UFSM
Universidade Federal de Santa Maria

DELC - Departamento de Eletrônica e Computação
ELC 1021 – Estudo de Casos em Engenharia Elétrica

Solução de Equações Diferenciais Ordinárias Usando Métodos Numéricos

Versão 0.1

Giovani Baratto

Fevereiro de 2007

Índice

Neste texto será apresentado o método de Euler e o método de Runge-Kutta para a solução de equações diferenciais ordinárias.

1 Método de Euler

O método de Euler, também conhecido como método da reta secante, é um dos métodos mais antigos que se conhece para a solução de equações diferenciais ordinárias. Problemas práticos não devem ser resolvidos com o método de Euler. Existem outros métodos que proporcionam resultados com uma melhor precisão e estabilidade se comparados ao método de Euler para o mesmo passo.

1.1 Derivação da Fórmula de Euler

Seja uma função $\frac{dy}{dx} = f(x, y)$ com a condição de contorno $y = y_n$ quando $x = x_n$. Da Figura 1, observa-se que o valor de y_{n+1}, em $x = x_{n+1}$, é dado por:

$$y_{n+1} = y_n + \Delta y \tag{1.1}$$

Do cálculo, pode-se escrever que:

$$dy = \frac{dy}{dx} \cdot dx \tag{1.2}$$

Da equação (1.2), encontra-se uma aproximação para Δy:

$$\Delta y \cong \frac{dy}{dx} \cdot \Delta x \tag{1.3}$$

Das equações (1.1) e (1.3), encontra-se:

$$y_{n+1} = y_n + (x_{n+1} - x_n) \cdot f(x_n, y_n) \tag{1.4}$$

Na Figura 1, observa-se que quanto menor o valor da diferença entre x_{n+1} e x_n (desprezando os erros causados pela representação finita dos números pelos computadores), menor o erro da estimativa para y_{n+1}. Todavia, o número de computações para um intervalo aumenta à medida que a diferença entre x_{n+1} e x_n reduzida. Define-se o passo h como sendo igual a:

$$h = x_{n+1} - x_n \tag{1.5}$$

Usando a equação (1.5) nas equações (1.5) e (1.4), tem-se:

$$x_{n+1} = x_n + h \tag{1.6}$$

e

$$y_{n+1} = y_n + h \cdot f\left(x_n, y_n\right) \tag{1.7}$$

A equação (1.7) é conhecida como fórmula de Euler. A solução de uma equação diferencial pelo método de Euler é realizada pelo uso recursivo das equações (1.6) e (1.7), usando as condições de contorno x_0 e y_0. O erro no método de Euler é da ordem de $O\left(h^2\right)$.

