

# Development and Numerical Simulation of Algorithms to the Computational Resolution of Ordinary Differential Equations

Leniel Braz de Oliveira Macaferi<sup>1</sup>, Dener Martins dos Santos<sup>2</sup>

Computer Engineering – Centro Universitário de Barra Mansa (UBM)  
714, 35 St., – CEP: 27261-140 – Fazenda Santa Cecília – Barra Mansa - RJ – Brazil

leniel@gmail.com, dener.martins@ubm.br

***Abstract.** The computational simulation nowadays consists of a great tool assisting the learning process of complex mathematical calculations. This informative article shows how the computational resolution of differential equations supports this learning. Two different types of computational resolution for differential equations are demonstrated: explicit Euler's method and Runge-Kutta's fourth order method. The programs were developed in C programming language. The results obtained via both methods are compared with the respective analytical solution (traditional); in these a low level of generated computational error was perceived during their simulation, not compromising the methods.*

***Keywords:** ordinary differential equations, numerical methods, C programming language.*

---

<sup>1</sup> UBM graduation student, Computer Engineering, 4<sup>th</sup> term, 2<sup>o</sup> semester, 2004.

<sup>2</sup> Ph.D. in Engineering - USP (São Paulo), UBM professor, Computer Engineering.

## 1 INTRODUCTION

Ordinary differential equations appear with great frequency in models that quantitatively describe natural phenomena susceptible to mathematical treatment from the most diverse knowledge areas (Physics, Engineering, Biology, Fluid Mechanics, etc.) [1]. The complexity of an ordinary differential equation solution resides in the fact that it's an expression that involves simultaneously a function originally unknown and its respective derivatives [2] [3].

Normally, in the description of any process (phenomena) more than one independent variable can be associated to it. Thus the differential equation is denominated partial. However, when it's possible to make simplifying considerations, the partial equation is reduced to an ordinary differential equation [4]. This way, as this author describes, an ordinary differential equation is an expression that establishes the relation between a function with its respective derivatives according to the enforced initial conditions.

## 2 OBJECTIVE

This informative article has as objective the development of programs in the C programming language to facilitate the study and learning of ordinary differential equations. This article also intends to serve as the base to numerical simulations of industrial routines and physical phenomena enabling the comprehension of these extreme process situations.

## 3 REVISION

The C programming language was created, influenced and tested in field

by professional programmers [5]. C is the most spread language all over the world because it enables the programmer to do what he wants: few restrictions, few errors, structures blocks, isolated functions and a compact set of keywords. Given the fact of its portability, it's possible to reutilize the developed source code, that is, the same code can be compiled and executed in other operating systems with little or no modification. This saves time and money. The generated code is extremely compact and fast. C is used in all types of programming works. Actually is quite employed in the area of industrial automation despite the complexity of its structure.

One of the greatest problems of automation in general is to mathematically describe determined routines and posteriorly codify them in a computer language that outputs the most reliable results, inside an error criterion pre-established. It's important that this generated code be representative of operational routines of the industry. Commonly, when an operational routine is simulated, it is described in the form of ordinary differential equations, due to the fact that they propitiate the description of the inherent phenomena of a given process, to which is related one or more variables simultaneously.

The verification of the efficacy of the developed programs was done comparing the obtained results through the numerical simulation with the obtained results of the analytical solution (traditional, manual solution – applying the different rules to the solution of differential equations). Doing so, it was possible to validate the computational model developed as well as to analyze the error produced by the methods being simulated.

Two numerical methods used in

the resolution of ordinary differential equations were used: explicit Euler's method and Runge-Kutta's fourth order method. Both methods are based in the Taylor's series formula and are of simple step [1] [6] [7]. The difference between them resides in the number of steps taken when the point  $y_{n+1}$  is being calculated. While the explicit Euler's method only uses the  $y_n$  point in the calculus of the  $y_{n+1}$  point, the Runge Kutta's fourth order method uses four steps.

