# MS211 - Cálculo Numérico

## Turma K - Prof João Batista Florindo

## Projeto 2

*Artur Lima (166916)*

*João Pedro Martins (176117)*

## Introdução

O objetivo deste projeto é analisar e comparar a performance computacional de dois métodos numéricos de passo um utilizados para se obter uma aproximação da solução de problemas de valor inicial (PVI) relacionados a equações diferenciais ordinárias (EDOs). Esses métodos recebem como *input* um PVI da forma $y' = f(x, y)$, $y(x_0) = y_0$ e uma variável *h* chamada de *passo* e produz como *output* vários pontos num intervalo desejado, que formam o gráfico da solução do PVI.

Os métodos estudados neste projeto são o método de Runge-Kutta de primeira ordem, mais conhecido como método de Euler, e o método de Runge-Kutta de quarta ordem. O PVI em que eles serão aplicados é o PVI da equação logística estudada por Pierre-François Verhulst que relaciona a taxa de crescimento populacional com a população existente e a quantidade de recursos disponíveis no meio, com um valor inicial: $y'(t) = ry(t)(1 - \frac{y(t)}{K})$, $y(0) = 1$

Além de comparados entre si, os resultados dos métodos serão comparados também à solução analítica $y(t) = \frac{Ky_0e^{rt}}{K+y_0(e^{rt}-1)}$, para verificar o quão próximo estão do resultado correto.

**Método de Euler**

O método de Euler é o método de Runge-Kutta mais simples. Consiste na ideia de que, a partir dos pontos iniciais $x_0$ e $y_0$ e da EDO, é possível se obter uma boa aproximação para $y_1$ utilizando um ponto $x_1$ e a equação da reta tangente em $(x_0, y_0)$ desde que $x_1$ esteja suficientemente próximo de $x_0$. Essa ideia pode ser aplicada sucessivamente para se obter quantos pontos desejar. Esses pontos, devido à sua natureza, são uma boa aproximação para os pontos da própria solução do PVI.

O método parte dos valores (conhecidos) de $x_0$, $y_0$ e, consequentemente, de $f(x_0, y_0) = y'(x_0)$ e utiliza as relações $x_{k+1} = x_k + h$ e $y_{k+1} = y_k + hf(x_k, y_k)$ obtidas através da equação da reta tangente $y = y_0 + f(x_0, y_0)(x - x_0)$ para obter cada um dos pontos da aproximação, dentro de um intervalo desejado.

**Método de Runge-Kutta**

Os métodos de Runge-Kutta são métodos de passo simples ($y_{k+1}$ só depende de $x_k$ e $y_k$). Eles não dependem do cálculo de qualquer derivada de $f$. Porém, para suprir essa falta de cálculo da derivada de $f$, é necessário uma função $\phi$ definida avaliando $f$ em diferentes pontos.

Especificamente, os métodos de Runge-Kutta são dados por:

$y_{k+1} = y_k + h\varphi(x_k, y_k), \forall k = 0, 1, \ldots,$ em que φ é uma função de x e y que depende indiretamente de $f$ e do tamanho do passo h (que varia para cada problema).

O método de Runge-Kutta varia dependendo da ordem que é usada. No projeto, usaremos o método de Runge-Kutta de ordem 4 para encontrar a solução numérica da equação logística, em um intervalo de tempo de 0 até 10.

Quando utilizamos o método na ordem 4, temos que $y_{k+1} = y_k + (h \div 6) \times (k_1 + 2k_2 + 2k_3 + k_4)$, onde:

$$k_1 = f(x_k, y_k),$$

$$k_2 = f(x_k + h \div 2, y_k + k_1 \times (h \div 2)),$$

$$k_3 = f(x_k + h \div 2, y_k + k_2 \times (h \div 2)),$$

$$k_4 = f(x_k + h, y_k + h \times k_3).$$

## Item a

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

// Retorna o valor da EDO para os parametros t, y, r, k
double dydx(double t, double y, double r, double k){
    return(r*y*(1-(y/k)));
}
// Solucao analitica do PVI
double analitica(double t, double y0, double r, double k){

    return k*y0*exp(r*t) / (k + y0*(exp(r*t)-1));
}
// Metodo de Euler com valores iniciais t0 e y0, step h e intervalo [t0,limit]
void euler(double t0, double y0, double h, double limit){
    printf("\nt\t\ty_euler\t\ty_analitica\n");
    double t = t0;
    double y = y0;
    while(t <= limit){
        printf("%lf\n", t, y, analitica(t, y0, 0.5, 10));
        y += h*dydx(t, y, 0.5, 10);
        t += h;
    }
}