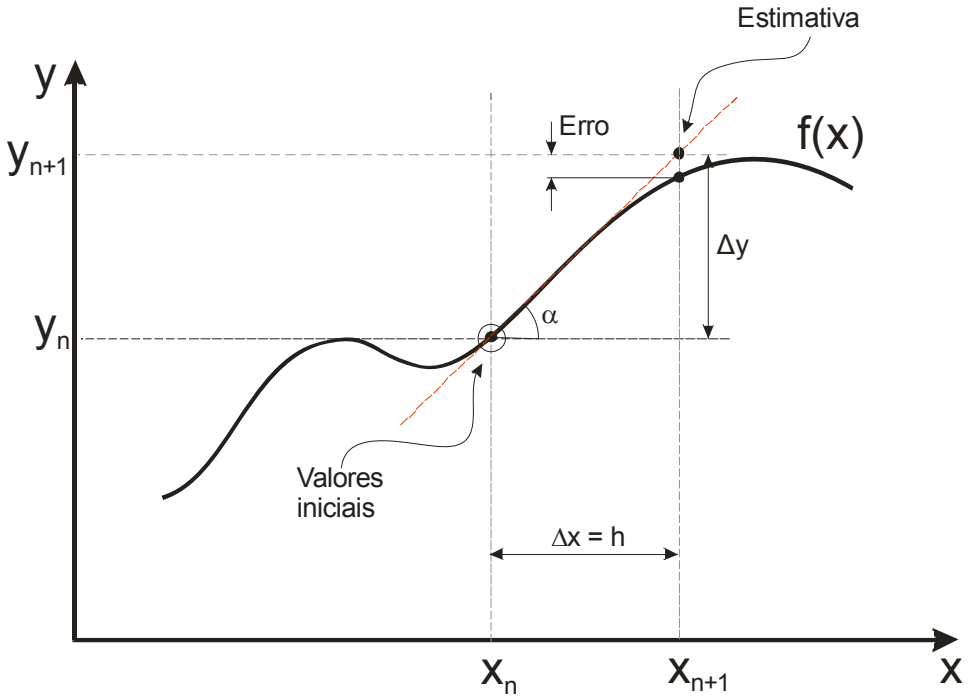

Figura 1 – Ilustração do método de Euler.

1.2 Exemplo Usando o Método de Euler

As seguir será apresentado o método de Euler na solução de uma equação diferencial ordinária de 1ª ordem. A equação escolhida será (Boyce, W. E.; DiPrima, R. C. Equações Diferenciais Elementares e Problemas de Valor de Contorno. ed. 7, pp. 420):

$$\frac{dy}{dx} = 1 - x + 4 \cdot y \tag{1.8}$$

Esta equação será resolvida de $x = 0\,\text{s}$ a $x = 2\,\text{s}$, com a seguinte condição de contorno:

$$y(0) = 1 \tag{1.9}$$

A solução da equação diferencial (1.8) com a condição de contorno (1.9) é conhecida:

$$y(x) = \frac{1}{4} \cdot x - \frac{3}{16} + \frac{19}{16} e^{4 \cdot x} \tag{1.10}$$

A solução numérica é encontrada com a avaliação das equações (1.6) e (1.7):

$$x_{n+1} = x_n + h \tag{1.11}$$

e

$$\begin{aligned} y_{n+1} &= y_n + h \cdot f(x_n, y_n) \\ &= y_n + h \cdot (1 - x + 4 \cdot y) \end{aligned} \tag{1.12}$$

Com a condição de contorno da equação (1.9), temos que $x_0 = 0$ e $y_0 = 1$. Os próximos valores são calculados com o uso recursivo das equações (1.11) e (1.12). O valor do passo é escolhido considerando-se o erro desejado. Neste exemplo, escolhemos $h = 0,001$. A seguir é apresentada uma tabela com alguns valores calculados de y pelo método de Euler e usando a solução algébrica.

Tabela 1 – Resultado da solução numérica da equação (1.8) usando o método de Euler.

n	x_n	y_n	y
0	0,000	1,000000	1,000000
1	0,001	1.005000	1.005010
2	0,002	1.010019	1.010038
3	0,003	1.015057	1.015086
4	0,004	1.020114	1.020153
5	0,005	1.025191	1.025239
...	...	...	...
500	0,500	8.677069	8.712004
1000	1,000	64.382558	64.897803
1500	1,500	473.559790	479.259192
2000	2,000	3484.160803	3540.200110

