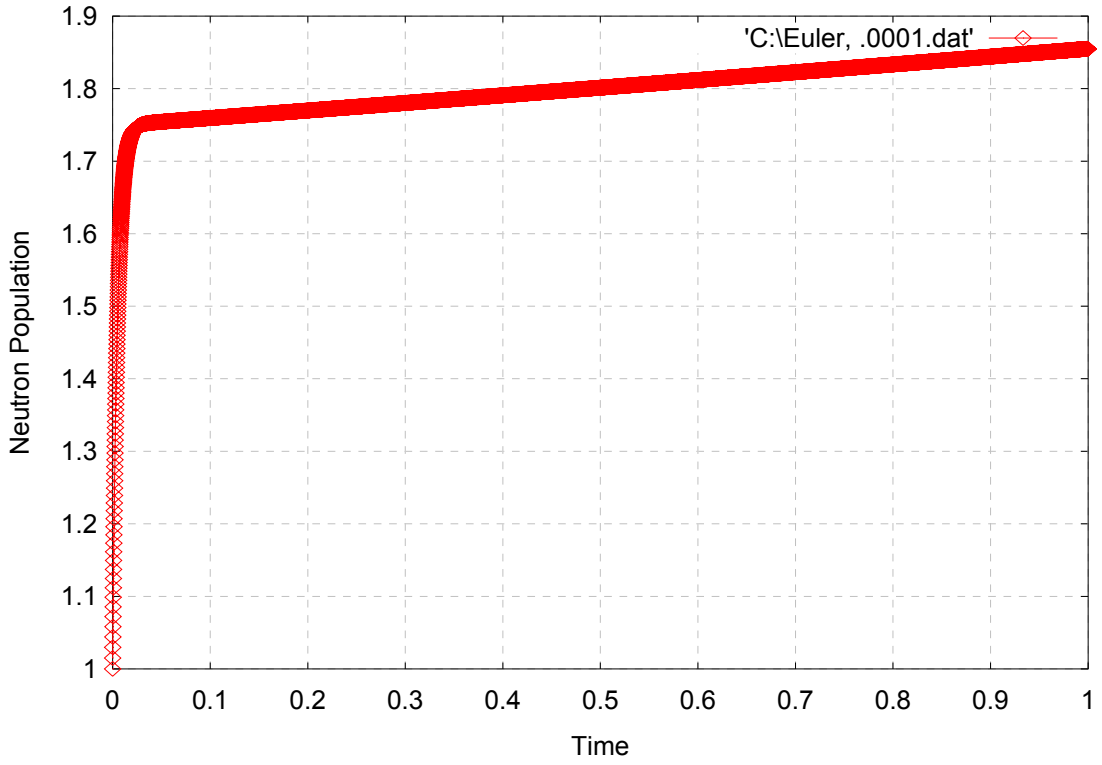
Euler, n(t), Time Step 0.1 seconds



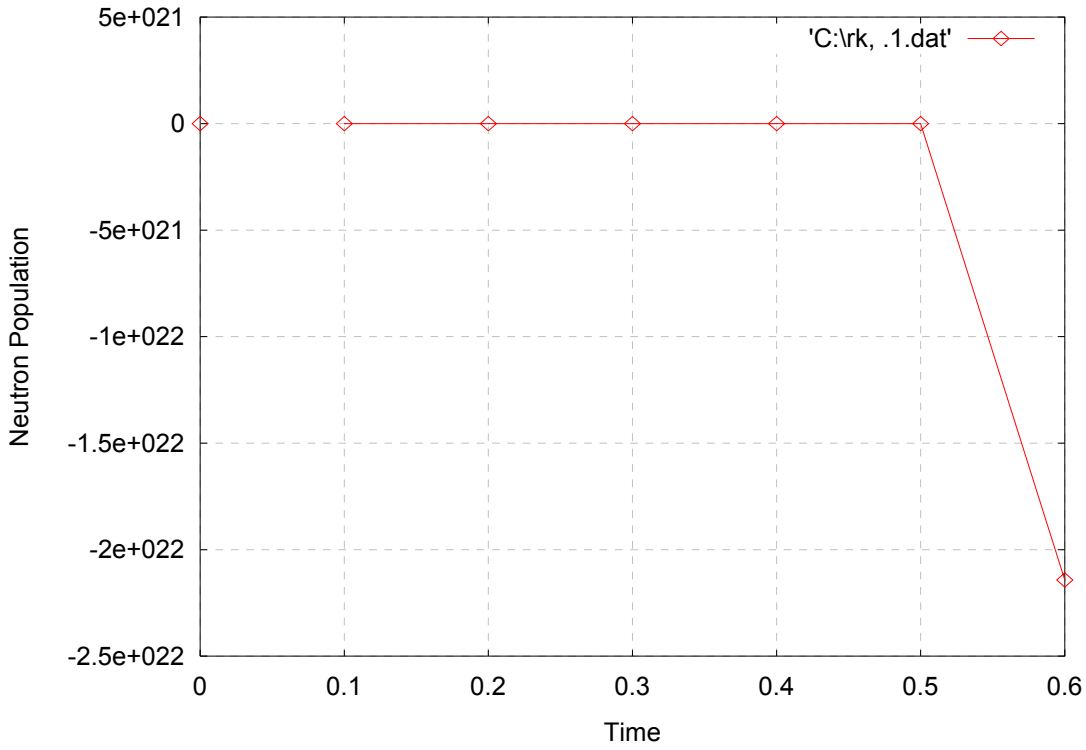
We see immediately the blatant inaccuracy of this plot.

