

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,

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

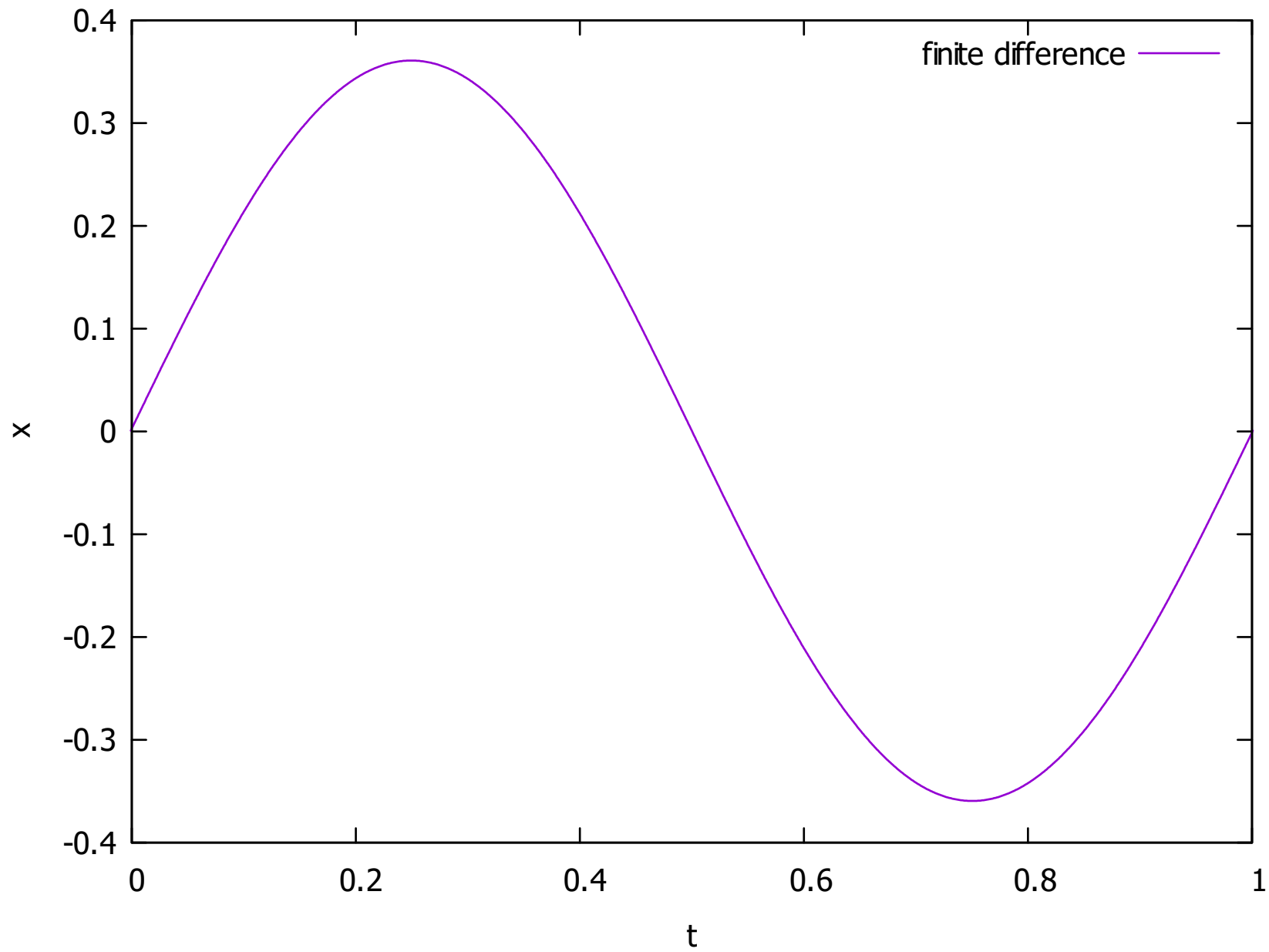
$$\text{i.e., } x_{n+1} = \frac{2x_n - x_{n-1}}{1 + 4\pi^2 \varepsilon^2}.$$

While the exact solution is,

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

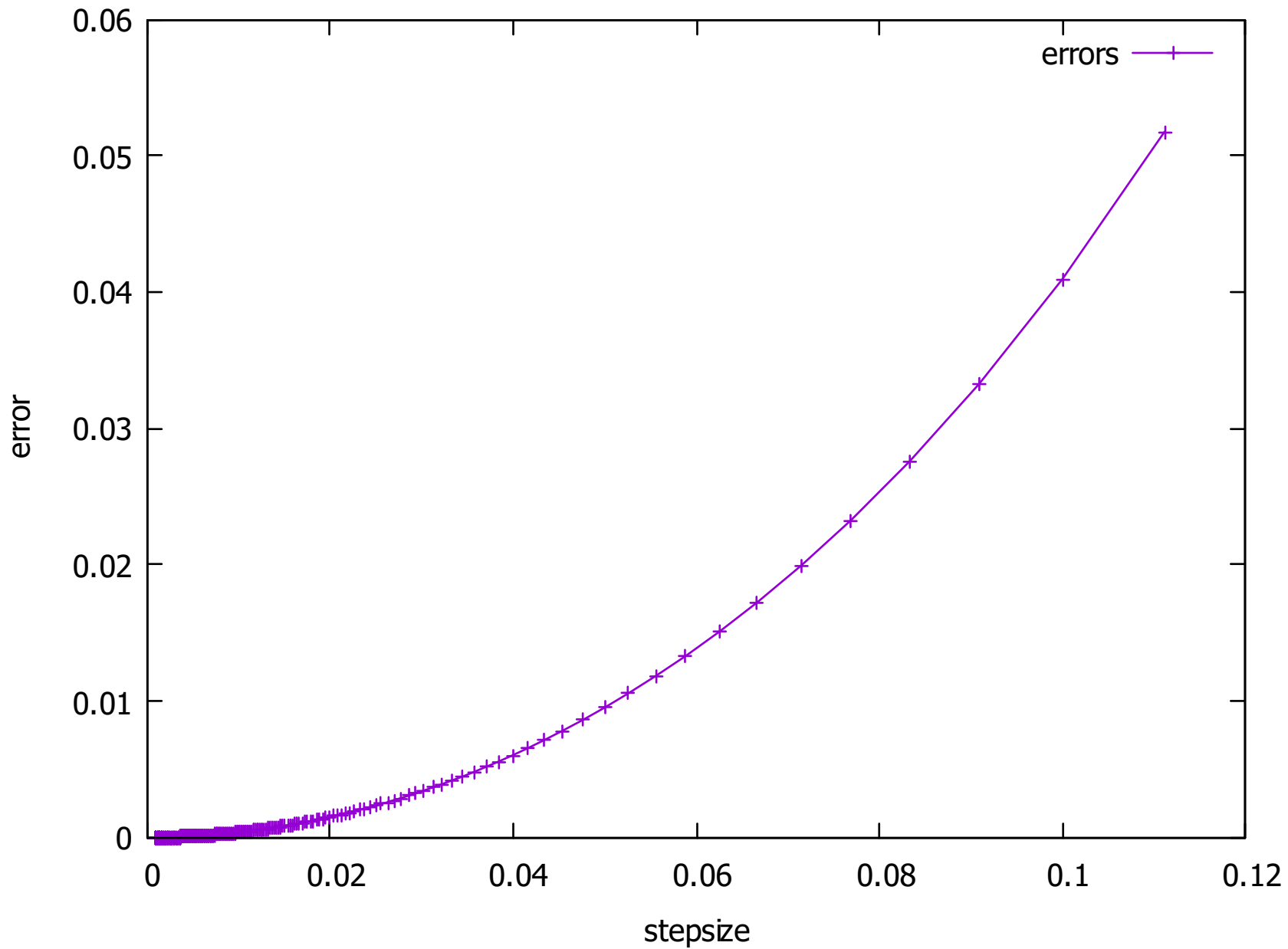
```
1  #include <iostream>
2  #include <fstream>
3  #include <vector>
4
5  //constant expressions appearing in the code
6  constexpr double PI = 3.14159265;    //pi
7  constexpr double rollnum = 2.26121014; //my roll number
8
9  typedef double state_type; //data type definition for dependant variable, ↗
    just a double here - should be std::vector<> for multidimensional case
10 typedef std::vector<double> solution; //data type definition for storing ↗
    the solutions
11
12 //This is the differential Equation
13 void System(state_type& x, state_type& d2xdt2){
14     d2xdt2 = - 4.0 * PI * PI * x;
15 }
16
17 //This sets the initial values x_0 and x_1 given the values of x(0) and ↗
    x'(0) and step size
18 void initsolver(solution& x_values, const state_type& x0, const state_type& ↗
    dx0dt, const double dt){
19     x_values.push_back(x0); //store x(0)
20     x_values.push_back(x0 + dx0dt * dt); //store x_1 = x(0) + x'(0) * dt
21 }
22
23 //The stepper function, calculates x_{n+1} given the differential equation, ↗
    list of past values and step size
24 void finite_diff_step(void (*System)(state_type& x, state_type& d2xdt2), ↗
    solution& x_values, const double dt){
25     state_type result = 0; //temporary variable for storing x_{n+1}
26     System(x_values[x_values.size()-1], result); //calculate x'_{n}
27     result *= dt * dt; //multiply by step_size^2
28     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.