// Funcao principal
int main(){
    // Chamada do metodo com intervalo [t0,limit] = [0,4], y0 = 1, h = 0.05
    euler(0, 1, 0.05, 4);
    return 0;
}
```

**Implementação em C do método de Euler**

Aplicando o método de Euler no intervalo [0, 4] com $r = 0.5$, $K = 10$, $h = 0.05$ e $y_0 = 1$ ao PVI estudado, obtivemos os seguintes pontos:

| | Método de Euler | Solução analítica |
|---|---|---|
| t | y | y |
| 0.0000 | 1.0000 | 1.0000 |
| 0.0500 | 1.0225 | 1.0227 |
| 0.1000 | 1.0454 | 1.0459 |
| 0.1500 | 1.0689 | 1.0696 |
| 0.2000 | 1.0927 | 1.0937 |
| 0.2500 | 1.1171 | 1.1183 |
| 0.3000 | 1.1419 | 1.1433 |
| 0.3500 | 1.1671 | 1.1689 |
| 0.4000 | 1.1929 | 1.1949 |
| 0.4500 | 1.2192 | 1.2215 |
| 0.5000 | 1.2459 | 1.2486 |
| 0.5500 | 1.2732 | 1.2761 |
| 0.6000 | 1.3010 | 1.3042 |
| 0.6500 | 1.3293 | 1.3328 |
| 0.7000 | 1.3581 | 1.3620 |
| 0.7500 | 1.3874 | 1.3917 |
| 0.8000 | 1.4173 | 1.4219 |
| 0.8500 | 1.4477 | 1.4527 |
| 0.9000 | 1.4787 | 1.4840 |
| 0.9500 | 1.5102 | 1.5158 |
| 1.0000 | 1.5422 | 1.5483 |
| 1.0500 | 1.5748 | 1.5813 |
| 1.1000 | 1.6080 | 1.6148 |
| 1.1500 | 1.6417 | 1.6490 |
| 1.2000 | 1.6761 | 1.6837 |
| 1.2500 | 1.7109 | 1.7190 |
| 1.3000 | 1.7464 | 1.7549 |
| 1.3500 | 1.7824 | 1.7913 |
| 1.4000 | 1.8190 | 1.8284 |
| 1.4500 | 1.8562 | 1.8660 |