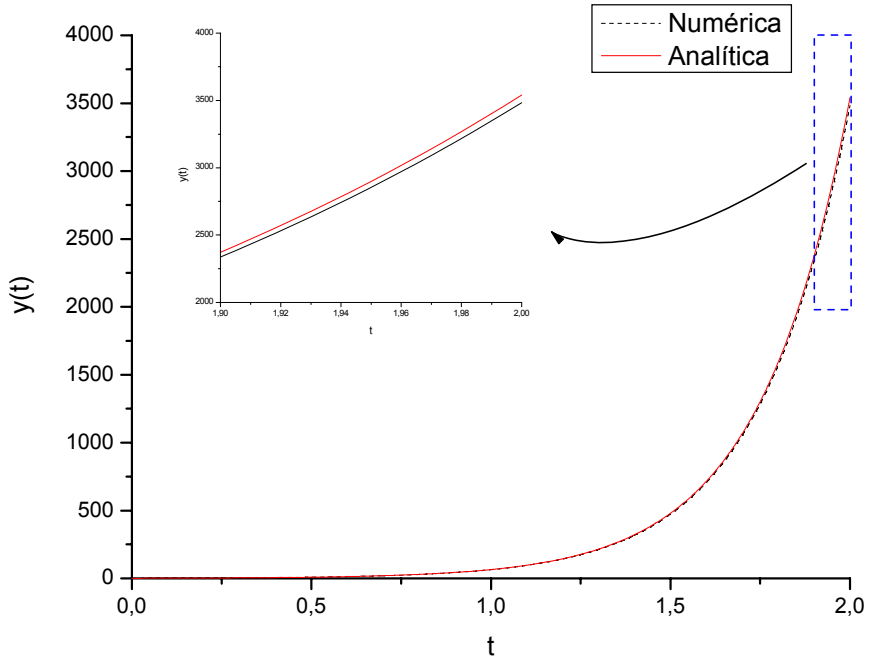

Figura 2 – Gráfico apresentando a solução numérica (método de Euler) e analítica da equação (1.8).

O método pode ser aplicado com o uso de lápis, papel e calculadora. No entanto, este processo é fastidioso pelo número de iterações. A programação em computador é simples. A programação segue os seguintes passos:

01: **Entre** com a função $f(x,y)$.
02: **Entre** com os valores iniciais x_0 e y_0.
03: **Entre** com o passo h.
04: **Enquanto** $x_n < x_{final}$ execute:
Escreva n, x_n, y_n
Avalie $x_{n+1} = x_n + h$
Avalie $y_{n+1} = y_n + h \cdot f(x_n, y_n)$
Faça $x_n = x_{n+1}$ e $y_n = y_{n+1}$
06: Fim.

Um programa escrito em C é apresentado a seguir. Este programa foi escrito para ser didático, podendo ser melhorado.

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double y(double x) /* a solução y(t) da equação */
{
       return (1.0/4.0)*x-(3.0/16.0)+(19.0/16.0)*exp(4.0*x);
}

double f(double x, double y) /* a função f(x,y)*/
{
       return 1-x+4*y;
}

int main(int arc, char* argv[])
{
    double xn, xn1, xmax; /* variáveis tn e tn+1 */
    double yn, yn1; /* variáveis yn e yn+1 */
    double y0, x0;  /* valores iniciais de y e t */
    double h;       /* passo */
    int n;