29     x_values.push_back(result); //store x_{n+1} in the list of solutions
30 }
31
32
33 int main(){
34     solution x_values; //variable for storing the solutions
35     int STEPS = 1000; //define number of steps
36     double dt = 1.0/STEPS; //define stepsize
37     initsolver(x_values, 0, rollnum, dt); //initilize the list of ↗
    solutions with x_{0} and x_{1}
38     for (size_t i = 0; i < STEPS - 1; i++) { //step through the total ↗
    number of steps and store the results
```

```
39     finite_diff_step(System, x_values, dt);
40 }
41
42     std::ofstream outfile; //save the results in a file named "solution
    for 1000 steps.txt"
43     outfile.open("solution for 1000 steps.txt", std::ios::out |
        std::ios::trunc );
44     for (size_t i = 0; i < x_values.size(); i++) {
45         outfile << 0 + i * dt << "\t" << x_values[i] << "\n";
46     }
47     outfile.close();
48 }
```



```
1  #include <iostream>
2  #include <fstream>
3  #include <functional>
4  #include <algorithm>
5  #include <vector>
6  #include <cmath>
7
8  //constant expressions appearing in the code
9  constexpr double PI = 3.14159265;    //pi
10 constexpr double rollnum = 2.26121014; //my roll number
11
12 typedef double state_type; //data type definition for dependant variable, ↗
    just a double here - should be std::vector<> for multidimensional case
13 typedef std::vector<double> solution; //data type definition for storing ↗
    the solutions
14
15 //This is the differential Equation
16 void System(state_type& x, state_type& d2xdt2){
17     d2xdt2 = - 4.0 * PI * PI * x;
18 }
19
20 //This sets the initial values x_0 and x_1 given the values of x(0) and ↗
    x'(0) and step size
21 void initsolver(solution& x_values, const state_type& x0, const state_type& ↗
    dx0dt, const double dt){
22     x_values.push_back(x0); //store x(0)
23     x_values.push_back(x0 + dx0dt * dt); //store x_1 = x(0) + x'(0) * dt
24 }
25
26 //The stepper function, calculates x_{n+1} given the differential equation, ↗
    list of past values and step size
27 void finite_diff_step(void (*System)(state_type& x, state_type& d2xdt2), ↗
    solution& x_values, const double dt){
28     state_type result = 0; //temporary variable for storing x_{n+1}
29     System(x_values[x_values.size()-1], result); //calculate x''_{n}
30     result *= dt * dt; //multiply by step_size^2
31     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.
32     x_values.push_back(result); //store x_{n+1} in the list of solutions
33 }
34
35 //function for calculating the exact solution,  $x(t) = x'(0) / (2 * \pi) * \sin(2 * \pi * t)$  ↗
36 double exact_solution(double t){
37     return rollnum / (2.0 * PI) * sin(2.0 * PI * t);
38 }
39
```

```