| 1.5000 | 1.8940 | 1.9043 |
| 1.5500 | 1.9324 | 1.9431 |
| 1.6000 | 1.9714 | 1.9826 |

| | | |
|---|---|---|
| 1.6500 | 2.0110 | 2.0226 |
| 1.7000 | 2.0511 | 2.0632 |
| 1.7500 | 2.0919 | 2.1045 |
| 1.8000 | 2.1332 | 2.1463 |
| 1.8500 | 2.1752 | 2.1888 |
| 1.9000 | 2.2178 | 2.2318 |
| 1.9500 | 2.2609 | 2.2755 |
| 2.0000 | 2.3046 | 2.3197 |
| 2.0500 | 2.3490 | 2.3645 |
| 2.1000 | 2.3939 | 2.4100 |
| 2.1500 | 2.4394 | 2.4560 |
| 2.2000 | 2.4855 | 2.5026 |
| 2.2500 | 2.5322 | 2.5498 |
| 2.3000 | 2.5795 | 2.5976 |
| 2.3500 | 2.6274 | 2.6459 |
| 2.4000 | 2.6758 | 2.6949 |
| 2.4500 | 2.7248 | 2.7444 |
| 2.5000 | 2.7743 | 2.7944 |
| 2.5500 | 2.8245 | 2.8450 |
| 2.6000 | 2.8751 | 2.8962 |
| 2.6500 | 2.9263 | 2.9479 |
| 2.7000 | 2.9781 | 3.0002 |
| 2.7500 | 3.0304 | 3.0529 |
| 2.8000 | 3.0832 | 3.1062 |
| 2.8500 | 3.1365 | 3.1600 |
| 2.9000 | 3.1903 | 3.2143 |
| 2.9500 | 3.2446 | 3.2690 |
| 3.0000 | 3.2994 | 3.3243 |
| 3.0500 | 3.3547 | 3.3800 |
| 3.1000 | 3.4104 | 3.4362 |
| 3.1500 | 3.4666 | 3.4928 |
| 3.2000 | 3.5232 | 3.5498 |
| 3.2500 | 3.5803 | 3.6072 |
| 3.3000 | 3.6377 | 3.6651 |
| 3.3500 | 3.6956 | 3.7233 |
| 3.4000 | 3.7538 | 3.7819 |
| 3.4500 | 3.8124 | 3.8409 |
| 3.5000 | 3.8714 | 3.9002 |
| 3.5500 | 3.9307 | 3.9598 |

| | | |
|---|---|---|
| 3.6000 | 3.9904 | 4.0198 |
| 3.6500 | 4.0503 | 4.0800 |