    x0 = 0.0;    /* valor incial para t */
    y0 = 1.0;    /* valor inicial para y */
    xmax = 2.0;  /* valor máximo para t */
    h = 0.001;   /* o valor do passo */
    xn = t0;
    yn = y0;
    n = 0;       /* numero de iterações */
    while(xn < xmax){
             printf("%i %f %f %f\n",n,xn,yn,y(xn)); /*escreva os valores das variáveis*/
             yn1 = yn + h*f(xn,yn); /* estime yn+1 pelo método de Euler */
             xn1 = xn + h;
               n = n+1;   /* Atribua os valores para a próxima iteração */
              yn = yn1;
              xn = xn1;
    }/* while */
    return 0; /* termina a computação */
}/* main */
```

Ilustração 1 – Exemplo de um programa em C para a solução da equação (1.8) pelo método de Euler.

2 Método de Runge-Kutta

O método de Runge-Kutta pode ser entendido como um aperfeiçoamento do método de Euler, com uma melhor estimativa da derivada da função. No método de Euler a estimativa do valor de y_{n+1} é realizado com o valor de y_n e com a derivada no ponto x_n. No método de Runge-Kutta, busca-se uma melhor estimativa da derivada com a avaliação da função em mais pontos no intervalo $[x_n, x_{n+1}]$. Um método de Runge-Kutta de ordem n possui um erro da ordem de $O\left(h^{n+1}\right)$. O método de Runge-Kutta de 4ª ordem é o mais usado na solução numérica de problemas com equações diferenciais ordinárias.
A seguir será discutido o método de Runge-Kutta de 2ª ordem, ilustrado pela Figura 3. No método de Euler de passo h, a estimativa de y_{n+1} é realizada com os valores de x_n e da derivada de y_n. No método de Runge-Kutta de 2ª ordem, o valor da estimativa de y_{n+1} é encontrado com o valor de y_n e com uma estimativa da derivada em um ponto mais próximo de x_{n+1}, em $x_n + \frac{h}{2}$:

$$y_{n+1} = y_n + h \cdot f\left(x_n + \frac{1}{2}h, y_{n+\frac{1}{2}} \right) \tag{1.13}$$

Na equação (1.13), $y_{n+\frac{1}{2}}$ é o valor de y em $x_n + \frac{1}{2}h$. Uma estimativa do valor de $y_{n+\frac{1}{2}}$ é encontrado com o auxílio do método de Euler:

$$y_{n+\frac{1}{2}} = y_n + \frac{h}{2} \cdot f\left(x_n, y_n \right) \tag{1.14}$$

Denominando:

$$\begin{aligned} k_1 &= h \cdot f\left(x_n . y_n \right) \\ k_2 &= h \cdot f\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1 \right) \end{aligned} \tag{1.15}$$

Escreve-se a equação (1.13) como:

$$y_{n+1} = y_n + k_2 \tag{1.16}$$

No método de Runge-Kutta de 2ª ordem, avaliam-se as equações (1.15) e (1.16).

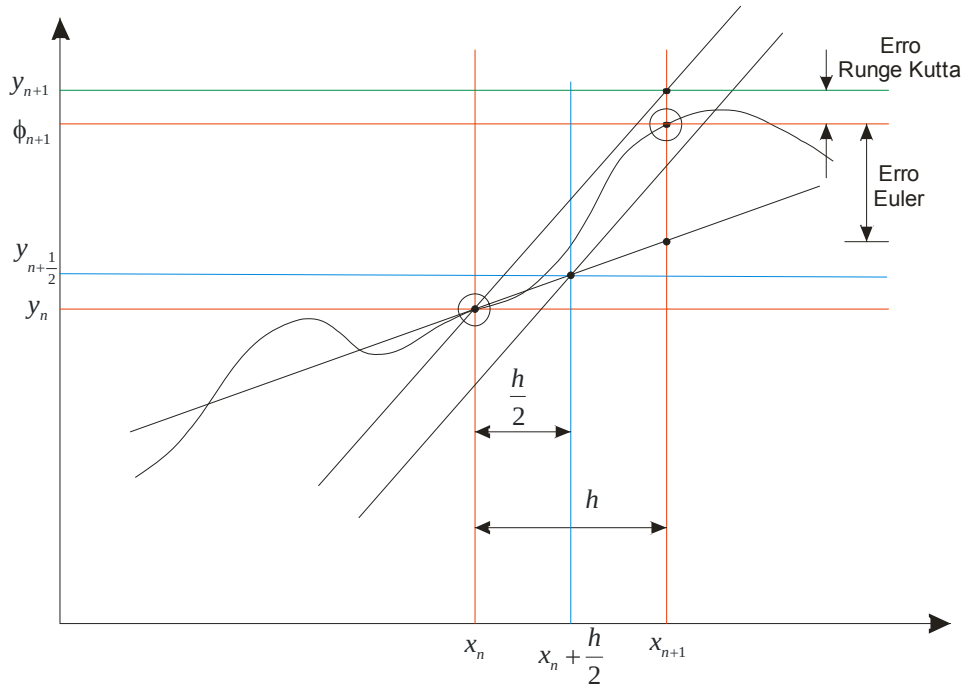

Figura 3 – Ilustração do método de Runge-Kutta de 2ª ordem.

O método de Runge-Kutta de 4ª ordem tem as seguintes equações:

$$
\begin{aligned}
k_1 &= h \cdot f\left(x_n, y_n\right) \\
k_2 &= h \cdot f\left(x_n + \frac{h}{2}, y_n + \frac{1}{2}k_1\right) \\
k_3 &= h \cdot f\left(x_n + \frac{h}{2}, y_n + \frac{1}{2}k_2\right) \\
k_4 &= h \cdot f\left(x_n + h, y_n + k_3\right) \\
y_{n+1} &= y_n + \frac{1}{6}\left(k_1 + 2 \cdot k_2 + 2 \cdot k_3 + k_4\right)
\end{aligned}
\tag{1.17}
$$

e

$$x_{n+1} = x_n + h \tag{1.18}$$

2.1 Exemplo Usando o Método de Runge-Kutta

A seguir será apresentado um exemplo usando o método de Runge-Kutta na solução de uma equação diferencial. Será usada a equação diferencial (1.8) com as condições iniciais dada por (1.9), a mesma equação que foi usada no Exemplo usando o método de Euler. O passo neste exemplo será reduzido para 0,01 s. A solução da equação diferencial é encontrada pelo uso iterativo das equações (1.17) e (1.18). A seguir é apresentada uma tabela com os valores calculados e com o valor analítico da solução.

Tabela 2 – Resultado da solução numérica da equação (1.8) usando o método de Runge-Kutta.

n	x_n	y_n	y
0	0,000	1,000000	1,000000
1	0,001	1.050963	1.050963
2	0,002	1.103903	1.103903
3	0,003	1.158903	1.158903
4	0,004	1.216044	1.216044
5	0,005	1.275416	1.275416
...	...		
50	0,500	8.712004	8.712004
100	1,000	64.897798	64.897803
150	1,500	479.259133	479.259192
200	2,000	3540.199525	3540.200110

Comparando-se os resultados da solução numérica usando o método de Euler (Tabela 1) com os resultados da solução usando o método de Runge-

Kutta (Tabela 2), observa-se que neste segundo método a precisão é maior, mesmo com o uso de um passo 10 vezes maior.
A seguir é apresentado um programa escrito em C, para a solução numérica da equação (1.8).