When it's desirable to obtain the numeric result of an ordinary differential equation, what one wants to identify is how the dependent variable varies according to the change of values of the independent variable. This change of values that the independent variable can assume is characterized as the calculus mesh. Therefore, the independent variable is divided in equidistant points of analysis within a pre-established interval.

The explicit Euler's method is described by the following formula, [1] [6] [7]:

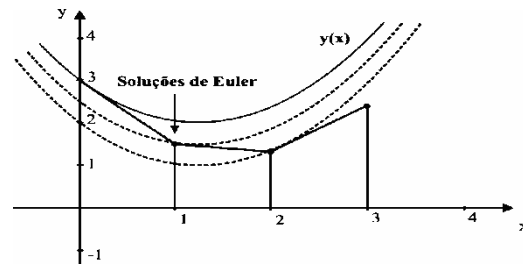
$$y_{n+1} = y_n + hf(x_n, y_n)$$

### Equation 1

Where:  $y_{n+1}$  = mesh's posterior point;  $y_n$  = mesh's anterior point;  $h$  = distance between the mesh's points;  $f(x_n, y_n)$  = equation being analyzed.

Figure 1 shows a delineation of the explicit Euler's method. Through this picture it's possible to see that the computation error propagates in a progressive fashion in the calculus mesh since each point receives the value of the anterior point with an inherent error plus the computational error of the point being calculated. To better the conditions of this method one should use derivatives of higher order to augment its precision. However, it's not

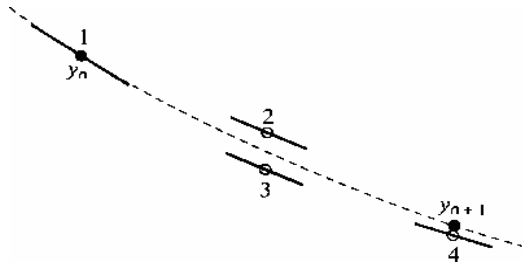
always possible because it involves equations with complex derivatives.



**Figure 1. Explicit Euler's method characterization [3]**

The local truncation error of the explicit Euler's method is from the order of  $O(h^2)$ . The total error originating from this method is measured by the deviation of a point of the solution's curve with relation to the point encountered through the computational numerical simulation – approximated solution as can be seen in Figure 1 [3].

Runge-Kutta's fourth order method is also based in Taylor's formula. However, to avoid the utilization of many derivatives what could make it more precise, the method uses previous calculated points to get the point  $y_{n+1}$ . Figure 2 shows the points where such calculus is done through the Runge-Kutta's fourth order method. In this figure it's possible to see that this method recurs once in  $y_n$  (point 1), recurs twice in  $y_{n+1/2}$  (points 2 and 3) and ultimately recurs once in  $y_{n+1}$  (point 4). Only after the calculation of each one of these recurring points is that the definitive point  $y_{n+1}$  is obtained. The Runge-Kutta's fourth order method's formula is shown below.



**Figure 2. Runge-Kutta's fourth order characterization [8]**

$$\begin{aligned}
 k_1 &= hf(x_n, y_n) \\
 k_2 &= hf\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right) \\
 k_3 &= hf\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right) \\
 k_4 &= hf(x_n + h, y_n + k_3) \\
 y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)
 \end{aligned}$$

**Equation 2**

Where:  $k_1$ ,  $k_2$ ,  $k_3$  and  $k_4$  correspond to each one of the recurring levels.

The truncation error generated by this method is from the order of  $O(h^4)$ . This makes this method more precise than the explicit Euler's method. The total error is calculated similarly to the described for the explicit Euler's method.

## 4 METHODOLOGY

A computational model requires the evidence of its generated results through some parameter. In this informative article were used the analytical solutions to verify the precision of the developed model. This way, the calculated errors were the absolute and the relative in relation to the analytical solution. The equations 3 and 4 show how these errors were calculated [1].

$$E_A = |y - \bar{y}|$$

**Equation 3**

$$E_R = \frac{E_A}{\bar{y}} \times 100$$

**Equation 4**

Where:  $E_A$  = absolute error;  $E_R$  = relative error (%);  $y$  = analytical solution;  $\bar{y}$  = numerical solution.