| 3.7000 | 4.1106 | 4.1406 |
| 3.7500 | 4.1711 | 4.2013 |
| 3.8000 | 4.2319 | 4.2624 |
| 3.8500 | 4.2929 | 4.3236 |
| 3.9000 | 4.3542 | 4.3851 |
| 3.9500 | 4.4156 | 4.4467 |
| 4.0000 | 4.4773 | 4.5085 |

**Tabela 1: método de Euler com h = 0.05 e solução analítica**

Plotando o gráfico dos pontos tabelados utilizando a ferramenta online *plot.ly*, é possível comparar melhor o resultado do método de Euler com a solução analítica para analisar o quão bom foi a aproximação gerada por esse método:

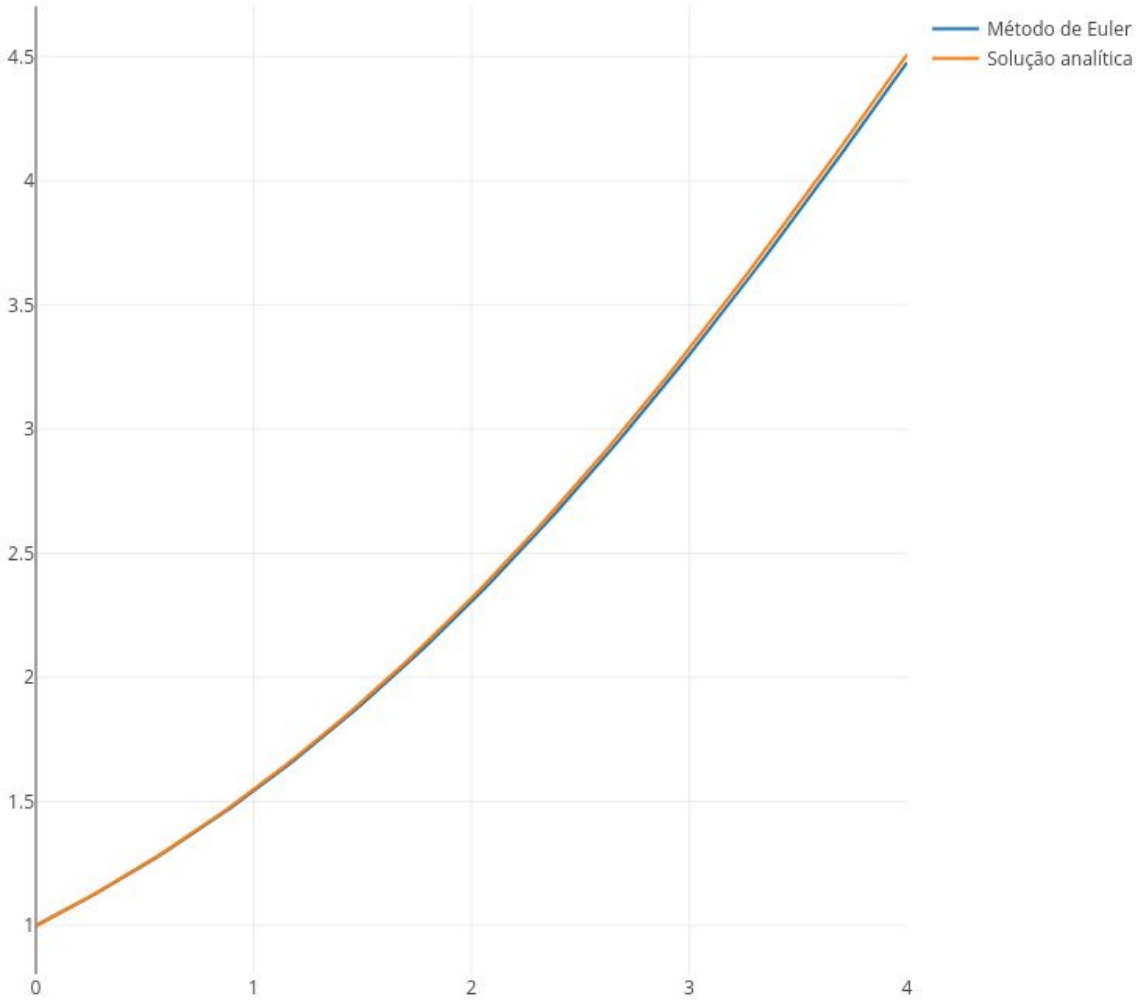


**Gráfico 1: método de Euler com h = 0.05**

**Item b**

Utilizando $h = 0.1$, portanto utilizando pontos mais afastados entre sí, o seguinte gráfico foi obtido:

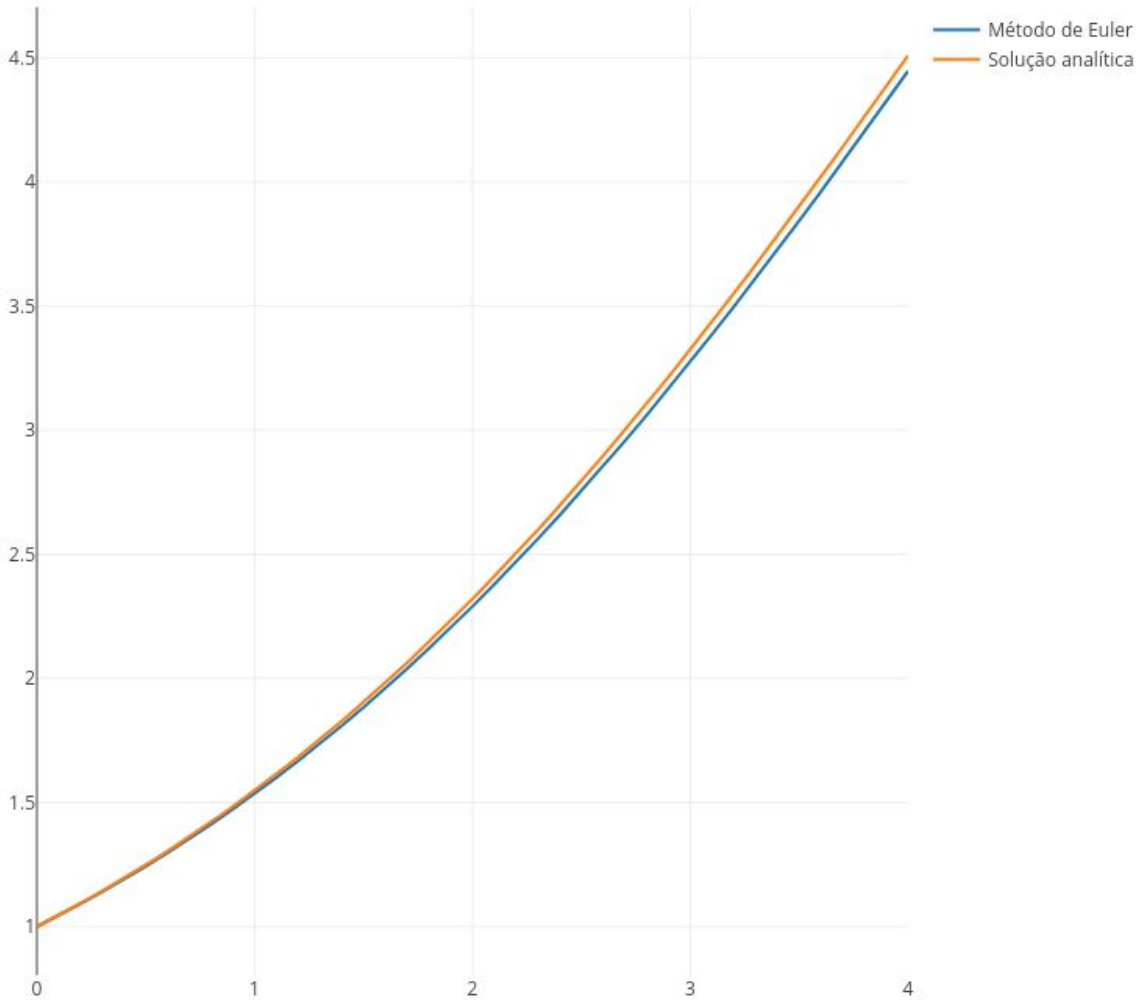


**Gráfico 2: método de Euler com h = 0.1**

Por fim, utilizando pontos ainda mais afastados, com $h = 0.5$, obtivemos o terceiro gráfico:

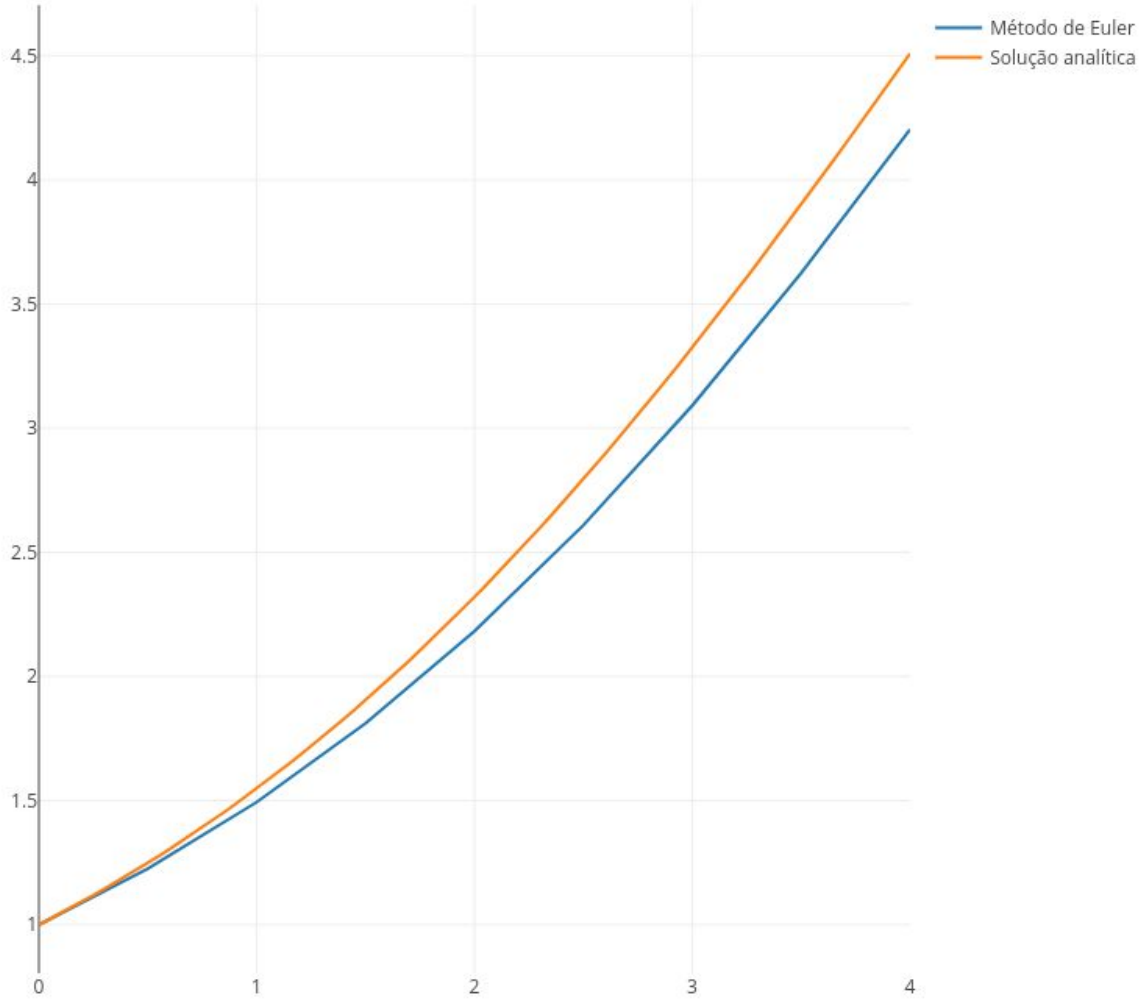


**Gráfico 3: método de Euler com h = 0.5**

É possível observar que, quanto maior o parâmetro $h$, pior é a aproximação gerada pelo método de Euler, já que os pontos são menos e mais afastados um do outro. Variamos também os parâmetros $K$ e $r$ mas não obtivemos diferença notável em termos da qualidade da aproximação do método de Euler.

## Item c