[5] Table made in Microsoft Excel; all plots made in GnuPlot.

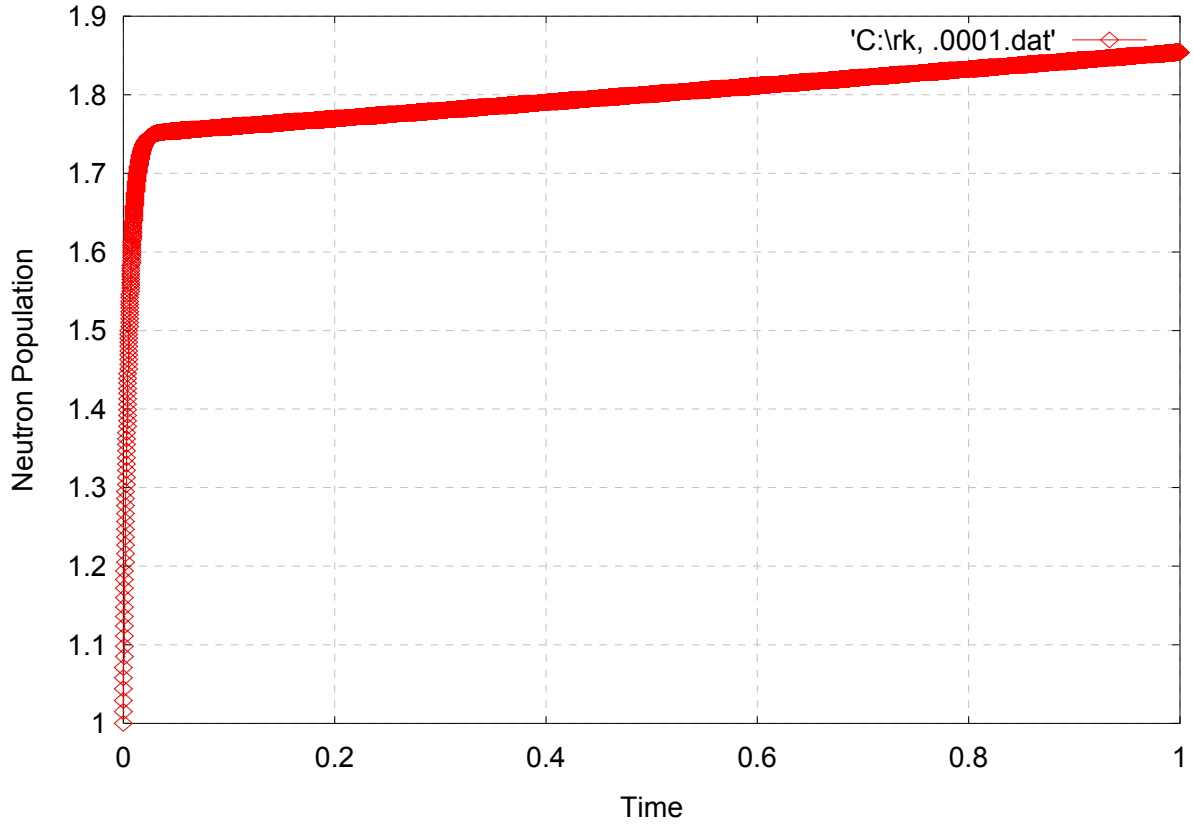
Euler, n(t), Time Step 0.0001 seconds



Runge Kutta, n(t), Time Step 0.1 seconds



Runge Kutta, n(t), Time Step 0.0001 seconds



b. *Source Code*

EULER PROGRAM

```
#include <iostream.h>
#include <math.h>
#include <fstream.h>

//global variables
const float beta=0.007;
const float lambda=0.078413045;
const float LAMBDA=0.00002;
const float rho=0.003;

//Function declaration
float F(int, float, float);

int main()
{
    int i, j, k;         //Counting Variables
    float to, no, co;    //Inital time, Neutron population, and Precursor
Concentration
    float h;             //Timestep
    float tF;            //Upper bound
    int   N;             //Number of Intervals
    float n, c;          //Current Neutron pop. and Precursor Con.

      //Output to file

    ofstream outdata;
    outdata.open("Euler2.dat");

      //User Interface

      cout<<"This program will use the Euler method of integration to"<<endl;
      cout<<"numerically solve two of the seven Neutron point reactor
equations."<<endl<<endl;

      cout<<"Inital time?"<<endl;
      cin>>to;
      cout<<"Final Time?"<<endl;
      cin>>tF;
      cout<<"Number of Intervals?"<<endl;
      cin>>N;
      cout<<"Intial neutron population?"<<endl;
      cin>>no;

      //Determine Precursor concentration and Timestep; output to file

    co=beta/(lambda*LAMBDA)*no;

    h=(tF-to)/N;

    outdata<<to<<"\t\t"<<no<<"\t\t"<<co<<endl<<endl;

      //Loop to iterate Euler method
```

```
    for(i=0; i<N; i++)
    {
        to=i*h;
        n=no+F(1, no, co)*h;
        c=co+F(2, no, co)*h;
        co=c;
        no=n;

        outdata<<to<<"\t\t"<<no<<"\t\t"<<co<<endl;
    }


    return 0;

}

//Function: Neutron Population and Precursor Density

float F(int j, float y1, float y2)
{
    float result;
    if(j==1)
    {
        result= (rho-beta)/(LAMBDA)*y1+lambda*y2;
        return result;
    }
    else if(j==2)
    {
        result= (beta/LAMBDA)*y1-lambda*y2;
        return result;
    }

}
```

RUNGE-KUTTA PROGRAM

```
//    This code applies 4th order Runge Kutta
//    to a system of 2 equations, the general
//    point reactor kinetic eqns.

#include<iostream.h>
#include<fstream.h>


// global variables
float beta = 0.007;