In this informative article algorithms were developed (explicit Euler's and Runge-Kutta's fourth order) to the resolution of first order ordinary differential equations. This is due the fact that second order differential equations or of greater order can always be transformed in first order differential equation systems. This process is always necessary when someone plans a computational numerical approach since almost all the code to generate approximated numerical solutions of differential equations are written to systems of first order equations [7].

### 4.1 Programs

The source code and executable files can be found online at: <http://lenielmacaferi.blogspot.com/>

## 5 RESULTS

Two differential equations were solved analytically and numerically through the explicit Euler's and Runge-Kutta's fourth order methods. The precision of the developed models is demonstrated graphically. Was admitted a maximum computational error inferior to 1% (or 0.01) to both ordinary differential equations simulated in terms of the two methods.

## 5.1 Differential equation 1

$$\frac{dy}{dx} = y' = x - y + 2$$

Equation 5

### 5.1.1 Initial conditions

$$y(0) = 2;$$
$$\text{Interval } [0; 1];$$
$$h = 0.1.$$

### 5.1.2 Analytical solution (traditional)

$$\text{General: } y = x + 1 + Ce^{-x}$$

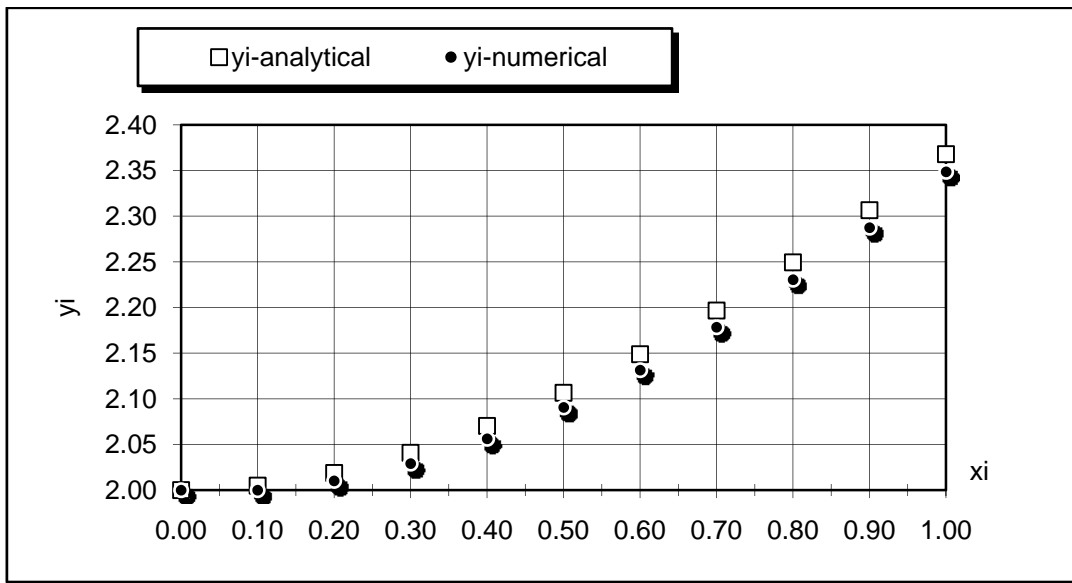
$$\text{Particular: } y = x + 1 + e^{-x}$$

### 5.1.3 Graphical solution

#### 5.1.3.1 Explicit Euler's method

Figure 3 shows the graphical solution in which are compared the results obtained from the traditional analytical solution and the obtained from the computational numerical simulation, all through the explicit Euler's method to the first differential equation analyzed.

We can see that the first obtained results from the computational numerical simulation are coincident with the ones obtained through the traditional analytical solution. However, as the algorithm progresses in the interval  $[0; 1]$  within the calculus mesh, these results show a slight variation. This difference is occasioned by the formulation of the explicit Euler's method that only uses the anterior point to calculate the subsequent point.