```c
#include<stdio.h>
#include<math.h>
// Usamos essa função para encontrar o valor
double dydt(double t, double y, double r, double K){
  return(r*y*(1-(y/K)));
}
// Encontraremos agora o valor de y para um intervalo t com passo h
double rungeKutta(double t0, double y0, double t, double h, double r, double K){
  // n será o numero de interações necessarias para ir encontrar o valor (no nosso caso n = 200)
  int n = (t - t0) / h;
  double k1, k2, k3, k4, y = y0;
  int i;
  // Aqui fazemos a iteração até o valor n
  for (i = 0; i<n; i++){
    // Aplicamos a formula de Runge-Kutta, com o cálculo da f(t,y) na função que colocamos lá em cima
    printf("%lf\n", y);
    k1 = h*dydt(t0, y, r, K);
    k2 = h*dydt(t0 + 0.5*h, y + 0.5*k1, r, K);
    k3 = h*dydt(t0 + 0.5*h, y + 0.5*k2, r, K);
    k4 = h*dydt(t0 + h, y + k3, r, K);
    // fazemos a mudança do próximo valor de y
    y = y + (1.0/6.0)*(k1 + 2*k2 + 2*k3 + k4);
    // fazemos a mudança para o próximo t, com a soma do nosso passo h
    t0 = t0 + h;
  }
  return y;
}
// Função principal
int main() {
  // Temos nosso intervalo [t0,t] = [0,10], y0 = 1, r = 0.5, K = 10 e h = 0.05
  double t0 = 0, t = 10, y0 = 1, r = 0.5, K = 10, h = 0.05;
  printf("\nO valor de y em t: %f\n", rungeKutta(t0, y0, t, h, r, K));
    return 0;
}
```

**Implementação em C do método de Runge-Kutta de 4a ordem**

Aplicando o método de Runge-Kutta no intervalo [0,10] e utilizando os mesmos parâmetros do item a, obtivemos os seguintes pontos:

| t | y método de Runge-Kutta | y solução analitica |
|---|---|---|
| 0 | 1 | 1 |
| 0.05 | 1.022726 | 1.022726 |
| 0.1 | 1.045909 | 1.045909 |
| 0.15 | 1.069554 | 1.069554 |
| 0.2 | 1.093669 | 1.093669 |
| 0.25 | 1.118259 | 1.118259 |
| 0.3 | 1.143331 | 1.143331 |
| 0.35 | 1.168892 | 1.168892 |
| 0.4 | 1.194946 | 1.194946 |
| 0.45 | 1.221501 | 1.221501 |
| 0.5 | 1.248563 | 1.248563 |
| 0.55 | 1.276137 | 1.276137 |
| 0.6 | 1.304229 | 1.304229 |
| 0.65 | 1.332845 | 1.332845 |
| 0.7 | 1.361991 | 1.361991 |
| 0.75 | 1.391671 | 1.391672 |
| 0.8 | 1.421892 | 1.421893 |
| 0.85 | 1.452659 | 1.452659 |
| 0.9 | 1.483976 | 1.483976 |
| 0.95 | 1.515848 | 1.515848 |
| 1 | 1.548281 | 1.548281 |
| 1.05 | 1.581278 | 1.581278 |
| 1.1 | 1.614844 | 1.614844 |
| 1.15 | 1.648983 | 1.648983 |
| 1.2 | 1.683699 | 1.683699 |
| 1.25 | 1.718995 | 1.718995 |
| 1.3 | 1.754875 | 1.754875 |