float lambda = 0.077848507;
float LAMBDA = 2E-5;
float rho = 0.003;

float k11, k12, k21, k22, k31, k32, k41, k42;
float y1, y2;

//Functions Declaration
float f1(float, float, float);
```

```
float f2(float, float, float);

int main()
{
      float tLow, tHigh, h;               //Lower and Upper Limits of
Integration, Timestep
      int   tInt, i,j,k;                  //Number of Intervals, counting
variables

      float n,c,t,   no, co, to;          //Time, Precursor Density, Neutron
Population, and the Intial Values of each


      //Output data to file

      ofstream outData;
      outData.open("rk.dat");
      outData.setf(ios::fixed);
      outData.precision(3);

      //User Interface

      cout<<"This program will use the Runge-Kutta Method of
Integration"<<endl;
      cout<<"to numerically integrate the general Neutron point reactor
equations."<<endl<<endl;

      cout<<"Intial Time?"<<endl;
      cin>>tLow;
      cout<<"Final Time?"<<endl;
      cin>>tHigh;
      cout<<"Number of Intervals?"<<endl;
      cin>>tInt;
      cout<<"Intial Neutron Population?"<<endl;
      cin>>n;

/*

  Recommended Values

      tLow = 0.0;
      tHigh = 1;
      tInt = 1000;
      n = 1.0;
*/
      c = beta/(lambda*LAMBDA);
      t=tLow;

      //Determine Values of Precursor Density and Timestep

      co = c;
      no = n;
      to = tLow;

      h = (tHigh - tLow)/tInt;

      //Output to file
```

```
	outData<<to<<"\t"<<no<<"\t"<<co<<endl<<endl;


	//Loop Runge Kutta Method

	for(i=0; i<tInt; i++)
	{
		k11 = f1(to, no, co);
		k12 = f2(to, no, co);

		n = no + k11*h/2;
		c = co + k12*h/2;
		t = to + h/2;

		k21 = f1(t, n, c);
		k22 = f2(t, n, c);

		n = no + k21*h/2;
		c = co + k22*h/2;
		t = to + h/2;

		k31 = f1(t, n, c);
		k32 = f2(t, n, c);

		n = no + k31*h;
		c = co + k32*h;
		t = to + h;

		k41 = f1(t, n, c);
		k42 = f2(t, n, c);

		no = no + (k11 + 2*k21 +2*k31 + k41)*h/6;
		co = co + (k12 + 2*k22 +2*k32 + k42)*h/6;
		to = to +h;

		outData<<to<<"\t"<<no<<"\t"<<co<<endl;
	}

	return 0;
}


//Neutron Population Function

float f1(float t, float n, float c)
{
	float N;

	N = (rho-beta)/LAMBDA*n + lambda*c;

	return N;
}

//Precursor Density Function

float f2(float t, float n, float c)
```

```
{
      float C;

      C = beta/LAMBDA*n - lambda*c;

      return C;
}
```