Project 1B

Solution of initial value problem:

$y'(x) = f(x)y(x) + g(x)$, where $f(x) = x^2 \sin(\cos(x)), g(x) = \sinh(\cosh(x))$ in the interval $[a,b]$, $y(a) = y_0$ using Euler's method.

(In my assignment were functions $f(x) = \sinh(\cosh(x))$ and $g(x) = x^2 \sin(\cos(x))$, but with such conditions growth of $y(x)$ was from 24 to number with about 200 zeroes in the interval from –4 to –3.9. Thus I decided to replace functions)

Abstract

This work is an introduction to the numerical Euler's method for finding solutions for the first order ordinary equations. Description of most commonly used algorithm is given with fully working code implemented on C++ language. Also discussed methods for decreasing error ratio for the method and its calculation.

Introduction

It is the exception rather than the rule when a differential equation of the general form

$$\frac{dy}{dx} = f(x,y) \quad (1)$$

can be solved exactly and explicitly by elementary analytical methods. To discuss the development and use of numerical procedures we will concentrate on the first order initial value problem consisting of the differential equation and the initial condition

$$y(x_0) = y_0 \qquad (2)$$

We assume that the functions f and f_y are continuous on some rectangle containing the point $(x_{0,} y_0)$. Then there is exists a unique solution $y = \varphi(x)$ of the given problem in some interval about x_0.

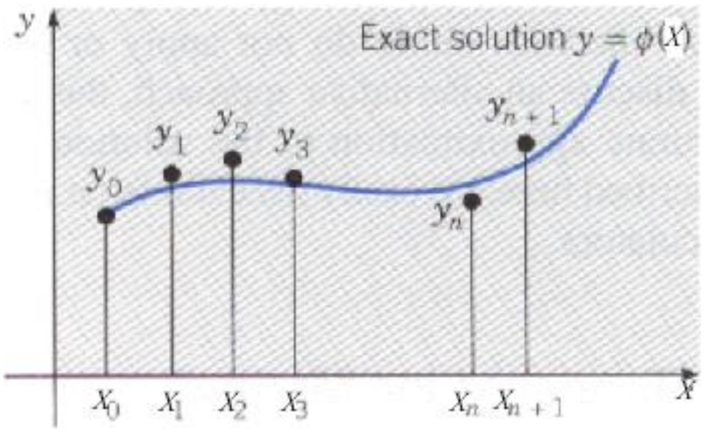

Figure 1. A numerical approximation to the solution.

To describe Euler's method, we first choose a fixed step size $h > 0$ and consider the points

$$x_0, x_1, x_2, ..., x_n, ...$$

where $x_n = x_0 + nh$, so that

$$x_{n+1} = x_n + h \quad (3)$$

for $n = 0,1,2...$ Our goal is to find suitable approximations

$$y_1, y_2, y_3, ..., y_n, ...$$

to the true values

$$\varphi(x_1),\varphi(x_2),\varphi(x_3),...,\varphi(x_n)...$$

of the solution $\varphi(x)$ of Eq. (1) at the points $x_1, x_2, x_3,...$ .Thus we seek reasonably accurate approximations

$$y_n \approx \varphi(x_n)$$

for $n = 0,1,2...$.When $x = x_0$ the rate to change of y with respect to x is $y' = f(x_0, y_0)$. If y continued to change at this same rate from $x = x_0$ to $x = x_0 + h$, then the change in y would be exactly $h \cdot f(x_0, y_0)$. Therefore

$$y_1 = y_0 + h \cdot f(x_0, y_0)$$

as out approximation to the true value $y(x_2)$. In general, having reached the nth approximate value $y_n \approx \varphi(x_n)$, we take

$$y_{n+1} = y_n + h \cdot f(x_n, y_n) \qquad (4)$$

as our approximation to the true value $\varphi(x_{n+1})$.

Equation (4) tells us how to make the typical step from y_n to y_{n+1} and is the heart of Euler's method.