40 //helper function for calculating the difference between calculated and exact solution
41 double diff(double x, double y){
42     return x - y;
43 }
44
45 int main(){
46     std::ofstream outfile; //file handle for writing the outputs in a text
    file named error.txt
47     outfile.open("error.txt", std::ios::out | std::ios::trunc );
48     //loop over various step sizes - from 1/9 to 1/1000
49     for (size_t i = 9; i < 1000; i++) {
50         solution x_values, exact_values; //variables for storing the
        calculated and exact solutions
51         double dt = 1.0/i; //step size = 1.0 / step numbers
52         initsolver(x_values, 0, rollnum, dt); //initilize the list of
        solutions with x_{0} and x_{1}
53         exact_values.push_back(exact_solution(0)); //store exact values of
        x(0) and x(1) in the list of exact solutions
54         exact_values.push_back(exact_solution(dt));
55         for (size_t j = 0; j < i - 1; j++) { //step through the total
            number of steps and store the calculated and exact solutions
56             finite_diff_step(System, x_values, dt);
57             exact_values.push_back(exact_solution((j + 2) * dt));
58         }
59         //calculate the differences between the list of exact and
        calculated solutions
60         std::transform(x_values.begin(), x_values.end(), exact_values.begin
            (), exact_values.begin(), diff);
61         //find out the maximum difference and write in the output file
62         outfile << dt << "\t" << *std::max_element(exact_values.begin(),
            exact_values.end()) << "\n";
63     }
64     outfile.close();
65 }
```



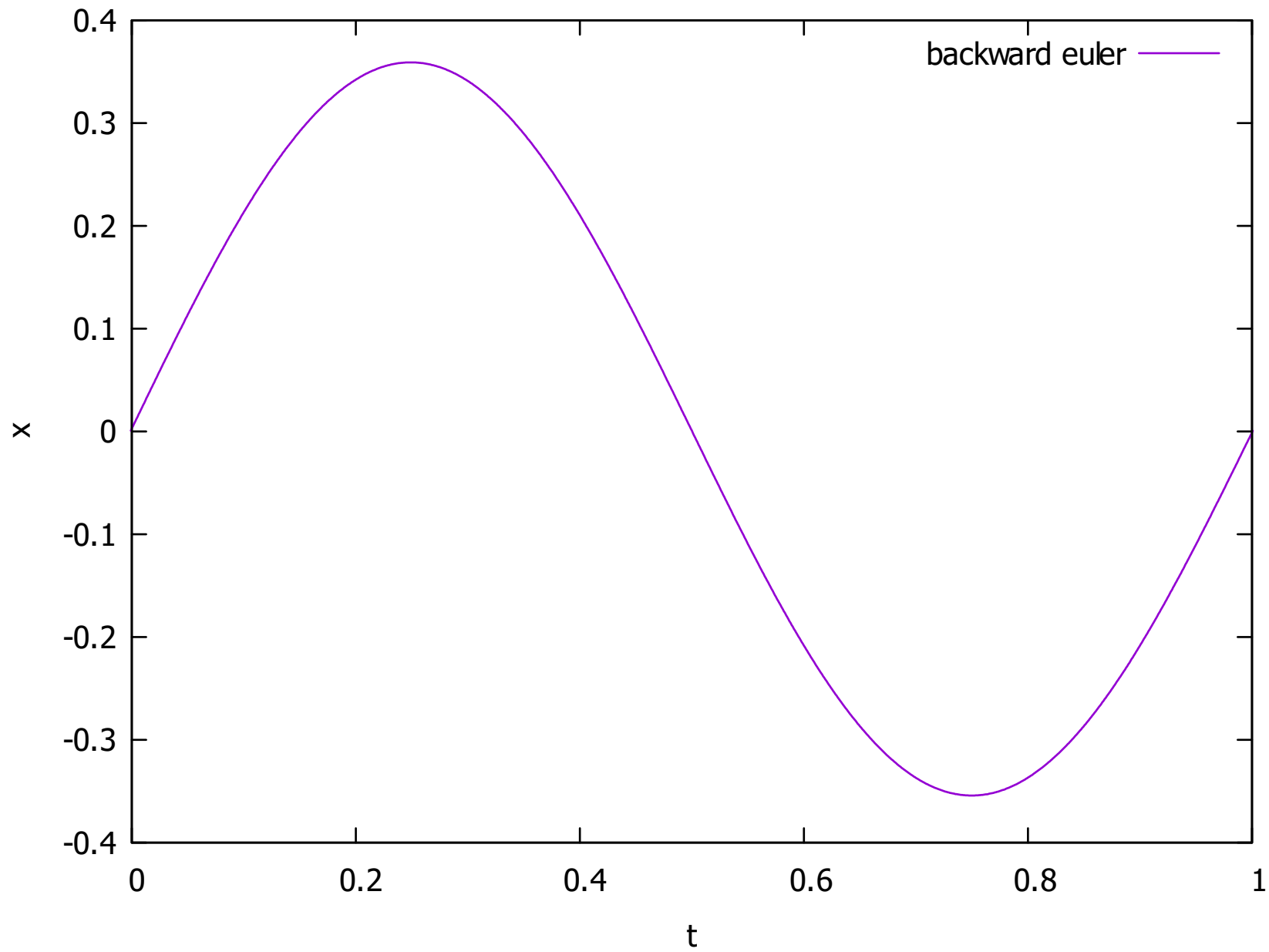
```

1  #include <iostream>
2  #include <fstream>
3  #include <vector>
4
5  //constant expressions appearing in the code
6  constexpr double PI = 3.14159265;    //pi
7  constexpr double rollnum = 2.26121014; //my roll number
8
9  typedef double state_type; //data type definition for dependant variable, ↗
    just a double here - should be std::vector<> for multidimensional case
10 typedef std::vector<double> solution; //data type definition for storing ↗
    the solutions
11
12 //This sets the initial values x_0 and x_1 given the values of x(0) and ↗
    x'(0) and step size
13 void initsolver(solution& x_values, const state_type& x0, const state_type& ↗
    dx0dt, const double dt){
14     x_values.push_back(x0); //store x(0)
15     x_values.push_back(x0 + dx0dt * dt); //store x_1 = x(0) + x'(0) * dt
16 }
17
18 //The stepper function, calculates x_{n+1} given the differential equation, ↗
    list of past values and step size
19 void finite_diff_step(solution& x_values, const double dt){
20     //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.