**Figure 3. Comparison between the results obtained from the traditional analytical solution and from the computational numerical solution through the explicit Euler's method to the first differential equation analyzed (Equation 5).**

Figure 4 shows the calculus mesh used, the respective obtained values from the computational numerical simulation through the implemented algorithm for the explicit Euler's method and the generated errors in each point of the first differential equation analyzed.

```

C:\ Explicit Euler's method
Explicit Euler's method C Sample Application
Copyright (c)2004 Leniel Braz de Oliveira Macaferi. All rights reserved.
UBM COMPUTER ENGINEERING - 4TH SEMESTER [http://www.ubm.br/]
This program example demonstrates the explicit Euler's method algorithm by
reading input values, making some calculations with them and writing the
results to the screen.
Equations:
1 - y' = x - y + 2 - [0, 1]
2 - y' = exp(-2x) - 2y - [0, 0.2]
Type the equation's number that you want to equate: 1
Type the upper limit: 1
Type the lower limit: 0
Type the number of partitions: 10
Type the initial condition for x: 0
Type the initial condition for y: 2

xi      yi-analytical yi-numerical  AE          RE          PE
0.00    2.000000000  2.000000000  0.000000000  0.000000000  0.00000000%
0.10    2.004837418  2.000000000  0.004837418  0.002418709  0.241870902%
0.20    2.018730753  2.010000000  0.008730753  0.004365825  0.434365825%
0.30    2.040818221  2.029000000  0.011818221  0.005824653  0.582465287%
0.40    2.070320046  2.056100000  0.014220046  0.006916028  0.691602842%
0.50    2.106530660  2.090490000  0.016040660  0.007673158  0.767315783%
0.60    2.148811636  2.131441000  0.017370636  0.008149715  0.814971472%
0.70    2.196585304  2.178296900  0.018288404  0.008395735  0.839573512%
0.80    2.249328964  2.230467210  0.018861754  0.008456414  0.845641399%
0.90    2.306569660  2.287420489  0.019149171  0.008371513  0.837151317%
1.00    2.367879441  2.348678440  0.019201001  0.008175236  0.817523623%

AE - Absolute error
RE - Relative error
PE - Percentage error

Do you want a new calculus? Y/N_

```

Figure 4. Screenshot with the output results of the computational numerical simulation through the explicit Euler's method for the first differential equation analyzed (Equation 5).

### 5.1.3.2 Runge-Kutta's fourth order method

Figure 5 shows the graphical solution in which are compared the results obtained from the traditional analytical solution and the obtained from the computational numerical simulation, all through the Runge-Kutta's fourth order method to the first differential equation analyzed.

It's notable that Runge-Kutta's fourth order method adapted well to the calculus mesh since there wasn't significant difference between the obtained results through the computational numerical simulation and the ones obtained through the traditional analytical solution.