Euler's method with step size h consists in applying the iterative formula

$$y_{n+1} = y_n + h \cdot f(x_n, y_n)\ ,\ n \geq 0$$

with x_0 to compute successive approximations $y_1, y_2, y_3,..., y_n,...$ to the true values $\varphi(x_1),\varphi(x_2),\varphi(x_3),...,\varphi(x_n)...$ of the exact solution $y = \varphi(x)$ at the points $x_0, x_1, x_2,..., x_n,...$ respectively.

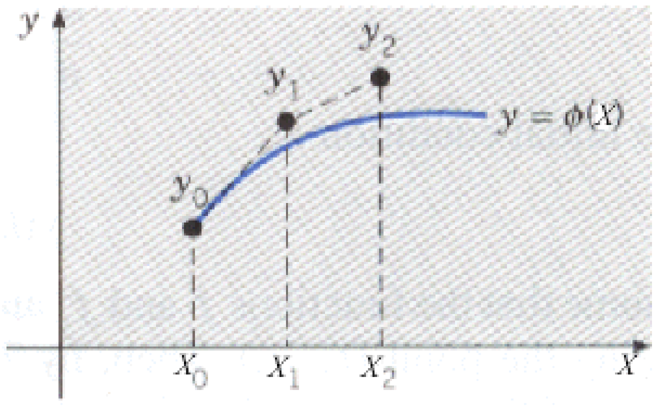

Figure 2. The Euler or tangent line approximation.

To obtain more accurate results there are essentially two alternatives: one is to use a smaller step size with correspondingly more steps being required to cover a given interval; the other is to devise a more accurate and efficient approximate formula to replace Eq. (4). Usually, the second alternative is the better one.

Suppose we assume that the solution $y = \varphi(x)$ has a Taylor series about the point x_n. Then

$$\varphi(x_n + h) = \varphi(x_n) + \varphi'(x_n) \cdot h + \varphi''(x_n)\frac{h^2}{2!} + ...$$

or

$$\varphi(x_{n+1}) = \varphi(x_n) + f(x_n, \varphi(x_n)) \cdot h + \varphi''(x_n)\frac{h^2}{2!} + ...$$

If values $\varphi(x_{n+1})$ and $\varphi(x_n)$ are replaced by their approximate values y_{n+1} and y_n we again obtain Euler formula:

$$y_{n+1} = y_n + h \cdot T^{(k)}(x_n, y_n), \quad (5)$$

where

$$T^{(k)}(x_n, y_n) = f(x_n, y_n) + \frac{h}{2!} f'(x_n, y_n) + \frac{h^2}{3!} f''(x_n, y_n) + \ldots + \frac{h^k}{(k+1)!} f^{(k)}(x_n, y_n)$$

Further, by using a Taylor series with a remainder it is possible to estimate the magnitude of the error in the formula.

Local and Cumulative Errors

There are several sources of error in Euler`s method that may make the approximation y_n to $\varphi(x_n)$ unreliable for large values of n, those for which x_n is not sufficiently close to x_0. The error in the linear approximation formula (4) is the amount by which the tangent line at (x_n, y_n) departs from the solution curve through (x_n, y_n). This error, introduces at each step in the process, is called **the *local error***.

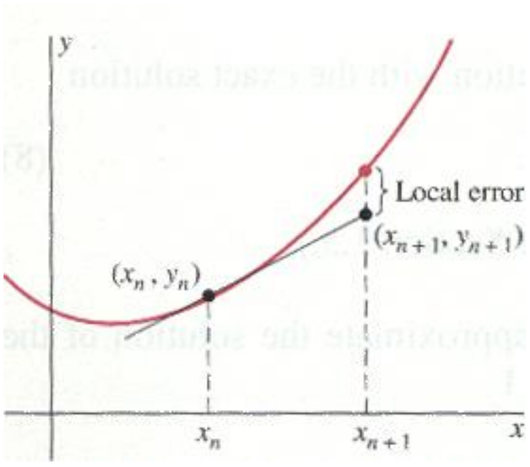

Figure 3. The local error in Euler's method.

