TUT10

1.cpp

```cpp
#include <iostream>
#include<bits/stdc++.h>

using namespace std;

double dNdt(double t) {
    return 3 * t * t;
}

// Using Midpoint method
vector<pair<double, double>> midpointMethod(double t0, double N0, double h,
double t_end) {
    vector<pair<double, double>> results;
    double t = t0, N = N0;
    results.push_back({t, N});

    while (t < t_end) {
        double k1 = dNdt(t);
        double k2 = dNdt(t + h / 2);
        N += h * k2; // Use k2 for the midpoint correction
        t += h;
        results.push_back({t, N});
    }
    return results;
}

//  Using Richardson's extrapolation
vector<pair<double, double>> richardsonsExtrapolation(double t0, double N0,
double h, double t_end, double tol) {
    vector<pair<double, double>> results;
    double t = t0, N = N0;
    results.push_back({t, N});

    while (t < t_end) {
        double h1 = h;
        double h2 = h / 2;
```

```
        double N_h1 = N + h1 * dNdt(t + h1 / 2);


        double N_h2_1 = N + h2 * dNdt(t + h2 / 2);
        double N_h2_2 = N_h2_1 + h2 * dNdt(t + h2 + h2 / 2);


        double N_richardson = (4 * N_h2_2 - N_h1) / 3;

        double error = abs(N_richardson - N_h1);
        if (error > tol) {
            h /= 2;
            continue;
        }

        N = N_richardson;
        t += h;
        results.push_back({t, N});
    }
    return results;
}

// Analytical solution
double analyticalSolution(double t) {
    return t * t * t;  // Assuming C = 0
}

int main() {
    cout<<"For my Case I have R parameter is 9 so I have used Mid point and
Richardson's Method";
    double t0 = 0.0, N0 = 0.0;
    double t_end = 0.5, h = 0.1;
    double tol = 0.001;

    auto midpointResults = midpointMethod(t0, N0, h, t_end);

    auto richardsonResults = richardsonsExtrapolation(t0, N0, h, t_end,
tol);

    cout << fixed;
    cout << "t\t     \tMidpoint
\tRichardson\tAnalytical\tError(MidPt)\tError(Richarsn)\n";
```

```
    for (size_t i = 0; i < midpointResults.size(); ++i) {
        double t = midpointResults[i].first;
        double midpointN = midpointResults[i].second;
        double richardsonN = i < richardsonResults.size() ?
richardsonResults[i].second : midpointN;
        double analyticalN = analyticalSolution(t);
        cout << t << "\t" << midpointN << "\t" << richardsonN << "\t" <<
analyticalN << "\t\t"
             << abs(midpointN - analyticalN) << "\t" << abs(richardsonN -
analyticalN) << "\n";
    }

    return 0;
}
```