| | | |
|---|---|---|
| 1.35 | 1.791342 | 1.791342 |
| 1.4 | 1.828398 | 1.828398 |
| 1.45 | 1.866047 | 1.866047 |
| 1.5 | 1.904291 | 1.904291 |
| 1.55 | 1.943131 | 1.943131 |
| 1.6 | 1.982569 | 1.982569 |
| 1.65 | 2.022607 | 2.022607 |
| 1.7 | 2.063245 | 2.063245 |
| 1.75 | 2.104484 | 2.104484 |
| 1.8 | 2.146325 | 2.146325 |
| 1.85 | 2.188766 | 2.188767 |
| 1.9 | 2.231809 | 2.231809 |
| 1.95 | 2.275452 | 2.275452 |
| 2 | 2.319693 | 2.319693 |
| 2.05 | 2.364531 | 2.364531 |
| 2.1 | 2.409964 | 2.409964 |
| 2.15 | 2.455988 | 2.455989 |
| 2.2 | 2.502603 | 2.502603 |
| 2.25 | 2.549802 | 2.549803 |
| 2.3 | 2.597584 | 2.597584 |
| 2.35 | 2.645943 | 2.645943 |
| 2.4 | 2.694874 | 2.694875 |
| 2.45 | 2.744373 | 2.744373 |
| 2.5 | 2.794433 | 2.794433 |
| 2.55 | 2.845048 | 2.845048 |
| 2.6 | 2.896211 | 2.896212 |
| 2.65 | 2.947916 | 2.947916 |
| 2.7 | 3.000153 | 3.000154 |
| 2.75 | 3.052916 | 3.052916 |
| 2.8 | 3.106194 | 3.106195 |

| | | |
|---|---|---|
| 2.85 | 3.15998 | 3.159981 |
| 2.9 | 3.214263 | 3.214264 |
| 2.95 | 3.269032 | 3.269033 |
| 3 | 3.324278 | 3.324279 |
| 3.05 | 3.379988 | 3.379989 |
| 3.1 | 3.436152 | 3.436152 |
| 3.15 | 3.492756 | 3.492757 |
| 3.2 | 3.549789 | 3.549789 |
| 3.25 | 3.607236 | 3.607237 |
| 3.3 | 3.665085 | 3.665086 |
| 3.35 | 3.723321 | 3.723322 |
| 3.4 | 3.781931 | 3.781931 |
| 3.45 | 3.840898 | 3.840899 |
| 3.5 | 3.900208 | 3.900208 |
| 3.55 | 3.959845 | 3.959846 |
| 3.6 | 4.019793 | 4.019793 |
| 3.65 | 4.080036 | 4.080036 |
| 3.7 | 4.140556 | 4.140556 |
| 3.75 | 4.201337 | 4.201337 |
| 3.8 | 4.262361 | 4.262361 |
| 3.85 | 4.32361 | 4.32361 |
| 3.9 | 4.385067 | 4.385067 |
| 3.95 | 4.446713 | 4.446714 |
| 4 | 4.508531 | 4.508531 |
| 4.05 | 4.5705 | 4.5705 |
| 4.1 | 4.632602 | 4.632602 |
| 4.15 | 4.694818 | 4.694818 |
| 4.2 | 4.75713 | 4.75713 |
| 4.25 | 4.819517 | 4.819517 |
| 4.3 | 4.88196 | 4.88196 |

| | | |
|---|---|---|
| 4.35 | 4.944441 | 4.944441 |
| 4.4 | 5.006938 | 5.006939 |
| 4.45 | 5.069434 | 5.069434 |
| 4.5 | 5.131908 | 5.131908 |
| 4.55 | 5.194341 | 5.194341 |
| 4.6 | 5.256713 | 5.256713 |
| 4.65 | 5.319005 | 5.319005 |
| 4.7 | 5.381198 | 5.381197 |
| 4.75 | 5.443272 | 5.443272 |
| 4.8 | 5.505209 | 5.505209 |
| 4.85 | 5.566989 | 5.566989 |
| 4.9 | 5.628595 | 5.628595 |
| 4.95 | 5.690008 | 5.690008 |
| 5 | 5.751209 | 5.751209 |
| 5.05 | 5.812181 | 5.81218 |
| 5.1 | 5.872905 | 5.872904 |