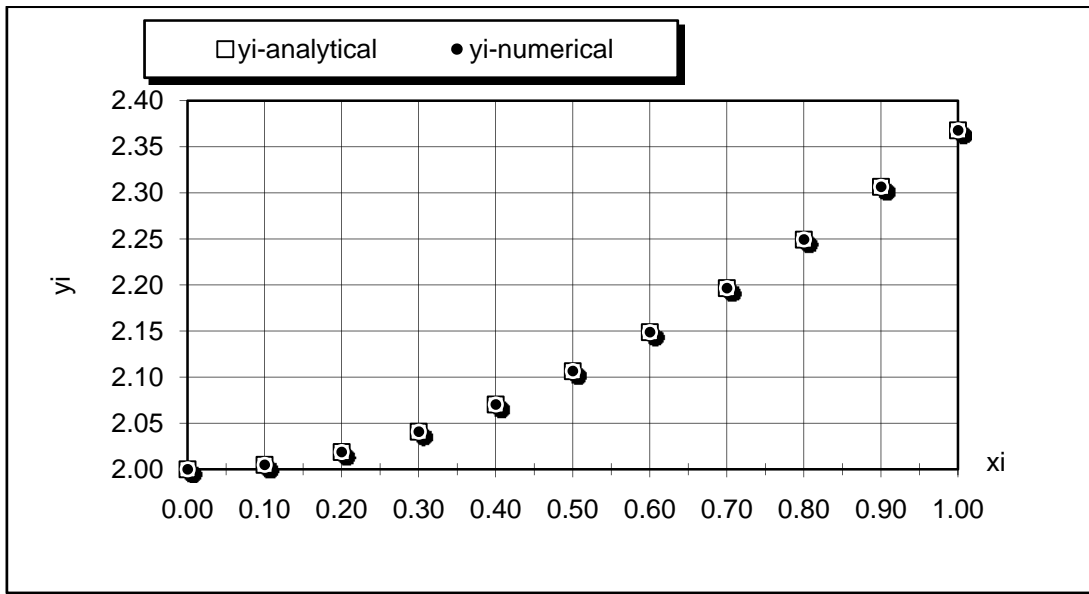


Figure 5. Comparison between the results obtained from the traditional analytical solution and from the computational numerical solution through the Runge-Kutta's fourth order method to the first differential equation analyzed (Equation 5).

Figure 6 shows the calculus mesh used, the respective obtained values from the computational numerical simulation through the implemented algorithm for the Runge-Kutta's fourth order method and the generated errors in each point of the first differential equation analyzed.

```

c:\ Runge-Kuttas's fourth order
Runge-Kuttas's fourth order method C Sample Application
Copyright (c)2004 Leniel Braz de Oliveira Macaferi. All rights reserved.
UBM COMPUTER ENGINEERING - 4TH SEMESTER [http://www.ubm.br/]
This program example demonstrates the Runge-Kuttas's fourth order method
algorithm by reading input values, making some calculations with them and
writing the results to the screen.
Equations:
1 - y' = x - y + 2 - [0, 1]
2 - y' = exp(-2x) - 2y - [0, 0.2]
Type the equation's number that you want to equate: 1
Type the upper limit: 1
Type the lower limit: 0
Type the number of partitions: 10
Type the initial condition for x: 0
Type the initial condition for y: 2

```

xi	yi-analytical	yi-numerical	AE	RE	PE
0.00	2.000000000	2.000000000	0.000000000	0.000000000	0.000000000%
0.10	2.004837418	2.004837500	0.000000082	0.000000041	0.000004088%
0.20	2.018730753	2.018730901	0.000000148	0.000000073	0.000007348%
0.30	2.040818221	2.040818422	0.000000201	0.000000099	0.000009865%
0.40	2.070320046	2.070320289	0.000000243	0.000000117	0.000011732%
0.50	2.106530660	2.106530934	0.000000275	0.000000130	0.000013041%
0.60	2.148811636	2.148811934	0.000000298	0.000000139	0.0000013881%
0.70	2.196585304	2.196585619	0.000000315	0.000000143	0.000014335%
0.80	2.249328964	2.249329290	0.000000326	0.000000145	0.000014476%
0.90	2.306569660	2.306569991	0.000000331	0.000000144	0.000014370%
1.00	2.367879441	2.367879774	0.000000333	0.000000141	0.000014073%