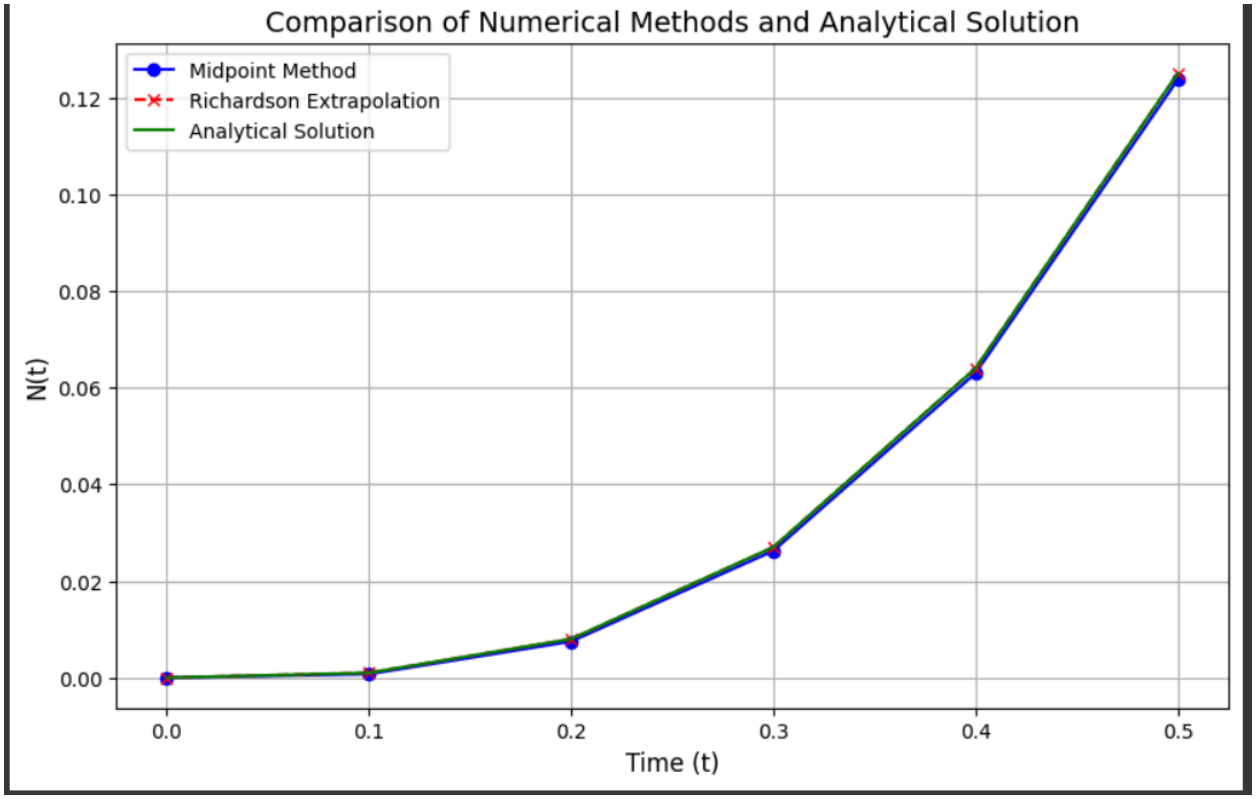
1a

Output Clear

```
For my Case I have R parameter is 9 so I have used Mid point and Richardson's Method
t           Midpoint    Richardson  Analytical  Error(MidPt)    Error(Richarsn)
0.000000    0.000000    0.000000    0.000000        0.000000    0.000000
0.100000    0.000750    0.001000    0.001000        0.000250    0.000000
0.200000    0.007500    0.008000    0.008000        0.000500    0.000000
0.300000    0.026250    0.027000    0.027000        0.000750    0.000000
0.400000    0.063000    0.064000    0.064000        0.001000    0.000000
0.500000    0.123750    0.125000    0.125000        0.001250    0.000000


=== Code Execution Successful ===
```

1b



2.cpp

```cpp
#include <iostream>
#include<bits/stdc++.h>
using namespace std;

double dYdx(double x, double Y, double lambda) {
    return -lambda*Y + (1 + lambda) * cos(x) - (1 - lambda) * sin(x);
}

vector<pair<double, double>> eulerExplicit(double x0, double Y0, double h,
double x_end, double lambda) {
    vector<pair<double, double>> results;
    double x = x0, Y = Y0;
    results.push_back({x, Y});

    while (x < x_end) {
```

```
        Y += h * dYdx(x, Y, lambda);
        x += h;
        results.push_back({x, Y});
    }
    return results;
}

double analyticalY(double x) {
    return sin(x) + cos(x);
}

int main() {
    cout<<"For my Case I have R parameter is 9 so I have used Euler
explicit";
    double x0 = 0.0, Y0 = 1.0;
    double x_end = M_PI, h = M_PI / 4;
    vector<double> lambdas = {2 / M_PI, 8 / M_PI};

    for (double lambda : lambdas) {
        auto results = eulerExplicit(x0, Y0, h, x_end, lambda);

        cout << fixed << setprecision(6);
        cout << "\nLambda = " << lambda << "\n";
        cout << "x\t \tEuler Explicit \t\tAnalytical\tError\n";
        for (const auto &res : results) {
            double x = res.first, Y = res.second;
            double analytical = analyticalY(x);
            cout << x << "\t" << Y << "\t\t" << analytical << "\t" << abs(Y
- analytical) << "\n";
        }
    }

    return 0;
}
```

2a

Output Clear

```
For my Case I have R parameter is 9 so I have used Euler explicit
Lambda = 0.636620
x       Euler Explicit      Analytical  Error
0.000000    1.000000        1.000000    0.000000
0.785398    1.785398        1.414214    0.371185
1.570796    1.599806        1.000000    0.599806
2.356194    0.514505        0.000000    0.514505
3.141593    -0.853468       -1.000000   0.146532

Lambda = 2.546479
x       Euler Explicit      Analytical  Error
0.000000    1.000000        1.000000    0.000000
0.785398    1.785398        1.414214    0.371185
1.570796    1.043029        1.000000    0.043029
2.356194    0.171573        0.000000    0.171573
3.141593    -1.282294       -1.000000   0.282294


=== Code Execution Successful ===
```