| 5.15 | 5.933365 | 5.933365 |
| 5.2 | 5.993544 | 5.993543 |
| 5.25 | 6.053424 | 6.053423 |
| 5.3 | 6.112989 | 6.112989 |
| 5.35 | 6.172225 | 6.172224 |
| 5.4 | 6.231113 | 6.231113 |
| 5.45 | 6.289641 | 6.289641 |
| 5.5 | 6.347793 | 6.347793 |
| 5.55 | 6.405554 | 6.405554 |
| 5.6 | 6.462911 | 6.46291 |
| 5.65 | 6.519849 | 6.519849 |
| 5.7 | 6.576357 | 6.576356 |
| 5.75 | 6.632421 | 6.63242 |
| 5.8 | 6.688029 | 6.688028 |

| | | |
|---|---|---|
| 5.85 | 6.74317 | 6.743169 |
| 5.9 | 6.797832 | 6.797831 |
| 5.95 | 6.852005 | 6.852005 |
| 6 | 6.905679 | 6.905679 |
| 6.05 | 6.958844 | 6.958843 |
| 6.1 | 7.01149 | 7.01149 |
| 6.15 | 7.06361 | 7.06361 |
| 6.2 | 7.115195 | 7.115195 |
| 6.25 | 7.166237 | 7.166237 |
| 6.3 | 7.21673 | 7.21673 |
| 6.35 | 7.266665 | 7.266666 |
| 6.4 | 7.316039 | 7.316039 |
| 6.45 | 7.364843 | 7.364844 |
| 6.5 | 7.413074 | 7.413075 |
| 6.55 | 7.460727 | 7.460728 |
| 6.6 | 7.507797 | 7.507798 |
| 6.65 | 7.554281 | 7.554281 |
| 6.7 | 7.600174 | 7.600175 |
| 6.75 | 7.645475 | 7.645476 |
| 6.8 | 7.690181 | 7.690181 |
| 6.85 | 7.73429 | 7.73429 |
| 6.9 | 7.777799 | 7.777799 |
| 6.95 | 7.820708 | 7.820709 |
| 7 | 7.863017 | 7.863017 |
| 7.05 | 7.904724 | 7.904724 |
| 7.1 | 7.945829 | 7.94583 |
| 7.15 | 7.986334 | 7.986335 |
| 7.2 | 8.026238 | 8.026239 |
| 7.25 | 8.065544 | 8.065545 |
| 7.3 | 8.104252 | 8.104252 |

| | | |
|---|---|---|
| 7.35 | 8.142364 | 8.142363 |
| 7.4 | 8.179881 | 8.179881 |
| 7.45 | 8.216806 | 8.216806 |
| 7.5 | 8.253142 | 8.253142 |
| 7.55 | 8.288893 | 8.288892 |
| 7.6 | 8.324059 | 8.324059 |
| 7.65 | 8.358646 | 8.358647 |
| 7.7 | 8.392658 | 8.392658 |
| 7.75 | 8.426098 | 8.426097 |
| 7.8 | 8.458969 | 8.458969 |
| 7.85 | 8.491277 | 8.491277 |
| 7.9 | 8.523026 | 8.523025 |
| 7.95 | 8.55422 | 8.55422 |
| 8 | 8.584866 | 8.584864 |
| 8.05 | 8.614966 | 8.614965 |
| 8.1 | 8.644527 | 8.644526 |
| 8.15 | 8.673555 | 8.673554 |
| 8.2 | 8.702055 | 8.702053 |
| 8.25 | 8.730032 | 8.73003 |
| 8.3 | 8.757492 | 8.75749 |
| 8.35 | 8.784441 | 8.784438 |
| 8.4 | 8.810884 | 8.810882 |
| 8.45 | 8.836829 | 8.836826 |
| 8.5 | 8.862281 | 8.862278 |
| 8.55 | 8.887245 | 8.887242 |
| 8.6 | 8.911729 | 8.911726 |
| 8.65 | 8.935739 | 8.935736 |
| 8.7 | 8.959281 | 8.959278 |
| 8.75 | 8.982362 | 8.982359 |
| 8.8 | 9.004988 | 9.004985 |

| | | |
|---|---|---|
| 8.85 | 9.027164 | 9.027162 |
| 8.9 | 9.0489 | 9.048897 |
