Used Formulas

For finite difference or, forward euler,

$$x''(t_n) = \frac{x_{n+1} + x_{n-1} - 2x_n}{\varepsilon^2}$$

$$\text{i.e., } x_{n+1} = \varepsilon^2 x''(t_n) + 2x_n - x_{n-1}.$$

And

$$x_1 = x_0 + \varepsilon x'(0)$$

For backward euler,

$$x''(t_{n+1}) = -4\pi^2 x_{n+1} = \frac{x_{n+1} + x_{n-1} - 2x_n}{\varepsilon^2}$$

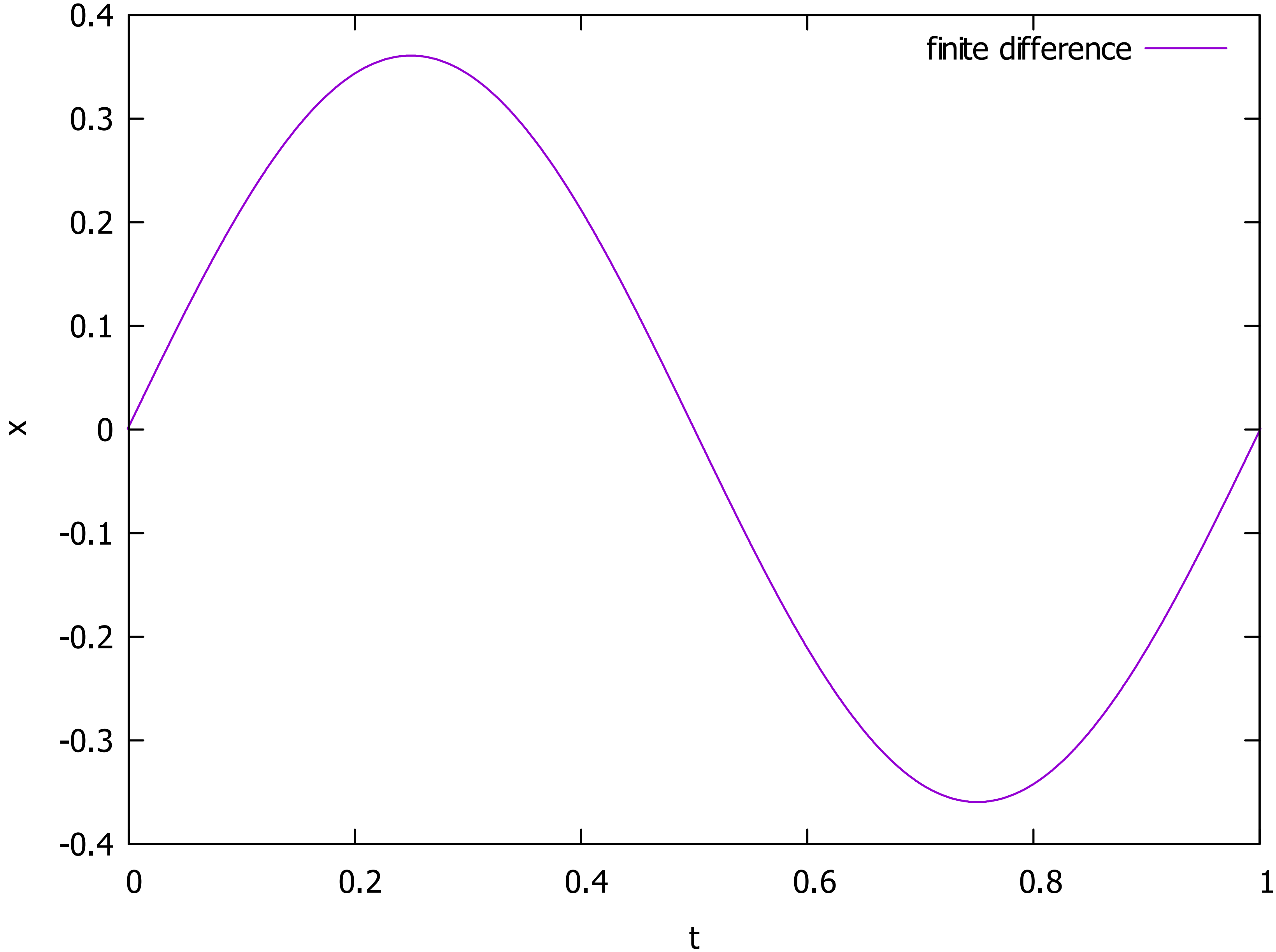
$$\text{i.e., } x_{n+1} = \frac{2x_n - x_{n-1}}{1 + 4\pi^2\varepsilon^2}.$$