The local error would be the total error in y_{n+1} if the starting point y_n were an exact value, rather than merely approximation to the actual value $\varphi(x_n)$. But y_n itself suffers from the accumulated effects of all the locals errors introduced at the previous steps. Thus, we have ***cumulative error*** in Euler's method.

Tree-Term Taylor Formula

An alternative way of improving the Euler formula rests on the Taylor expansion of the solution of the initial value problem (1),(2). By retaining the first two terms in the expansion of φ about $x = x_n$ we obtain the Euler formula, so by keeping more terms we should obtain a more accurate formula. Assuming that f has continuous second partial derivatives, so that φ has at least three continuous derivatives in the interested interval, we can write

$$\varphi(x_n + h) = \varphi(x_n) + \varphi'(x_n) \cdot h + \varphi''(x_n) \cdot \frac{h^2}{2!} + \varphi''(\bar{x}_n) \cdot \frac{h^3}{3!} \qquad (6)$$

where $\bar{x}_n$ is some point in the interval $x_n < \bar{x}_n < x_n + h$.

From Eq. (1) we have

$$\varphi'(x_n) = f(x_n, \varphi(x_n)) \qquad (7)$$

Further, by differentiating Eq. (7) and setting $x = x_n$ we find that

$$\varphi''(x_n) = f_x(x_n, \varphi(x_n)) + f_y(x_n, \varphi(x_n)) \cdot \varphi'(x_n) \qquad (8)$$

The three-term Taylor formula is obtained by replacing $\varphi(x_n)$ by its approximate value y_n in the formulas, and then neglecting the term $\varphi'''(\bar{t}_n) \cdot \frac{h^3}{3!}$ in Eq. (6). Thus we obtain

$$y_{n+1} = y_n + h \cdot f_n + \frac{h^2}{2!} \cdot (f_x(x_n, y_n) + f_y(x_n, y_n) \cdot f_n) \qquad (9)$$

Assuming temporarily that $\varphi(x_n) = y_n$, it follows that the local truncation error ε_{n+1} associated with formula (9) is

$$\varepsilon_{n+1} = \varphi(x_{n+1}) - y_{n+1} = \frac{\varphi'''(\bar{t}_n) \cdot h^3}{6} \qquad (10)$$

where $x_n < \bar{x}_n < x_n + h$. Thus the local truncation error for the three-term Taylor formula is proportional to h^3. It can be shown that for a finite interval the global truncation error is no greater than a constant times h^2.

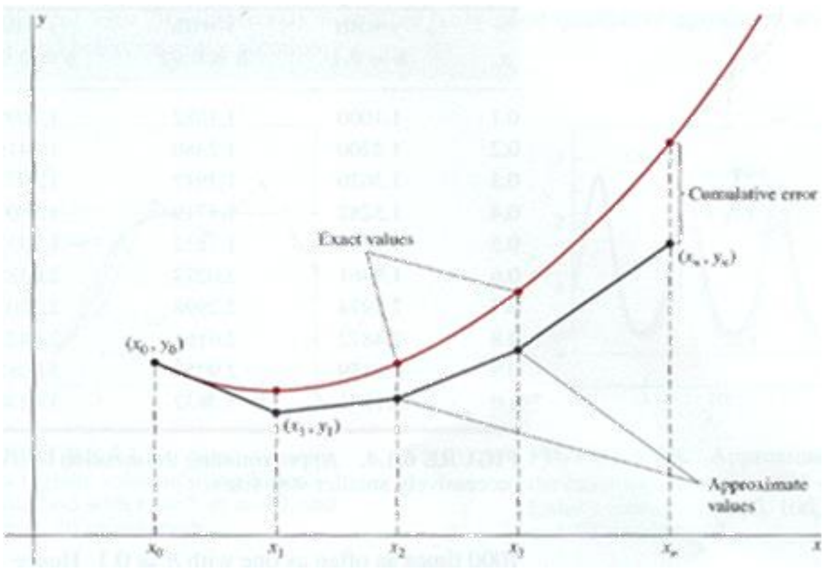

Figure 4. The cumulative error in Euler's method.