| 8.95 | 9.070199 | 9.070196 |
| 9 | 9.091069 | 9.091066 |
| 9.05 | 9.111517 | 9.111514 |
| 9.1 | 9.131549 | 9.131546 |
| 9.15 | 9.151171 | 9.151168 |
| 9.2 | 9.17039 | 9.170387 |
| 9.25 | 9.189213 | 9.189209 |
| 9.3 | 9.207645 | 9.207642 |
| 9.35 | 9.225694 | 9.22569 |
| 9.4 | 9.243365 | 9.243362 |
| 9.45 | 9.260666 | 9.260662 |
| 9.5 | 9.277601 | 9.277597 |
| 9.55 | 9.294178 | 9.294175 |
| 9.6 | 9.310403 | 9.3104 |
| 9.65 | 9.326282 | 9.326279 |
| 9.7 | 9.341821 | 9.341818 |
| 9.75 | 9.357026 | 9.357024 |
| 9.8 | 9.371904 | 9.371902 |
| 9.85 | 9.38646 | 9.386458 |
| 9.9 | 9.400701 | 9.400699 |
| 9.95 | 9.414631 | 9.41463 |
| 10 | 9.428257 | 9.428256 |

**Tabela 2: método de Runge-Kutta de 4a ordem com h = 0.05**

Com esses pontos, obtivemos o seguinte gráfico:

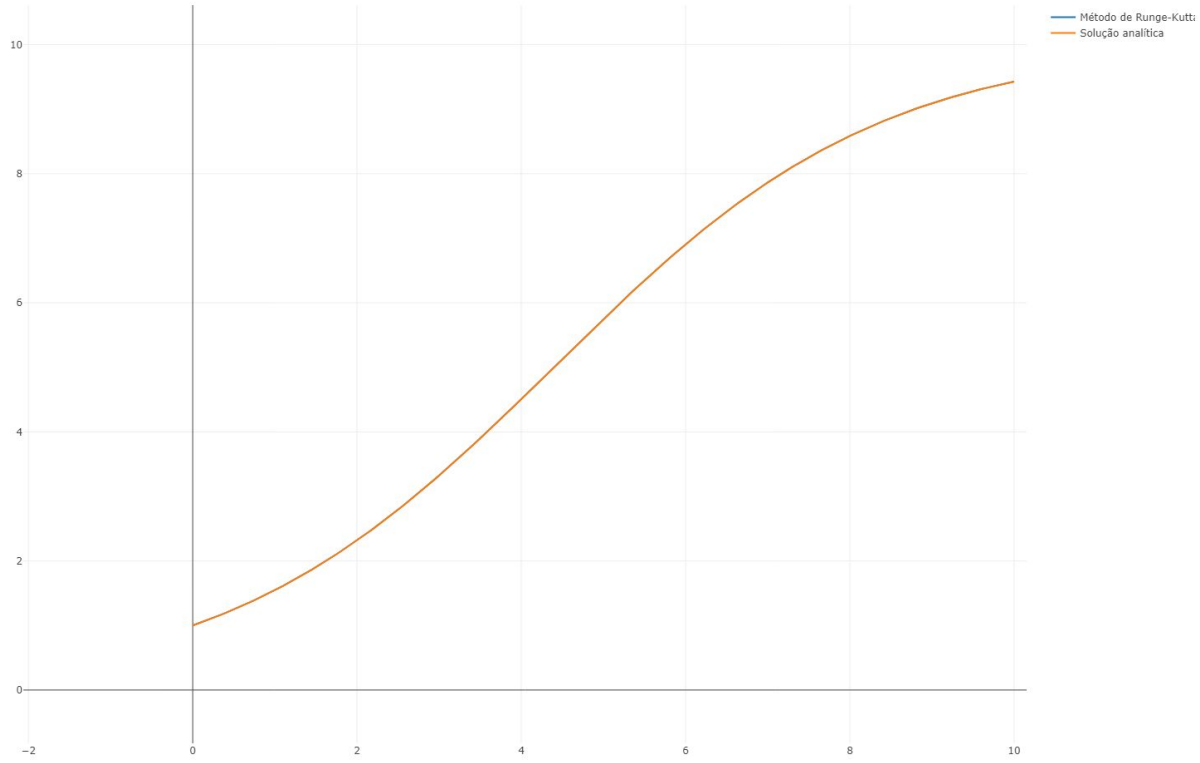


**Gráfico 4: método de Runge-Kutta de 4a ordem com h = 0.05**

Pelo gráfico e pela tabela, podemos observar que o método de Runge-Kutta chega em resultados muito próximos ao da solução analítica, sendo que com a escala do gráfico acima fica impossível distinguir as duas linhas.

**Item d**

Fazendo uma comparação entre os gráficos dos dois métodos temos que o método de Runge-Kutta de quarta ordem se aproxima mais do valor da solução analítica do que o método de Euler. Observando a tabela 2 é possível ver que, mesmo no intervalo [0,4] (onde também foi executado o método de Euler), a aproximação aparece correta na maioria dos pontos até a quinta casa decimal, enquanto o método de Euler, pela tabela 1, gerou aproximações nesse intervalo que estão corretas apenas até a primeira casa decimal. Isso se deve ao fato do método de Euler ser um método de Runge-Kutta de ordem menor que 4, portanto menos preciso que o Runge-Kutta de ordem 4, já que há menos precisão no cálculo dos valores.

## Conclusão

Entre os dois métodos estudados, que são utilizados para se obter uma aproximação da solução de problemas de valor inicial (PVI) relacionados a equações diferenciais ordinárias (EDOs), o método de Runge-Kutta de quarta ordem foi o que mais se aproximou do resultado analítico. O método de Euler, por ser o método Runge-Kutta mais simples, tem o *trade-off* de não gerar aproximações tão boas quanto seus primos de ordem mais alta.

**Bibliografia**

*RUGGIERO, M.; LOPES, V. Cálculo Numérico, Aspectos Teóricos e Computacionais: 2. ed.*