While the exact solution is,

$$x(t) = \frac{x'(0)}{2\pi}\sin(2\pi t).$$

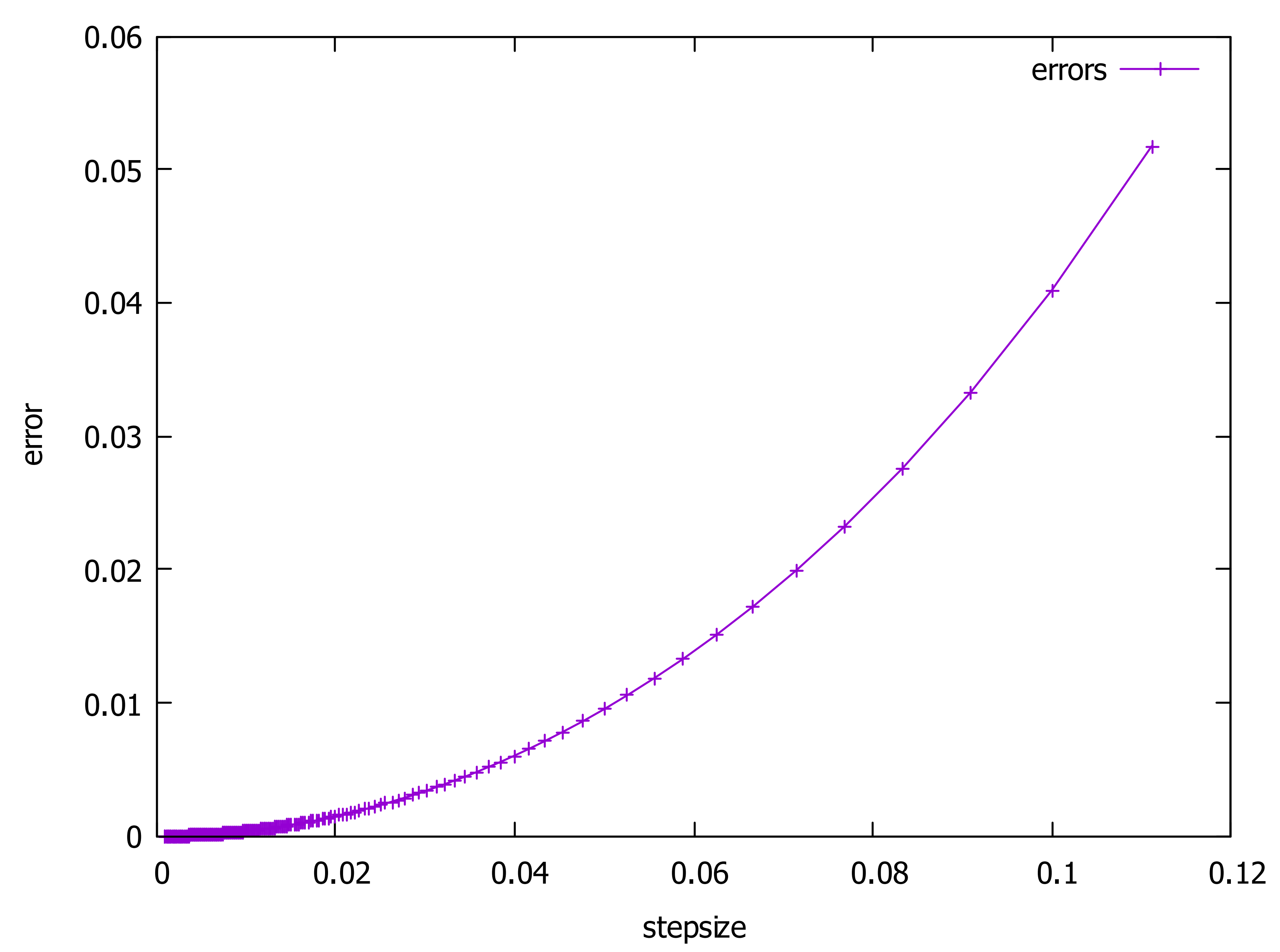
```cpp
#include <iostream>
#include <fstream>
#include <vector>

//constant expressions appearing in the code
constexpr double PI = 3.14159265;   //pi
constexpr double rollnum = 2.26121014;  //my roll number

typedef double state_type;  //data type definition for dependant variable, just a double here - should be std::vector<> for multidimensional case
typedef std::vector<double> solution;   //data type definition for storing the solutions

//This is the differential Equation
void System(state_type& x, state_type& d2xdt2){
    d2xdt2 = - 4.0 * PI * PI * x;
}

//This sets the initial values x_0 and x_1 given the values of x(0) and x'(0) and step size
void initsolver(solution& x_values, const state_type& x0, const state_type& dx0dt, const double dt){
    x_values.push_back(x0); //store x(0)
    x_values.push_back(x0 + dx0dt * dt);    //store x_1 = x(0) + x'(0) * dt
}

//The stepper function, calculates x_{n+1} given the differential equation, list of past values and step size
void finite_diff_step(void (*System)(state_type& x, state_type& d2xdt2), solution& x_values, const double dt){