```c
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

double y(double x) /* a solução y(x) da equação */
{
       return (1.0/4.0)*x-(3.0/16.0)+(19.0/16.0)*exp(4.0*x);
}

double f(double x, double y)    /* a função f(x,y)*/
{
       return 1-x+4*y;
}

int main(int argc, char* argv[])
{
    double xn, xn1, xmax; /* variáveis tn e tn+1 */
    double yn, yn1; /* variáveis yn e yn+1 */
    double y0, t0;  /* valores iniciais de y e t */
    double h;       /* passo */
    double k1, k2, k3, k4;   /* variáveis auxiliares */
    int n;          /* número de iterações */

    x0 = 0.0;    /* valor incial para t */
    y0 = 1.0;    /* valor inicial para y */
    xmax = 2.0;  /* valor máximo para t */
    h = 0.01;   /* o valor do passo */
    xn = t0;
    yn = y0;
    n = 0;       /* numero de iterações */
    while(xn < xmax){
             printf("%i %f %f %f\n",n,tn,yn,y(tn)); /*escreva os valores das variáveis*/
             k1 = h*f(xn,yn);                /* aplica o método de Runge-Kutta */
             k2 = h*f(xn+h/2, yn+k1/2.0);
             k3 = h*f(xn+h/2, yn+k2/2.0);
             k4 = h*f(xn+h, yn+k3);
             xn1 = xn + h;
             yn1 = yn + (k1+2*k2+2*k3+k4)/6.0;
             n = n+1; /* atribua os valores para a próxima iteração */
             xn = xn1;
             yn = yn1;