k1	k2	k3	k4	xi + 1	yi_num + 1
0.000000000	0.005000000	0.004750000	0.009525000	0.100000000	2.004837500
0.009516250	0.014040438	0.013814228	0.018134827	0.200000000	2.018730901
0.018126910	0.022220564	0.022015882	0.025925322	0.300000000	2.040818422
0.025918158	0.029622250	0.029437045	0.032974453	0.400000000	2.070320289
0.032967971	0.036319573	0.036151992	0.039352772	0.500000000	2.106530934
0.039346907	0.042379561	0.042227928	0.045124114	0.600000000	2.148811934
0.045118807	0.047862866	0.047725663	0.050346240	0.700000000	2.196585619
0.050341438	0.052824366	0.052700220	0.055071416	0.800000000	2.249329290
0.055067071	0.057313717	0.057201385	0.059346933	0.900000000	2.306569991
0.059343001	0.061375851	0.061274208	0.063215580	1.000000000	2.367879774
0.063212023	0.065051421	0.064959451	0.066716077	1.100000000	2.432871415

```

AE - Absolute error
RE - Relative error
PE - Percentage error
k1 - Initial point of evaluation
k2 - First mid-point of evaluation
k3 - Second mid-point of evaluation
k4 - Final point of evaluation
Do you want a new calculus? Y/N_

```

Figure 6. Screenshot with the output results of the computational numerical simulation through the Runge-Kutta's fourth order method for the first differential equation analyzed (Equation 5).

## 5.2 Differential equation 2

$$\frac{dy}{dx} = y' = e^{-2x} - 2y$$

Equation 6

### 5.2.1 Initial conditions

$$y(0) = 1;$$
$$\text{Interval } [0; 1];$$
$$h = 0.05.$$

### 5.2.2 Analytical solution (traditional)

$$\text{General: } y = e^{-x}(x + C)$$

$$\text{Particular: } y = e^{-x}(x + 1)$$

### 5.2.3 Graphical solution

#### 5.2.3.1 Explicit Euler's method

Figure 7 shows the graphical solution in which are compared the results obtained from the traditional analytical solution and the obtained from the computational numerical simulation, all through the explicit Euler's method to the second differential equation analyzed.

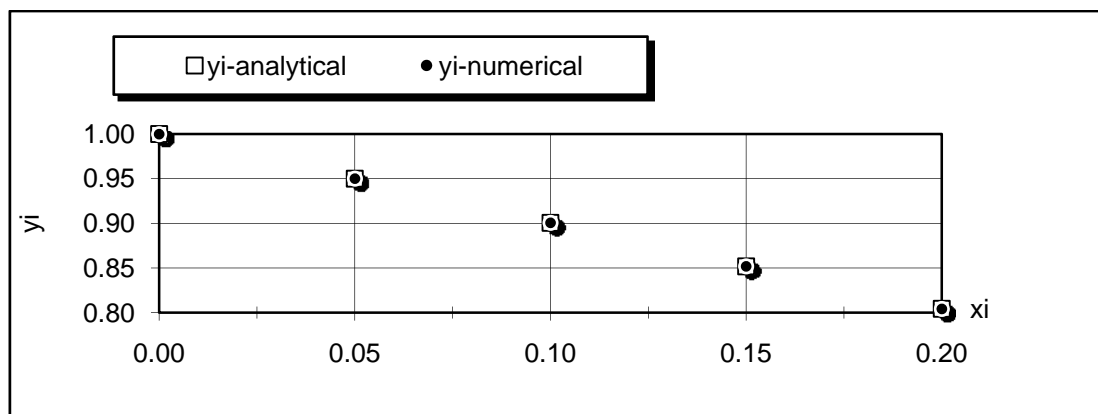


Figure 7. Comparison between the results obtained from the traditional analytical solution and from the computational numerical solution through the explicit Euler's method to the second differential equation analyzed (Equation 6).