    state_type result = 0;  //temporary variable for storing x_{n+1}
    System(x_values[x_values.size()-1], result);    //calculate x''_{n}
    result *= dt * dt;  //multiply by step_size^2
    result += 2 * x_values[x_values.size()-1] - x_values[x_values.size()-2];    //add 2 * x_{n} - x_{n-1}. The final expression becomes x_{n+1} = x''_{n} * step_size^2 + 2 * x_{n} - x_{n-1} which is obtained from the expression drawn on whiteboard.
    x_values.push_back(result); //store x_{n+1} in the list of solutions
}


int main(){
    solution x_values;  //variable for storing the solutions
    int STEPS = 1000;   //define number of steps
    double dt = 1.0/STEPS;  //define stepsize
    initsolver(x_values, 0, rollnum, dt);   //initilize the list of solutions with x_{0} and x_{1}
    for (size_t i = 0; i < STEPS - 1; i++) { //step through the total number of steps and store the results
```

```
        finite_diff_step(System, x_values, dt);
    }

    std::ofstream outfile;  //save the results in a file named "solution for 1000 steps.txt"
    outfile.open("solution for 1000 steps.txt", std::ios::out | std::ios::trunc );
    for (size_t i = 0; i < x_values.size(); i++) {
        outfile << 0 + i * dt << "\t" << x_values[i] << "\n";
    }
    outfile.close();
}
```