    }/* laço while */
    return 0; /* termina a computação */
}/* função main */
```

Ilustração 2 – Exemplo de um programa em C para a solução da equação (1.8) pelo método de Runge-Kutta.

3 Equações Ordinárias de Ordem de 2ª Ordem

Equações diferenciais de 2ª ordem, ou de ordem superior, podem ser reduzidas a um conjunto de equações diferenciais de 1ª ordem. Este conjunto de equações de 1ª ordem pode ser resolvido pelo método de Runge-Kutta ou outro método numérico. Seja uma equação diferencial de 2ª ordem:

$$\frac{d^2y}{dx^2} + q(x)\frac{dy}{dx} = r(x,y) \tag{1.19}$$

Defini-se uma variável auxiliar z:

$$z(x,y) = \frac{dy}{dx} \tag{1.20}$$

A equação (1.19) é escrita como um conjunto de duas equações diferenciais de 1ª ordem, com o auxílio da variável $z(x,y)$, definida na equação (1.20).

$$\begin{aligned} \frac{dy}{dx} &= z(x,y) \\ \frac{dz}{dx} &= r(x,y) - q(x)\cdot z(x) \end{aligned} \tag{1.21}$$

O conjunto de equações em (1.21) pode ser escrito como:

$$\begin{aligned} \frac{dy}{dx} &= f(x,y,z) \\ \frac{dz}{dx} &= g(x,y,z) \end{aligned} \tag{1.22}$$

Para a solução da equação diferencial ordinária de 2ª ordem (1.19), é necessário que as equações diferenciais ordinárias de primeira ordem, apresentadas em (1.22) seja resolvidas. A seguir, apresenta-se a solução destas equações, usando o método de Euler e o método de Runge-Kutta.

3.1 Solução Usando o Método de Euler

As equações em (1.22) são resolvidas pelo método de Euler, usando iterativamente as equações apresentadas a seguir:

$$\begin{aligned} y_{n+1} &= y_n + h\cdot f(x_n, y_n, z_n) \\ z_{n+1} &= z_n + h\cdot g(x_n, y_n, z_n) \end{aligned} \tag{1.23}$$

Exemplo: Resolva a seguinte equação diferencial de 2ª ordem, usando o método de Euler.

$$\frac{d^2y}{dx^2} + 30\cdot\frac{dy}{dx} + 200\cdot y = 1000 \tag{1.24}$$

Resolva a equação (1.24) com x variando de 0 a 1. Considere as seguintes condições de contorno:

$$\begin{aligned} y(0) &= 0 \\ \left.\frac{dy}{dx}\right|_{t=0} &= 0 \end{aligned} \tag{1.25}$$

O primeiro passo é reduzir esta equação diferencial de 2ª ordem para um conjunto de 2 equações diferenciais ordinárias de 1ª ordem:

$$\begin{aligned} \frac{dy}{dx} &= z \\ \frac{dz}{dx} &= 1000 - 30 \cdot z - 200 \cdot y \end{aligned} \tag{1.26}$$

O conjunto de equações em (1.26) é resolvido usando as expressões em (1.23) . De acordo com as condições iniciais em (1.25), tem-se $x_0 = 0$ e $y_0 = 0$. O passo usando neste exemplo será $h = 0{,}01\,\text{s}$. A seguir é apresentada uma tabela com os valores calculados numericamente e obtidos da solução analítica, que para este exemplo é igual a:

$$y(t) = 5 - 10 \cdot e^{-10 \cdot x} + 5 \cdot e^{-20 \cdot x} \tag{1.27}$$

Tabela 3 – Resultado da solução numérica da equação (1.24), usando o método de Euler.

n	x_n	y_n	y
0	0,0	0.000000	0.000000
1	0,01	0.000000	0.045280
2	0,02	0.100000	0.164293
3	0,03	0.270000	0.335876
4	0,04	0.487000	0.543444
5	0,05	0.733500	0.774091
...	...		
50	0,5	4.948534	4.932848
70	0,7	4.993735	4.990885
90	0,9	4.999238	4.998766
100	1,0	4.999734	4.999546

A seguir é apresentado um programa em C que implementa esta solução.