User requirements and Algorithm

We need to find numerical solution to the next problem:

$\int_a^b (f(x)y(x) + g(x))dx$, where $y'(x) = f(x)y(x) + g(x)$, interval $[a,b]$ is known and $y(a) = y_0$.

We can rewrite integral to the next form:

$$\int_a^b (f(x)y(x) + g(x))dx = \int_a^b y'(x)dx = y(x)\Big|_a^b = y(b) - y(a)$$

Thus, our problem is reduced to the alternative problem for finding value of $y(x_n)$, where $x_n = b$.

It is easily to write computer program, that uses Euler's method, to carry out calculations required to produce result for this problem.

The Euler's Method

Step 1. Define $f(x, y)$

Step 2. Define initial values x_0 and y_0

```
Step 3. Define step size $h$ and number of steps $n$
Step 4. For $i$ from 1 to $n$ do
        $y = y + h \cdot f(x, y)$
        $x = x + h$
Step 5. Output of $y$ and value of integral $y - y_0$.
```

In using a numerical procedure we must always keep in mind the question of whether the results are accurate enough to be useful. In the preceding algorithm, the accuracy of the numerical results could be ascertained directly by a comparison with the solution obtained from the smaller step size. We can obtain the following procedure for our problem:

1. Apply the algorithm to the initial values and reasonable value of h.
2. repeat with $\frac{h}{2}, \frac{h}{4}$, and so forth, at each stage having the step size for the next application of Euler's method.
3. Continue until the results obtained at one stage agree – to an appropriate number of signigicant digits – with those obtained at the previous stage. Then the approximate values obtained at this stage are considered likely to be accurate to the indicated number of significant digits.

User information for Code

Full listing of code implemented on C++ can be found in Appendix.

For our initial value problem $f(x) = x^2 \cdot \sin(\cos(x)), g(x) = \sinh(\cosh(x))$, interval $[-4,3]$ and $y(-4) = y_0 = 24$. All functions are continuous in the interval and have at least two derivatives, thus, we can use Tree-Term Taylor Formula for solution.

$$y'(x) = f(x_n, y_n) = x^2 \cdot \sin(\cos(x)) \cdot y(x) + \sinh(\cosh(x))$$

$$y''(x) = f'(x_n, y_n) = f'(x)y(x) + y'(x)f(x) + g'(x) = (\sin(\cos(x)) \cdot \sin^2(x) +$$

$$+ \cos(\cos(x)) \cdot \cos(x)) \cdot y(x) + y'(x) \cdot x^2 \sin(\cos(x)) + \cosh(\cosh(x)) \cdot \sinh(x)$$

$$y'''(x) = f''(x_n, y_n) = f''(x) \cdot y(x) + y''(x) \cdot f(x) + 2 \cdot f'(x) \cdot y'(x) + g''(x) = (\sin(\cos(x)) \cdot (2 - x^2 \cdot \sin^2(x)) -$$

$$- x \cdot \cos(\cos(x)) \cdot (4 + x \cdot \cos(x))) \cdot y(x) + y''(x) \cdot x^2 \cdot \sin(\cos(x)) + 2 \cdot (2 \cdot x \cdot \sin(\cos(x)) -$$

$$- x^2 \cdot \cos(\cos(x)) \cdot \sin(x)) \cdot y'(x) + \sinh(\cosh(x)) \cdot \sinh^2(x) + \cosh(\cosh(x)) \cdot \cosh(x)$$

In our case

$$T^2(x_n, y_n) = f(x_n, y_n) + \frac{h}{2!} \cdot f'(x_n, y_n) + \frac{h^2}{3!} \cdot f''(x_n, y_n)$$

I n program for all calculations and variables used ***long double*** data type. According to the Microsoft Specific of ***long double*** data type, its range $\pm 1.2 \cdot 10^{4932}$ with at least 19 digits of precision.