finite difference
x
t
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
0
0.2
0.4
0.6
0.8
1

```cpp
#include <iostream>
#include <fstream>
#include <functional>
#include <algorithm>
#include <vector>
#include <cmath>

//constant expressions appearing in the code
constexpr double PI = 3.14159265;   //pi
constexpr double rollnum = 2.26121014;  //my roll number

typedef double state_type;  //data type definition for dependant variable, just a double here - should be std::vector<> for multidimensional case
typedef std::vector<double> solution;   //data type definition for storing the solutions

//This is the differential Equation
void System(state_type& x, state_type& d2xdt2){
    d2xdt2 = - 4.0 * PI * PI * x;
}

//This sets the initial values x_0 and x_1 given the values of x(0) and x'(0) and step size
void initsolver(solution& x_values, const state_type& x0, const state_type& dx0dt, const double dt){
    x_values.push_back(x0); //store x(0)
    x_values.push_back(x0 + dx0dt * dt);    //store x_1 = x(0) + x'(0) * dt
}

//The stepper function, calculates x_{n+1} given the differential equation, list of past values and step size
void finite_diff_step(void (*System)(state_type& x, state_type& d2xdt2), solution& x_values, const double dt){
    state_type result = 0;  //temporary variable for storing x_{n+1}
    System(x_values[x_values.size()-1], result);    //calculate x''_{n}
    result *= dt * dt;  //multiply by step_size^2
    result += 2 * x_values[x_values.size()-1] - x_values[x_values.size()-2];    //add 2 * x_{n} - x_{n-1}. The final expression becomes x_{n+1} = x''_{n} * step_size^2 + 2 * x_{n} - x_{n-1} which is obtained from the expression drawn on whiteboard.
    x_values.push_back(result); //store x_{n+1} in the list of solutions
}

//function for calculating the exact solution, x(t) = x'(0) / (2 * pi) * sin(2 * pi * t)
double exact_solution(double t){
    return rollnum / (2.0 * PI) * sin(2.0 * PI * t);
}