```c
#include <stdlib.h> /* biblioteca de funções */
#include <stdio.h>
#include <math.h>


double y(double x)  /* A solução analítica de y(x)*/
{
	return 5.0-10.0*exp(-10.0*x)+5.0*exp(-20.0*x);
}
```

```c
double f(double x, double y, double z)  /* a função f(x,y,z)=dy/dx */
{
       return z;
}

double g(double x, double y, double z)  /* a função g(x,y,z)=dz/dx */
{
       return 1000.0-30.0*z-200.0*y;
}

int main(int arc, char* argv[])
{
    double xn, xn1, xmax;    /* declaração de xn, xn+1 e máximo valor de x na simulação
*/
    double yn, yn1;          /* declaração de yn e yn+1 */
    double zn, zn1;          /* declaração de zn e zn+1 */
    double h;                /* passo */
    double x0,y0,z0;         /* valores iniciais de x, y e z */
    int n;                   /* número de iterações */

    x0 = 0.0; /* atribuição dos valores de contorno */
    y0 = 0.0;
    z0 = 0.0;
    h = 0.01;
    xmax = 1.0;

    n = 0; /* inicializa as variáveis com os valores iniciais*/
    xn = x0;
    yn = y0;
    zn = z0;

    while(xn<xmax){
        printf("%i %f %f %f\n",n,xn,yn,y(xn));  /* escreve os valores das variáveis */

        xn1 = xn + h;   /* realiza uma iteração do método de Runge-Kutta */
        yn1 = yn + h * f(xn,yn,zn);
        zn1 = zn + h * g(xn,yn,zn);

        n = n + 1;      /* atualiza as variáveis para a próxima iteração */
        xn = xn1;
        yn = yn1;
        zn = zn1;
    }/* while */
    return 0;   /* termina a execução do programa */
}/* main */
```

Ilustração 3 – Programa em C usando o método de Euler para a solução da equação (1.24), uma equação diferencial ordinária de 2ª ordem.

3.2 Solução Usando o Método de Runge-Kutta

As equações em (1.22) são resolvidas pelo método de Runge-Kutta, usando iterativamente as equações apresentadas a seguir:

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{6}\left(k_1 + 2\cdot k_2 + 2\cdot k_3 + k_4\right) \\ z_{n+1} &= z_n + \frac{1}{6}\left(l_1 + 2\cdot l_2 + 2\cdot l_3 + l_4\right) \end{aligned} \tag{1.28}$$

onde

$$
\begin{aligned}
k_1 &= h \cdot f\left(x_n, y_n, z_n\right) \\
l_1 &= h \cdot g\left(x_n, y_n, z_n\right) \\
k_2 &= h \cdot f\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1, z_n + \frac{1}{2}l_1\right) \\
l_2 &= h \cdot g\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1, z_n + \frac{1}{2}l_1\right) \\
k_3 &= h \cdot f\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2, z_n + \frac{1}{2}l_2\right) \\
l_3 &= h \cdot g\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}, z_n + \frac{1}{2}l_2\right) \\
k_4 &= h \cdot f\left(x_n + h, y_n + k_3, z_n + l_3\right) \\
l_4 &= h \cdot g\left(x_n + h, y_n + k_3, z_n + l_3\right)
\end{aligned}
\tag{1.29}
$$

Exemplo: Resolva a seguinte equação diferencial de 2ª ordem, usando o método de Runge-Kutta.

$$
\frac{d^2y}{dx^2} + 30 \cdot \frac{dy}{dx} + 200 \cdot y = 1000 \tag{1.30}
$$

Resolva a equação (1.24), para x entre 0 a 1. Considere as seguintes condições de contorno:

$$
\begin{aligned}
y(0) &= 0 \\
\left.\frac{dy}{dx}\right|_{t=0} &= 0
\end{aligned}
\tag{1.31}
$$

O primeiro passo é reduzir esta equação diferencial de 2ª ordem para um conjunto de 2 equações diferenciais ordinárias de 1ª ordem:

$$
\begin{aligned}
\frac{dy}{dx} &= z \\
\frac{dz}{dx} &= 1000 - 30 \cdot z - 200 \cdot y
\end{aligned}
\tag{1.32}
$$

O conjunto de equações em (1.32) é resolvido usando as expressões em(1.29). De acordo com as condições iniciais em (1.31) , tem-se $x_0 = 0$ e $y_0 = 0$. O passo usando neste exemplo será $h = 0{,}01$. A seguir é apresentada uma tabela com os valores calculados numericamente e obtidos da solução analítica, que para este exemplo é igual a:

$$
y(t) = 5 - 10 \cdot \mathrm{e}^{-10 \cdot x} + 5 \cdot \mathrm{e}^{-20 \cdot x} \tag{1.33}
$$

Tabela 4 – Resultado da solução numérica da equação(1.30), usando o método de Runge-Kutta.

n	x_n	y_n	y
0	0,0	0.000000	0.000000
1	0,01	0.016667	0.045280
2	0,02	0.123528	0.164293
3	0,03	0.293960	0.335876
4	0,04	0.507390	0.543444
5	0,05	0.748036	0.774091
...	...		
50	0,5	4.943575	4.932848
70	0,7	4.992869	4.990885
90	0,9	4.999100	4.998766
100	1,0	4.999680	4.999546

A seguir é apresentado um programa em C que implementa esta solução.