Testing

Here is a table of results given by the program for simple functions $f(x)$ and $g(x)$ with known solutions that can be easily calculated by analytical methods with good precision.

Function $f(x)$	Function $g(x)$	Interval $[a,b]$	Value of y_0	Calculated value $y(b)$	Exact value $y(b)$

1.0	0.0	[0,1]	1.0	2.718245	2.718288
1.0	x	[0,1]	1.0	3.436523	3.436562
4.0	1- x	[0,1]	1.0	64.897798	64.897803

Exact solution for first initial value problem is $\varphi(x) = e^x$, for third is $\varphi(x) = \frac{1}{4}x - \frac{3}{16} + \frac{19}{16}e^{4t}$

We can see that program gives results that are clear to the real values.

Evaluation Integral for given initial value problem

Output of the program for given input values that have been described above is in the table

Precision ε	Number of steps n	Step size h	Value of $y(b)$	Integral value
0.0001	32768	0.0002136230	998.8108969488	974.8108969488
0.00001	65536	0.0001068115	998.8108952733	97481089527.33
0.000001	131072	$5.340576 \cdot 10^{-5}$	998.8108950643	9748108950643.
0.0000001	262144	$2.670288 \cdot 10^{-5}$	998.8108950382	9748108950382.

As we can the value of integral converges to 974.81. The result calculated by program MathCad 992.058. Graph of the function $y(x)$ has form:

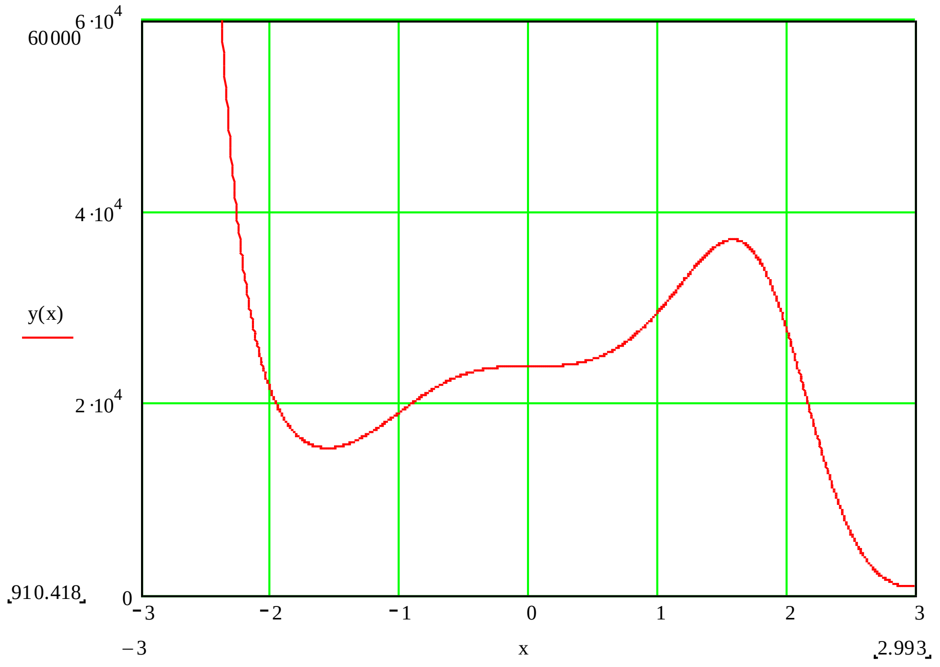

Figure 5. Graph of the function y(x)

Appendix

Code listing of program implemented on C++ for solving problem described above.