```

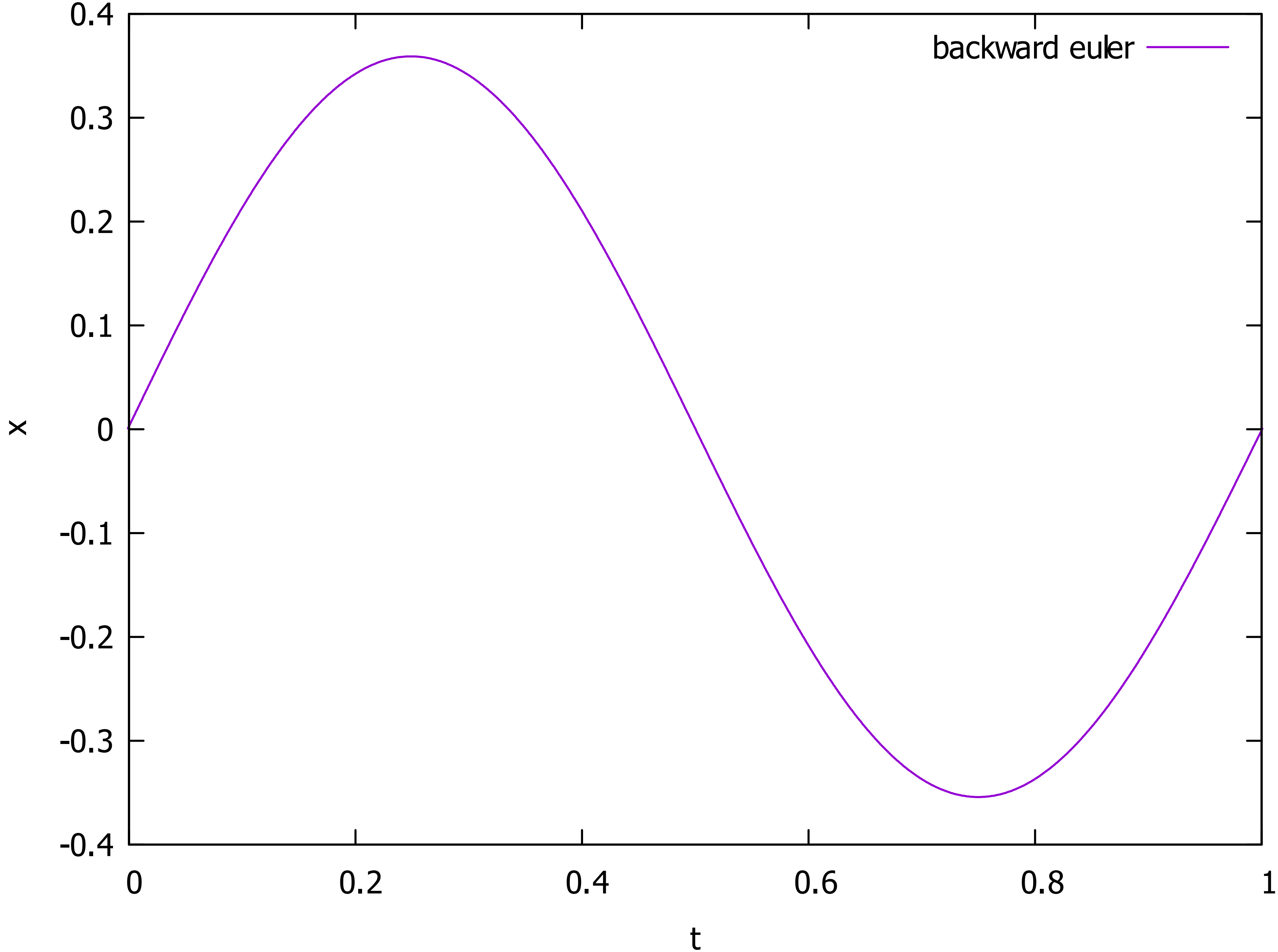
```cpp
//helper function for calculating the difference between calculated and exact solution
double diff(double x, double y){
    return x - y;
}

int main(){
    std::ofstream outfile;  //file handle for writing the outputs in a text file named error.txt
    outfile.open("error.txt", std::ios::out | std::ios::trunc );
    //loop over various step sizes - from 1/9 to 1/1000
    for (size_t i = 9; i < 1000; i++) {
        solution x_values, exact_values;    //variables for storing the calculated and exact solutions
        double dt = 1.0/i;  //step size = 1.0 / step numbers
        initsolver(x_values, 0, rollnum, dt);   //initilize the list of solutions with x_{0} and x_{1}
        exact_values.push_back(exact_solution(0));  //store exact values of x(0) and x(1) in the list of exact solutions
        exact_values.push_back(exact_solution(dt));
        for (size_t j = 0; j < i - 1; j++) {    //step through the total number of steps and store the calculated and exact solutions
            finite_diff_step(System, x_values, dt);
            exact_values.push_back(exact_solution((j + 2) * dt));
        }
        //calculate the differences between the list of exact and calculated solutions
        std::transform(x_values.begin(), x_values.end(), exact_values.begin(), exact_values.begin(), diff);
        //find out the maximum difference and write in the output file
        outfile << dt << "\t" << *std::max_element(exact_values.begin(), exact_values.end()) << "\n";
    }
    outfile.close();
}
```