2b

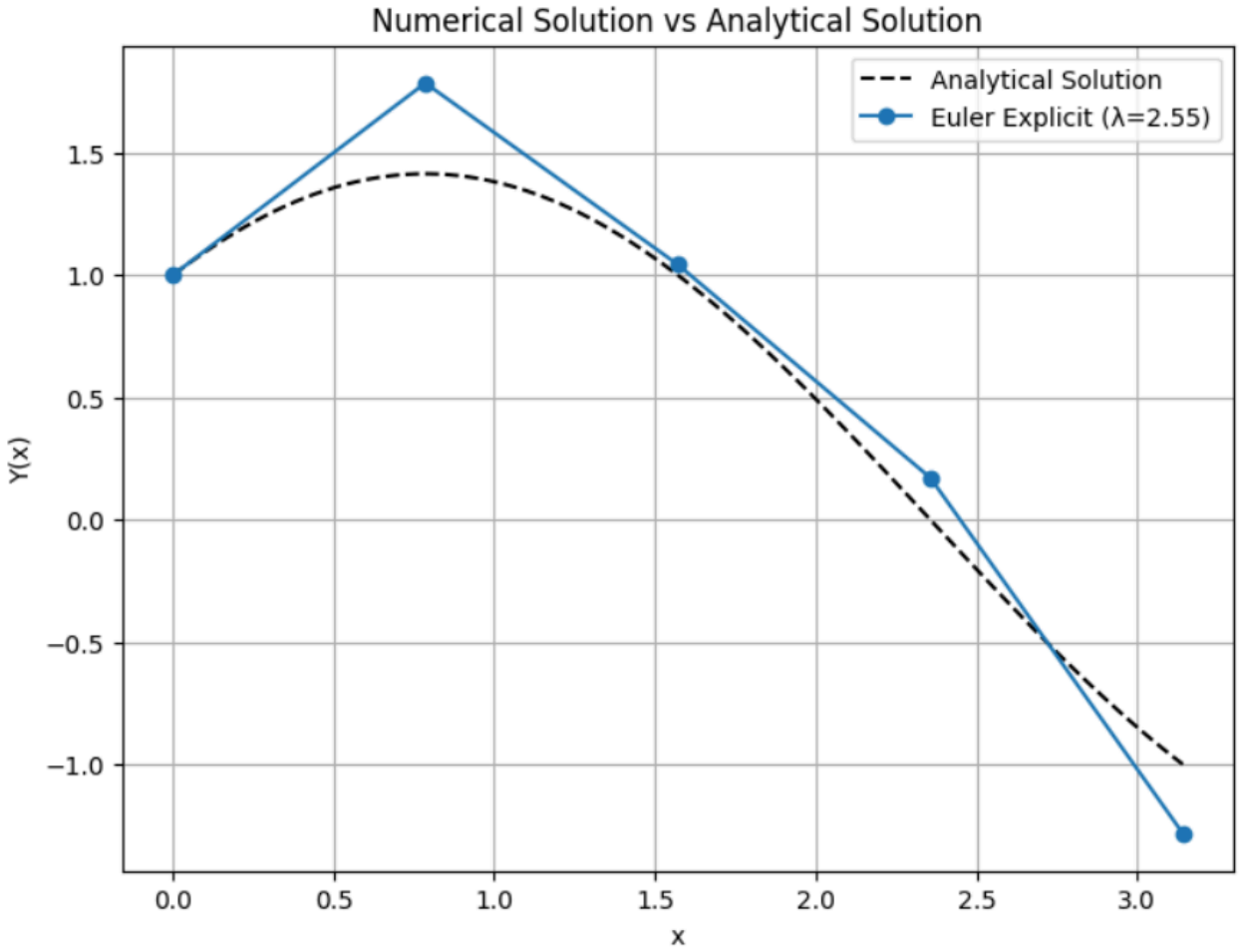
Output Clear

```
For my Case I have R parameter is 9 so I have used Euler explicit
Lambda = 0.636620
x       Euler Explicit      Analytical  Error
0.000000    1.000000        1.000000    0.000000
0.785398    1.785398        1.414214    0.371185
1.570796    1.599806        1.000000    0.599806
2.356194    0.514505        0.000000    0.514505
3.141593    -0.853468       -1.000000   0.146532

Lambda = 2.546479
x       Euler Explicit      Analytical  Error
0.000000    1.000000        1.000000    0.000000
0.785398    1.785398        1.414214    0.371185
1.570796    1.043029        1.000000    0.043029
2.356194    0.171573        0.000000    0.171573
3.141593    -1.282294       -1.000000   0.282294


=== Code Execution Successful ===
```

2c


Numerical Solution vs Analytical Solution
Analytical Solution
Euler Explicit (λ=2.55)
Y(x)
x
1.5
1.0
0.5
0.0
−0.5
−1.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0