Figure 8 shows the calculus mesh used, the respective obtained values from the computational numerical simulation through the implemented algorithm for the explicit Euler's method and the generated errors in each point of the second differential equation analyzed.

```

C:\ Explicit Euler's method
Explicit Euler's method C Sample Application
Copyright (c)2004 Leniel Braz de Oliveira Macaferi. All rights reserved.
UBM COMPUTER ENGINEERING - 4TH SEMESTER [http://www.ubm.br/]
This program example demonstrates the explicit Euler's method algorithm by
reading input values, making some calculations with them and writing the
results to the screen.
Equations:
1 - y' = x - y + 2 - [0, 1]
2 - y' = exp(-2x) - 2y - [0, 0.2]
Type the equation's number that you want to equate: 2
Type the upper limit: 0.2
Type the lower limit: 0
Type the number of partitions: 4
Type the initial condition for x: 0
Type the initial condition for y: 1

xi      yi-analytical yi-numerical  AE          RE          PE
0.00    1.000000000  1.000000000  0.000000000  0.000000000  0.000000000%
0.05    0.950079289  0.950000000  0.000079289  0.000083462  0.008346204%
0.10    0.900603828  0.900241871  0.000361957  0.000402067  0.040206693%
0.15    0.851940954  0.851154221  0.000786732  0.000924312  0.092431230%
0.20    0.804384055  0.803079710  0.001304345  0.001624179  0.162417861%

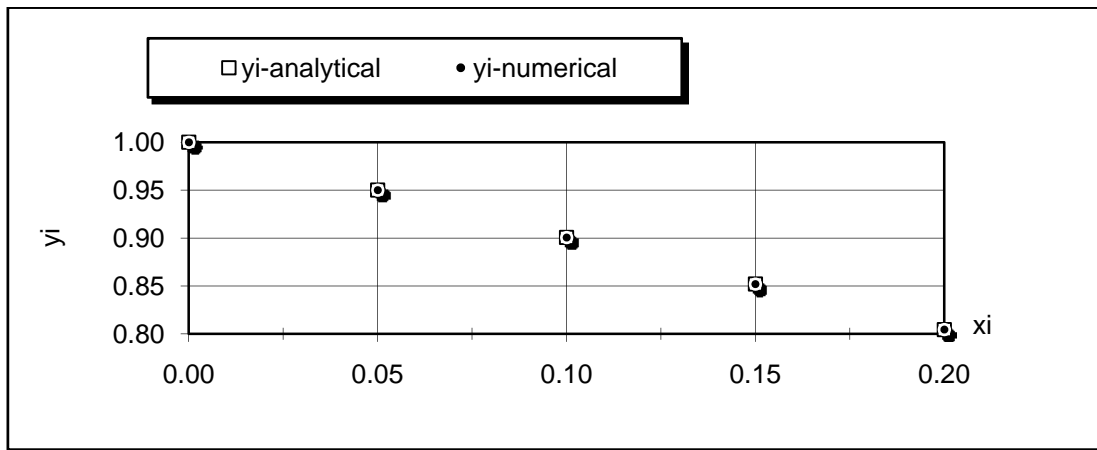
AE - Absolute error
RE - Relative error
PE - Percentage error
Do you want a new calculus? Y/N_

```

Figure 8. Screenshot with the output results of the computational numerical simulation through the explicit Euler's method for the second differential equation analyzed (Equation 6).

### 5.2.3.2 Runge-Kutta's fourth order method

Figure 9 shows the graphical solution in which are compared the results obtained from the traditional analytical solution and the obtained from the computational numerical simulation, all through the Runge-Kutta's fourth order method to the second differential equation analyzed.



**Figure 9. Comparison between the results obtained from the traditional analytical solution and from the computational numerical solution through the Runge-Kutta's fourth order method to the second differential equation analyzed (Equation 6).**