errors
error
stepsize
0
0.01
0.02
0.03
0.04
0.05
0.06
0.02
0.04
0.06
0.08
0.1
0.12

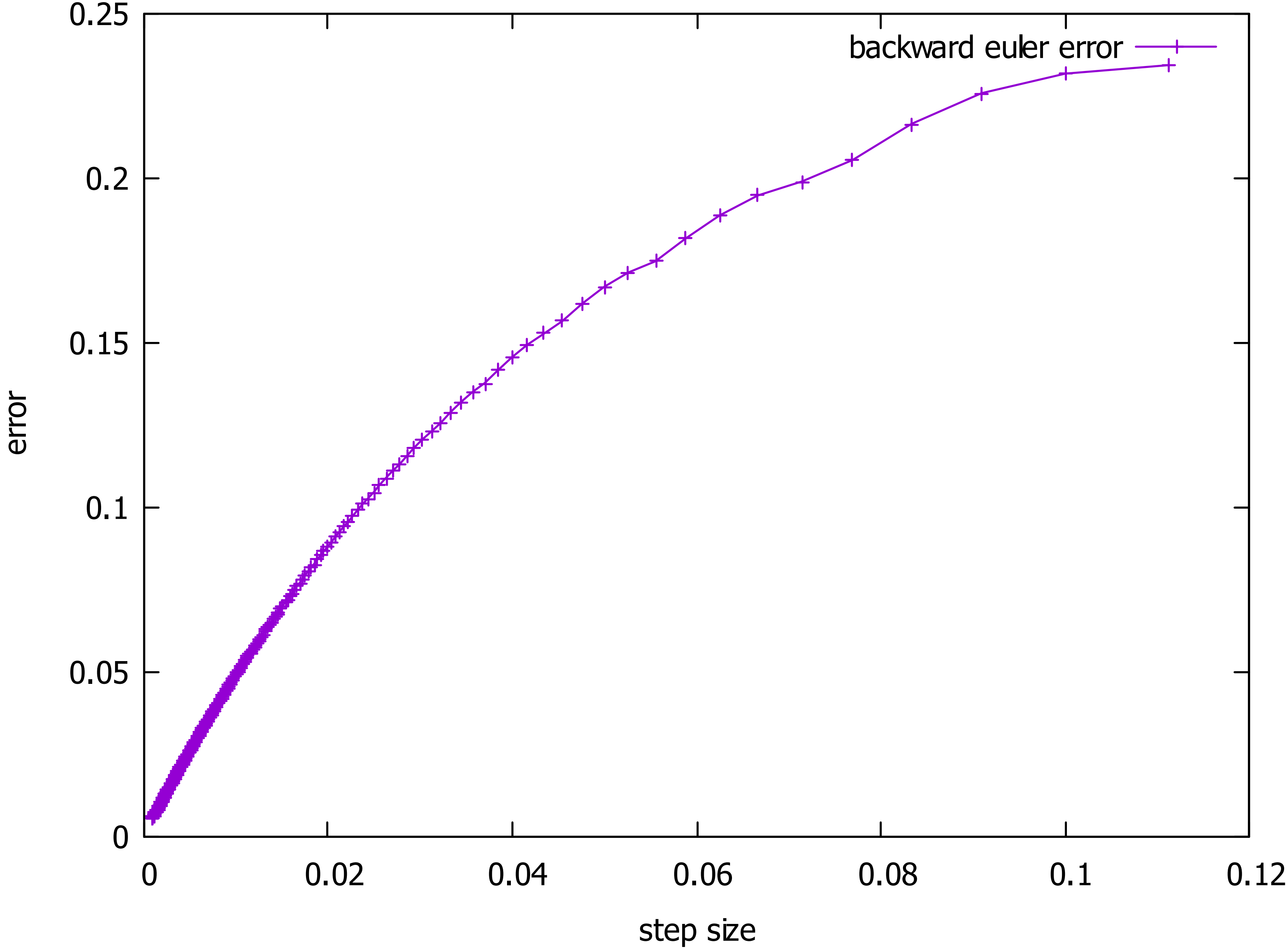
```cpp
#include <iostream>
#include <fstream>
#include <vector>

//constant expressions appearing in the code
constexpr double PI = 3.14159265;   //pi
constexpr double rollnum = 2.26121014;  //my roll number

typedef double state_type;  //data type definition for dependant variable, just a double here - should be std::vector<> for multidimensional case
typedef std::vector<double> solution;   //data type definition for storing the solutions

//This sets the initial values x_0 and x_1 given the values of x(0) and x'(0) and step size
void initsolver(solution& x_values, const state_type& x0, const state_type& dx0dt, const double dt){
    x_values.push_back(x0); //store x(0)
    x_values.push_back(x0 + dx0dt * dt);    //store x_1 = x(0) + x'(0) * dt
}

//The stepper function, calculates x_{n+1} given the differential equation, list of past values and step size
void finite_diff_step(solution& x_values, const double dt){
    //This is the solution of the implicit equation for the backwards euler. This can be solved exactly in this simple case, for non-trivial equations, we might need Newton-raphson to solve for this.
    state_type result = (2 * x_values[x_values.size()-1] - x_values[x_values.size()-2]) / (1 + 4 * PI * PI * dt * dt);  //temporary variable for storing x_{n+1}

    x_values.push_back(result); //store x_{n+1} in the list of solutions
}


int main(){
    solution x_values;  //variable for storing the solutions
    int STEPS = 1000;   //define number of steps
    double dt = 1.0/STEPS;  //define stepsize
    initsolver(x_values, 0, rollnum, dt);   //initilize the list of solutions with x_{0} and x_{1}
    for (size_t i = 0; i < STEPS - 1; i++) { //step through the total number of steps and store the results
        finite_diff_step(x_values, dt);
    }

    std::ofstream outfile;  //save the results in a file named "solution for 1000 steps.txt"
    outfile.open("backward euler 1000 steps.txt", std::ios::out |
```

```
        std::ios::trunc );
    for (size_t i = 0; i < x_values.size(); i++) {
        outfile << 0 + i * dt << "\t" << x_values[i] << "\n";
    }
    outfile.close();
}
```