```
#include <stdlib.h> /* biblioteca de funções */
#include <stdio.h>
#include <math.h>


double y(double x)  /* A solução analítica de y(x)*/
{
	return 5.0-10.0*exp(-10.0*x)+5.0*exp(-20.0*x);
}

double f(double x, double y, double z)  /* a função f(x,y,z)=dy/dx */
{
	return z;
}

double g(double x, double y, double z)  /* a função g(x,y,z)=dz/dx */
{
	return 1000.0-30.0*z-200.0*y;
}

int main(int arc, char* argv[])
{
    double xn, xn1, xmax;   /* declaração de xn, xn+1 e val máximo de x na simulação */
    double yn, yn1;         /* declaração de yn e yn+1 */
    double zn, zn1;         /* declaração de zn e zn+1 */
    double h;               /* passo */
    double x0,y0,z0;        /* valores iniciais de x, y e z */
    double k1, k2, k3, k4;  /* variáveis auxiliares do método de Runge-Kutta */
    double l1, l2, l3, l4;  /* variáveis auxiliares do método de Runge-Kutta */
    int n;                  /* número de iterações */

    x0 = 0.0;   /* atribuição dos valores de contorno */
    y0 = 0.0;
    z0 = 0.0;
    h = 0.01;
    xmax = 1.0;

    n = 0;  /* inicializa as variáveis com os valores iniciais*/
    xn = x0;
    yn = y0;
    zn = z0;

    while(xn<xmax){
                printf("%i %f %f %f\n",n,xn,yn,y(xn));     /* escreve os valores das
variáveis */

                k1 = h*f(xn, yn, zn); /* realiza uma iteração do método Runge-Kutta */
```

```
                l1 = h*g(xn, yn, zn);
                k2 = h*f(xn+(1/2)*h, yn+(1/2)*k1, zn+(1/2)*l1);
                l2 = h*g(xn+(1/2)*h, yn+(1/2)*k1, zn+(1/2)*l1);
                k3 = h*f(xn+(1/2)*h, yn+(1/2)*k2, zn+(1/2)*l2);
                l3 = h*g(xn+(1/2)*h, yn+(1/2)*k2, zn+(1/2)*l2);
                k4 = h*f(xn+h, yn+k3, zn+l3);
                l4 = h*g(xn+h, yn+k3, zn+l3);

                xn1 = xn + h;
                yn1 = yn + (1/6.0)*(k1 + 2*k2 + 2*k3 + k4);
                zn1 = zn + (1/6.0)*(l1 + 2*l2 + 2*l3 + l4);


                n = n + 1;    /* atualiza as variáveis para a próxima iteração */
                xn = xn1;
                yn = yn1;
                zn = zn1;
    }/* while */
    return 0;   /* termina a execução do programa */
}/* main */
```

Ilustração 4 – Programa em C usando o método de Euler para a solução da equação (1.30), uma equação diferencial ordinária de 2ª ordem.