Figure 10 shows the calculus mesh used, the respective obtained values from the computational numerical simulation through the implemented algorithm for the Runge-Kutta's fourth order method and the generated errors in each point of the second differential equation analyzed.

```

Runge-Kuttas's fourth order

Runge-Kuttas's fourth order method C Sample Application
Copyright (c)2004 Leniel Braz de Oliveira Macaferi. All rights reserved.

UBM COMPUTER ENGINEERING - 4TH SEMESTER [http://www.ubm.br/]

This program example demonstrates the Runge-Kuttas's fourth order method
algorithm by reading input values, making some calculations with them and
writing the results to the screen.

Equations:
1 - y' = x - y + 2 - [0, 1]
2 - y' = exp(-2x) - 2y - [0, 0.2]

Type the equation's number that you want to equate: 2
Type the upper limit: 0.2
Type the lower limit: 0
Type the number of partitions: 4
Type the initial condition for x: 0
Type the initial condition for y: 1

xi      yi-analytical yi-numerical  AE          RE          PE
0.00    1.000000000  1.000000000  0.000000000  0.000000000  0.000000000%
0.05    0.950079289  0.950079295  0.000000006  0.000000006  0.000000619%
0.10    0.900603828  0.900603843  0.000000014  0.000000016  0.000001593%
0.15    0.851940954  0.851940978  0.000000025  0.000000029  0.000002877%
0.20    0.804384055  0.804384091  0.000000036  0.000000044  0.000004431%

k1      k2      k3      k4      xi + 1  yi_num + 1
-0.050000000 -0.049938529 -0.049941602 -0.049763969 0.050000000 0.950079295
-0.049766059 -0.049484228 -0.049498319 -0.049121560 0.100000000 0.900603843
-0.049123847 -0.048664153 -0.048687137 -0.048150759 0.150000000 0.851940978
-0.048153187 -0.047552034 -0.047582092 -0.046919886 0.200000000 0.804384091
-0.046922407 -0.046210881 -0.046246457 -0.045487230 0.250000000 0.758163372

AE - Absolute error
RE - Relative error
PE - Percentage error
k1 - Initial point of evaluation
k2 - First mid-point of evaluation
k3 - Second mid-point of evaluation
k4 - Final point of evaluation

Do you want a new calculus? Y/N_

```

Figure 10. Screenshot with the output results of the computational numerical simulation through the Runge-Kutta's fourth order method for the second differential equation analyzed (Equation 6).

## 6 CONCLUSION

Through the results presented, it was observed that the developed programs to both explicit Euler's and Runge-Kutta's fourth order methods adapted to the behavior description of different differential equations in the given calculus mesh. It was found that Runge-Kutta's fourth order method generates punctual errors of smaller value if compared to explicit Euler's errors for both simulated differential equations.

Other first order differential equations can be analyzed through the numerical model here presented. In the same way it's possible to describe industrial process's routines or any other physical phenomena susceptible to mathematical treatment in the mold of differential equations and numerically simulate their respective behavior so that these serve as a comprehensible tool to future evaluations. Anyway, it's necessary a perfect study in topics related to the characterizations of a working routine aiming at the description of such routine in the form of a differential equation.

## 7 BIBLIOGRAPHY

- [1] Ruggiero, Marcia A. Gomes e Lopes, Vera Lucia da Rocha. **Cálculo Numérico (Aspectos Teóricos e Computacionais)**. 2ª ed. São Paulo : Makron Books, 1996. pp. 316 -339.
- [2] Leithold, Louis. **O Cálculo com Geometria Analítica**. 3ª ed. São Paulo : Harbra, 1994. pp. 1.131 - 1.134. Vol. 2.
- [3] Roque, Waldir L. **Introdução ao Cálculo Numérico**. 1ª ed. São Paulo : Atlas, 2000. p. 195.
- [4] Hoffman, Joe D. **Numerical Methods for Engineers and Scientists**. 1st ed. New York : McGraw-Hill, 1992. pp. 279 - 283.
- [5] Schildt, Herbert. **C, Completo e Total**. 3ª ed. São Paulo : Makron Books, 1997. p. 14.
- [6] Barroso, Leônidas Conceição, Barroso, Magali Maria de Araújo e Filho, Frederico Ferreira Campos. **Cálculo Numérico (com Aplicações)**. 2ª ed. São Paulo : Harbra, 1987. pp. 279 - 284.
- [7] Chapra, Steven C. and Canale, Raymond P. **Numerical Methods for Engineers**. 3rd ed. New York : McGraw-Hill, 1998. pp. 676 - 687.
- [8] Press, William H., et al. **Numerical Recipes in Fortran (The Art of Scientific Computing)**. 2nd ed. Cambridge : Cambridge University Press, 1992. p. 706.