backward euler
x
t
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
0
0.2
0.4
0.6
0.8
1

```cpp
#include <iostream>
#include <fstream>
#include <functional>
#include <algorithm>
#include <vector>
#include <cmath>

//constant expressions appearing in the code
constexpr double PI = 3.14159265;   //pi
constexpr double rollnum = 2.26121014;  //my roll number

typedef double state_type;  //data type definition for dependant variable, just a double here - should be std::vector<> for multidimensional case
typedef std::vector<double> solution;   //data type definition for storing the solutions

//This sets the initial values x_0 and x_1 given the values of x(0) and x'(0) and step size
void initsolver(solution& x_values, const state_type& x0, const state_type& dx0dt, const double dt){
    x_values.push_back(x0); //store x(0)
    x_values.push_back(x0 + dx0dt * dt);    //store x_1 = x(0) + x'(0) * dt
}

//The stepper function, calculates x_{n+1} given the differential equation, list of past values and step size
void finite_diff_step(solution& x_values, const double dt){
    //This is the solution of the implicit equation for the backwards euler. This can be solved exactly in this simple case, for non-trivial equations, we might need Newton-raphson to solve for this.
    state_type result = (2 * x_values[x_values.size()-1] - x_values[x_values.size()-2]) / (1 + 4 * PI * PI * dt * dt);  //temporary variable for storing x_{n+1}

    x_values.push_back(result); //store x_{n+1} in the list of solutions
}

//function for calculating the exact solution, x(t) = x'(0) / (2 * pi) * sin(2 * pi * t)
double exact_solution(double t){
    return rollnum / (2.0 * PI) * sin(2.0 * PI * t);
}

//helper function for calculating the difference between calculated and exact solution
double diff(double x, double y){
    return abs(y - x);
}

```

```cpp
int main(){
    std::ofstream outfile;  //file handle for writing the outputs in a text file named error.txt
    outfile.open("backward_error.txt", std::ios::out | std::ios::trunc );
    //loop over various step sizes - from 1/9 to 1/1000
    for (size_t i = 9; i < 1000; i++) {
        solution x_values, exact_values;    //variables for storing the calculated and exact solutions
        double dt = 1.0/i;  //step size = 1.0 / step numbers
        initsolver(x_values, 0, rollnum, dt);   //initilize the list of solutions with x_{0} and x_{1}
        exact_values.push_back(exact_solution(0));  //store exact values of x(0) and x(1) in the list of exact solutions
        exact_values.push_back(exact_solution(dt));
        for (size_t j = 0; j < i - 1; j++) {    //step through the total number of steps and store the calculated and exact solutions
            finite_diff_step(x_values, dt);
            exact_values.push_back(exact_solution((j + 2) * dt));
        }
        //calculate the differences between the list of exact and calculated solutions
        std::transform(x_values.begin(), x_values.end(), exact_values.begin(), exact_values.begin(), diff);
        //find out the maximum difference and write in the output file
        outfile << dt << "\t" << *std::max_element(exact_values.begin(), exact_values.end()) << "\n";
    }
    outfile.close();
}
```



backward euler error
0.25
0.2
0.15
0.1
0.05
0
error
0
0.02
0.04
0.06
0.08
0.1
0.12
step size