Numerical Methods

Lab 8 - Solving simultaneous ODEs and 2^{nd} order ODEs with RK-4 method

Jan 2022

Submitted by

Aabhusan Aryal

076BCT001

Submitted to

Department of Applied Sciences and

Chemical Engineering, Pulchowk Campus

÷ Solution to simultaneous first order ODE using RK-4 method:-

Given two first order ODEs

$$y' = \frac{dy}{dx} = f1(x, y, z)$$

and, $$z' = \frac{dz}{dx} = f2(x, y, z)$$

with initial conditions $y(x_0) = y_0$ and $z(x_0) = z_0$,

we can work out y_{i+1} and z_{i+1} corresponding to $x_{i+1} = x_i + h$ with the help of the RK-4 method as:-

$$K_1 = h \times f1(x_i, y_i, z_i),$$

$$l_1 = h \times f2(x_i, y_i, z_i);$$

$$k_2 = h \times f1(x_i + h/2,\ y_i + k_1/2 + z_i + l_1/2),$$

$$l_2 = h \times f2(x_i + h/2,\ y_i + K_1/2 + z_i + l_1/2);$$

$$K_3 = h \times f1(x_i + h/2,\ y_i + K_2/2,\ z_i + l_2/2),$$

$$l_3 = h \times f2(x_i + h/2,\ y_i + K_2/2,\ z_i + l_2/2);$$

$$K_4 = h \times f1(x_i + h,\ y_i + K_3,\ z_i + l_3)$$

$$l_4 = h \times f2(x_i + h,\ y_i + K_3,\ z_i + l_3)$$

and, $$k = \frac{K_1 + 2K_2 + 2K_3 + K_4}{6}, \quad l = \frac{l_1 + 2l_2 + 2l_3 + l_4}{6}$$

Then, $y_{i+1} = y_i + k$ and $z_{i+1} = z_i + l$

In this way, various points of the curves y and z can be found.

÷ Solution to second order ODE using RK-4 method:-

Given a second order ODE

$$y'' = \frac{d^2y}{dx^2} = f(x, y, y'); \text{ with } y(x_0) = y_0 \text{ and } y'(x_0) = y'_0$$

Let $y' = z$. Then, $y'' = z'$

$\Rightarrow y'(x_0) = y'_0 = z(x_0) = z_0$

Then, $y' = f_1(x, y, z) = z$

and $z' = f_2(x, y, z) = f(x, y, z)$

So, if we plug these values in the previous case of Solving simultaneous ODEs, we can easily solve a Second order ODE by the same method as discussed before.

Algorithm :-

1) Define $f1(x, y, z)$ and $f2(x, y, z)$ where $f1$ returns y' in case of simultaneous equn^s and returns z in case of 2^{nd} order ODE. Also, $f2(x, y, z)$ returns z' in case of simultaneous ODE and returns y'' in case of 2^{nd} order ODE.

2) Take input for x_0, y_0, z_0 (initial conditions), x_n and n.

3) Set step size $h = (x_n - x_0)/n$

4) Loop n times :-

   4.1) Calculate k1, l1, k2, l2, k3, l3, k4, l4 with formulae defined in theory

   4.2) set k = (k1 + 2k2 + 2k3 + k4)/6 and
   set l = (l1 + 2l2 + 2l3 + l4)/6

   4.3) increment y0 by k, z0 by l and x0 by h,
   i.e. y0 += k; z0 += l; x0 += h

   4.4) print (x0, y0, z0)

5) Stop

-:- Source Code

```cpp
#include <iostream>
using namespace std;

bool selection;
enum ODE{
    simul=0, secondOrder
};

// y' = f1(x,y,z)
float f1(float x, float y, float z){
    if (selection == ODE::simul)
        return 3*x+y-z;
    return z; // since f1(x,y,z) = z for second order...
}

// z' = f2(x,y,z)
float f2(float x, float y, float z){
    if (selection == ODE::simul)
        return 2*x-y+z;
    return x+2*y+z; // f2(x,y,z) for 2nd order...
}

int main(){
    cout<<"Press 0 if you want to solve simultaneous ODE
or press 1 if you want to solve second order ODE:";
    cin >> selection;
    float x0, y0, z0, xn, yp0;
    int n;
    if (selection == ODE::simul){
        cout<<"Enter x0, y0, z0, xn and n: ";
        cin >> x0>>y0>>z0>>xn>>n;
    }
```

```
else {
    cout<<" Enter x0, y0, y'0, xn and n : ";
    cin >> x0>>y0 >> yp0 >> xn >> n;
    z0 = yp0;
}
// Setting h= interval
Cout << "The points on the required curve are : "<<endl;
cout<< "(" << x0<< "," << "," << z0 << ")\n";

for (int i=0; i<n; i++) {
    float k1 = h* f1(x0,y0,z0);
    float l1 = h* f2( x0,y0,z0);
    float k2 = h* f1(x0+h/2, y0+k1/2, z0+l1/2);
    float l2 = h* f2(x0+h/2, y0+k1/2, z0+l1/2);
    float k3 = h* f1 (x0+h/2, y0+k2/2, z0+l2/2);
    float l3 = h*f2(x0+h/2, y0+k2/2, z0+l2/2);
    float k4 = h*f1(x0+h, y0+k3 , z0+l3);
    float l4 = h * f2(x0+h, y0+k3, z0+l3);

    float k = (k1+ 2*k2 + 2*k3 +k4) / 6;
    float l = (l1+ 2*l2 + 2xl3 + l4) / 6;

    y0 +=k;
    z0 += l;
    x0+ = h;

    Cout<<"(" << x0 <<"," <<y0<<"," <<z0 << ")\n";
}
return 0;
}
```