21     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}
22
23     x_values.push_back(result); //store x_{n+1} in the list of solutions
24 }
25
26
27 int main(){
28     solution x_values; //variable for storing the solutions
29     int STEPS = 1000; //define number of steps
30     double dt = 1.0/STEPS; //define stepsize
31     initsolver(x_values, 0, rollnum, dt); //initilize the list of ↗
    solutions with x_{0} and x_{1}
32     for (size_t i = 0; i < STEPS - 1; i++) { //step through the total ↗
    number of steps and store the results
33         finite_diff_step(x_values, dt);
34     }
35
36     std::ofstream outfile; //save the results in a file named "solution ↗
    for 1000 steps.txt"
37     outfile.open("backward euler 1000 steps.txt", std::ios::out | ↗

```



```
        std::ios::trunc );  
38     for (size_t i = 0; i < x_values.size(); i++) {  
39         outfile << 0 + i * dt << "\t" << x_values[i] << "\n";  
40     }  
41     outfile.close();  
42 }
```



```
1  #include <iostream>
2  #include <fstream>
3  #include <functional>
4  #include <algorithm>
5  #include <vector>
6  #include <cmath>
7
8  //constant expressions appearing in the code
9  constexpr double PI = 3.14159265;    //pi
10 constexpr double rollnum = 2.26121014; //my roll number
11
12 typedef double state_type; //data type definition for dependant variable, ↗
    just a double here - should be std::vector<> for multidimensional case
13 typedef std::vector<double> solution; //data type definition for storing ↗
    the solutions
14
15 //This sets the initial values x_0 and x_1 given the values of x(0) and ↗
    x'(0) and step size
16 void initsolver(solution& x_values, const state_type& x0, const state_type& ↗
    dx0dt, const double dt){
17     x_values.push_back(x0); //store x(0)
18     x_values.push_back(x0 + dx0dt * dt); //store x_1 = x(0) + x'(0) * dt
19 }
20
21 //The stepper function, calculates x_{n+1} given the differential equation, ↗
    list of past values and step size
22 void finite_diff_step(solution& x_values, const double dt){
23     //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.
24     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}
25
26     x_values.push_back(result); //store x_{n+1} in the list of solutions
27 }
28
29 //function for calculating the exact solution,  $x(t) = x'(0) / (2 * \pi) * \sin(2 * \pi * t)$  ↗
30 double exact_solution(double t){
31     return rollnum / (2.0 * PI) * sin(2.0 * PI * t);
32 }
33
34 //helper function for calculating the difference between calculated and ↗
    exact solution
35 double diff(double x, double y){
36     return abs(y - x);
37 }
38
```

```
39 int main(){
40     std::ofstream outfile; //file handle for writing the outputs in a text
        file named error.txt
41     outfile.open("backward_error.txt", std::ios::out | std::ios::trunc );
42     //loop over various step sizes - from 1/9 to 1/1000
43     for (size_t i = 9; i < 1000; i++) {
44         solution x_values, exact_values; //variables for storing the
            calculated and exact solutions
45         double dt = 1.0/i; //step size = 1.0 / step numbers
46         initsolver(x_values, 0, rollnum, dt); //initilize the list of
            solutions with x_{0} and x_{1}
47         exact_values.push_back(exact_solution(0)); //store exact values of
            x(0) and x(1) in the list of exact solutions
48         exact_values.push_back(exact_solution(dt));
49         for (size_t j = 0; j < i - 1; j++) { //step through the total
            number of steps and store the calculated and exact solutions
50             finite_diff_step(x_values, dt);
51             exact_values.push_back(exact_solution((j + 2) * dt));
52         }
53         //calculate the differences between the list of exact and
            calculated solutions
54         std::transform(x_values.begin(), x_values.end(), exact_values.begin(),
            exact_values.begin(), diff);
55         //find out the maximum difference and write in the output file
56         outfile << dt << "\t" << *std::max_element(exact_values.begin(),
            exact_values.end()) << "\n";
57     }
58     outfile.close();
59 }
```