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

```
#include <stdlib.h>

typedef long double l_double; /* defining new type name */
/* global definitions */
const l_double a_value=-4.0; /* $a$ value */
const l_double b_value=3.0; /* $b$ value */
const l_double __y0=24.0; /* $y_0$ value, there are already defined
y0 and  _y0 variables in module math.h */
unsigned long n; /* $n$ value */
/*calculates $f(x)$ */
l_double f(const l_double value)
{
	return powl(value,2)*sinl(cosl(value));
}
/* calculates $f'(x)$ */
l_double f1(const l_double value)
{
	return value*(2*sinl(cosl(value))-value*sinl(value)*cosl(cosl(value)));
}
/* calculates $f''(x)$ */
l_double f11(const l_double value)
{
	return sinl(cosl(value))*(2-powl(value*sinl(value),2))-
		value*cosl(cosl(value))*(4*sinl(value)+value*cosl(value));
}
/* calculates $g(x)$ */
l_double g(const l_double value)
{
	return sinhl(coshl(value));
}
/* calculates $g'(x)$ */
l_double g1(const l_double value)
{
	return sinhl(value)*coshl(coshl(value));
}
/* calculates $g''(x)$ */
l_double g11(const l_double value)
{
	return sinhl(coshl(value))*powl(sinhl(value),2)+coshl(coshl(value))*coshl(value);
}
/* calculates $f'(x_n,y_n)$ */
l_double y1(const l_double x_value,const l_double y_value)
{
	return f(x_value)*y_value+g(x_value);
}
/* calculates $f''(x_n,y_n)$ */
l_double y11(const l_double x_value,const l_double y_value)
{
	return f1(x_value)*y_value+y1(x_value,y_value)*f(x_value)+g1(x_value);
}
```

```
/* calculates $f'''(x_n,y_n)$ */
l_double y111(const l_double x_value,const l_double y_value)
{
        return f11(x_value)*y_value+y11(x_value,y_value)*f(x_value)+
                2*f1(x_value)*y1(x_value,y_value)+g11(x_value);
}
/* calculates $T^2(x_n,y_n)$ */
l_double t2(const l_double x_value,const l_double y_value,const l_double h_value)
{
        return y1(x_value,y_value)+(h_value/2)*y11(x_value,y_value)+
                (powl(h_value,2)/6)*y111(x_value,y_value);
}
/* main function */
void main()
{
        bool flag=false;/* first calculation of $y(b)$ flag*/
        bool end_flag=false; /*finishing calculations flags*/
        l_double y;/* value of $y_{n+1}$ */
        l_double h;/* step size $h$ */
        l_double error; /*precision $\varepsilon$ */

        cout.precision(20);/*setting up output precision*/
        n=1;/*initial value of $n$ */
        cout<<”Enter error value : ”;
        cin>>error;
        while (!end_flag)
        {
                /* initialization of variables */
                n*=2;
                l_double x=a_value;
                y=__y0;
                h=(b_value-a_value)/n;
                l_double previous; /*previous result*/
                for (int i=0; i<n; i++)
                {
                        y=y+h*t2(x,y,h);/*calculation of $y_{n+1}$ */
                        x+=h;/*increasing of $x$ */
                }
                if (!flag)
                {
                        /*first iteration*/
                        previous=y;
                        flag=true;
                }
                else
                        if (error>=fabsl(previous-y))/*we have got our precision*/
                                end_flag=true;
                        else
                        {
                        cout<<”y(b)=”<<”,n=”<<n<<”,h=”<<h<<”,error=”fabsl(previous-y)<<endl;
                        previous=y;
```

```
                    }

            }
            cout<<"Final result="<<y<<",n="<<n<<",h="<<h<<endl;
            cout<<"Integral value: "<<y-__y0<<endl;
}
```

References

1. C. Henry Edwards, David E. Penney, ”Calculus with analytic geometry”, Prentice Hall, 1998
2. C. Henry Edwards, David E. Penney, ”Elementary differential equations with boundary problems”, 2000
3. William E. Boyce, Richard C. DiPrima, “Elementary Differential equations and boundary value problems”, John Wiley&Sons, Inc1996

Help program: MathCad