Scanned with CamScanner

:- Source Code :- [Polynomial Curve Fitting]

```
#include <iostream>
# include <iomanip>
# include <math.h>
using namespace std;

enum aug {
    left=0, full, right
};

void log( float **arr, int n , aug p) {
  // prints the matrix to console
  int leftlim, rightlim
  if (p== aug::left) { leftlim=0; rightlim = n; }
  else if (p== aug::right) { leftlim= n ; rightlim= n+1 ; }
  else { leftlim=0; rightlim= n+1 ; }

  for (int i=0; i<n; i++){
      cout<< "|\t";
      for (int j= leftlim; j<rightlim ; j++) {
          if (p== aug::full && j == n) cout<< "|\t";
          cout<<fixed << setprecision (2) << arr[i][j] << "\t";
      }
      cout<< "|" << endl;
  }
  cout << endl ;
}

int main() {
    int deg, n;
    cout<< "Enter the degree of polynomial and the number
  of input points: ";
    cin>> deg >> n;

    float *x= new float[n];
    float *y= new float[n];
```

```
//Taking input for given points
for (int i=0; i<n; i++) {
    cout<<"Enter x["<<i<<"] and y["<<i<<"]: ";
    cin >> x[i] >> y[i];
}

//Creating the required augumented matrix
float **matrix = new float *[deg+1];

for (int i=0; i<deg+1; i++) {
    matrix[i] = new float [deg+2];
    for (int j=0; j<deg+2; j++){
        float c=0;
        for (int k=0; k<n; k++) {
            if (j<=deg) c += pow(x[k], i+j);
            else c += pow(x[k], i) * y[k];
        }
        matrix[i][j] = c;
    }
}

//logging the array
cout<<"The augumented matrix is :"<<endl;
log(matrix, deg+1, aug::full);

// Performing Gauss Jordan to solve the aug. matrix
n = deg+1;
for (int j=0; j<n; j++) { //For each column
    if (i==j) continue; //ignoring diag elements
    if (matrix[j][j] == 0){
        log(matrix, n, aug::full);
        throw "The pivot element is 0";
    }
    float ratio = matrix[i][j] / matrix[j][j];
    for ( int k=0; k<n+1; k++){
        matrix[i][k] -= ratio * matrix[j][k];
    }
```

```
cout << "The final array is: " << endl;
log (matrix, n, aug::full);

cout<<" Finally, the required polynomial is: \n";
for ( int i = 0; i<n; i++) {
    float coeff = matrix[i][n] / matrix[i][i];
    cout<< "y= " << coeff << " x^ " << i;
    if (i != n-1) cout << " + ";
  }
}
```

Outputs

Simultaneous ODEs:

1. $y' = 3x + y - z$ and $z' = 2 * x - y + z$

```
Press 0 if you want to solve simulODE or press 1 if you want to solve second order ODE:0
Enter x0, y0, z0, xn and n: 0 0 0 10 10
The points on the required curve are:
(0,0,0)
(1,1.75,0.75)
(2,10.5,-0.5)
(3,53.25,-30.75)
(4,319,-279)
(5,2130.75,-2068.25)
(6,14749.5,-14659.5)
(7,103002,-102880)
(8,720678,-720518)
(9,5.0443e+006,-5.0441e+006)
(10,3.53095e+007,-3.53093e+007)
```

2. $y' = x + y + z$ and $z' = x$

```
Press 0 if you want to solve simulODE or press 1 if you want to solve second order ODE:0
Enter x0, y0, z0, xn and n: 0 1 2 2 4
The points on the required curve are:
(0,1,2)
(0.5,3.11719,2.125)
(1,7.08673,2.5)
(1.5,14.2719,3.125)
(2,26.9199,4)
```

3. $y' = x$ and $z' = x$

```
Press 0 if you want to solve simulODE or press 1 if you want to solve second order ODE:0
Enter x0, y0, z0, xn and n: 0 0 0 1 5
The points on the required curve are:
(0,0,0)
(0.2,0.02,0.02)
(0.4,0.08,0.08)
(0.6,0.18,0.18)
(0.8,0.32,0.32)
(1,0.5,0.5)
```

2nd order ODE:

1. $y'' = x + y' + 2y$

```
Press 0 if you want to solve simulODE or press 1 if you want to solve second order ODE:1
Enter x0, y0, y'0, xn and n: 0 0 0 2 5
The points on the required curve are:
(0,0,0)
(0.4,0.0117333,0.0938667)
(0.8,0.111776,0.472989)
(1.2,0.46428,1.42986)
(1.6,1.41553,3.63305)
(2,3.72269,8.5808)
```

2. $y'' = 6y - 3xy'$

```
Press 0 if you want to solve simulODE or press 1 if you want to solve second order ODE:1
Enter x0, y0, y'0, xn and n: 0 1 0.1 0.2 2
The points on the required curve are:
(0,1,0.1)
(0.1,1.04005,0.701462)
(0.2,1.1404,1.30587)
```

3. $y'' =- xy' - y$

```
Press 0 if you want to solve simulODE or press 1 if you want to solve second order ODE:1
Enter x0, y0, y'0, xn and n: 0 1 0 0.3 3
The points on the required curve are:
(0,1,0)
(0.1,0.995012,-0.0995013)
(0.2,0.980199,-0.19604)
(0.3,0.955998,-0.286799)
```

Discussion and Conclusion

Hence RK4 method was applied to solve first order simultaneous ODEs and the same concept was extended and applied to solve second order ODEs. This was achieved by making the function `f1(...)` return z in case of 2nd order ODE.

The code was written in C++ and was compiled using g++. Expected outputs were obtained without errors.