Instituto Politécnico do Rio de Janeiro

Universidade Estadual do Rio de Janeiro

Curso de Graduação em Engenharia da Computação

Trabalho 1 - Métodos Numéricos para Equações Diferenciais

Professor: Helio Pedro Amaral Souto

Aluno: Victor Luis Teixeira Reis

Nova Friburgo

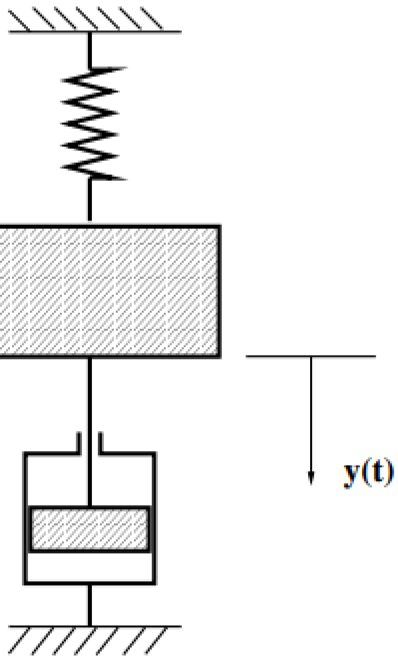
2024

1. Introdução

O trabalho a seguir consiste no uso de métodos numéricos e analíticos para obter a solução de um sistema massa-mola com amortecimento. Para isso, será utilizado o método de Runge-Kutta de 4° ordem do tipo ⅜ para resolver numericamente o problema com diferentes valores de coeficientes de amortecimento e constantes elásticas, sempre comparando os resultados com a solução analítica proveniente do cálculo convencional.

1.1 Apresentação do problema

O problema que será solucionado envolve a modelagem de um sistema massa-mola com amortecimento usando equações diferenciais de 2° ordem. Em tais sistemas, uma mola distendida ligada a um bloco de massa, irá oscilar com uma amplitude decrescente, devido à dissipação da energia por meio do amortecimento. A figura abaixo ilustra esse tipo de problema:



1.2 Apresentação da solução analítica

Usando a 2° Lei de Newton, é possível deduzir a posição do bloco em função do tempo por meio de uma equação diferencial do 2° grau da forma:

$$\frac{d^2y}{dt^2} + A\frac{dy}{dt} + By = 0, com A = \frac{c}{m}, B = \frac{k}{m}$$

Onde m é a massa do bloco, c é o coeficiente de amortecimento e k é a constante elástica da mola. Tal equação, pode ser resolvida analiticamente usando técnicas de solução de equações diferenciais, resultando na seguinte expressão para y(t):

$$y(t) = y_0 exp(Pt)[cos(Qt) - (\frac{P}{Q})sen(Qt)]$$

Para determinar a velocidade do bloco, basta derivar a expressão acima:

$$v(t) = -y_0 exp(Pt)(\frac{P^2 + Q^2}{Q})sen(Qt)$$

Tais soluções são únicas e verdadeiras se considerarmos que o bloco de massa terá uma velocidade inicial nula, com uma determinada posição inicial arbitrária. Para fins de padronização, será considerado neste trabalho a posição inicial y_0 de 0.5 e velocidade inicial nula em todas as análises. Além disso, as expressões para P e Q são:

$$P = -\frac{A}{2}, Q = \sqrt{B - (\frac{A^2}{4})}$$

Vale ressaltar que dependendo do valor de B, a solução será complexa. Dessa forma, para que a solução não seja desse tipo, as seguintes condições devem ser respeitadas:

$$B > \frac{A^2}{4}$$

Por consequência:

$$k > \frac{c^2}{4m}$$

Se a solução não for complexa, ela terá um decaimento exponencial oscilatório, caso contrário, seu decaimento será exponencial não oscilatório.

1.3 Apresentação do método numérico

O método numérico utilizado para resolver o problema é o método Runge-Kutta de 4° ordem do tipo ⅜ . Como a equação a ser resolvida é de 2° ordem, será necessário a decompor em um sistema de duas equações de 1° ordem, da seguinte forma:

$$\frac{dy}{dt} = f_1(t, y, v) = v$$

$$\frac{dv}{dt} = f_2(t, y, v) = -(Av + By)$$

Tais equações, se substituídas uma na outra, resultam na expressão original de 2° ordem:

$$\frac{d^2y}{dt^2} + A\frac{dy}{dt} + By = 0, com A = \frac{c}{m}, B = \frac{k}{m}$$

Portanto, o problema agora se resume a utilizar o método de Runge-Kutta para resolver essas duas equações ao mesmo tempo.

O método em questão tem a seguinte fórmula geral, que será utilizada ao longo das iterações para estimar os valores em cada momento de tempo:

$$y_{i+1} = y_i + \left[\frac{1}{8}(k_1 + 3k_2 + 3k_3 + k_4)\right]\Delta t$$

O termo y_{i+1} será a nova estimativa a ser descoberta, enquanto y_i é a estimativa atual e Δt é o incremento de tempo a cada loop. Os valores de k são obtidos com base nas seguintes expressões.

$$k_1 = f(t_i, y_i)$$

$$k_2 = f\left(t_i + \frac{1}{3}\Delta t,\ y_i + \frac{1}{3}k_1\Delta t\right)$$

$$k_3 = f\left(t_i + \frac{2}{3}\Delta t,\ y_i - \frac{1}{3}k_1\Delta t + k_2\Delta t\right)$$

$$k_4 = f\left(t_i + \Delta t,\ y_i + k_1\Delta t - k_2\Delta t + k_3\Delta t\right)$$

Como estamos resolvendo um sistema de duas equações, essas fórmulas terão que ser adaptadas, pois será necessário duas fórmulas gerais, uma para a posição e outra para a velocidade, além de um total de 8 expressões para k, para abranger essas duas fórmulas:

$$y_{i+1} = y_i + \left[\frac{1}{8}(k_1^1 + 3k_2^1 + 3k_3^1 + k_4^1)\right]\Delta t$$

$$v_{i+1} = v_i + \left[\frac{1}{8}(k_1^2 + 3k_2^2 + 3k_3^2 + k_4^2)\right]\Delta t$$

$$k_1^1 = f_1(t_0, y_0, v_0)$$
$$k_1^2 = f_2(t_0, y_0, v_0)$$
$$k_2^1 = f_1\left(t_1 + \frac{\Delta t}{3}, y_1 + \frac{\Delta t}{3}k_1^1, v_1 + \frac{\Delta t}{3}k_1^2\right)$$
$$k_2^2 = f_2\left(t_1 + \frac{\Delta t}{3}, y_1 + \frac{\Delta t}{3}k_1^1, v_1 + \frac{\Delta t}{3}k_1^2\right)$$
$$k_3^1 = f_1\left(t_2 + \frac{2\Delta t}{3}, y_2 - \frac{\Delta t}{3}k_1^1 + k_2^1\Delta t, v_2 - \frac{\Delta t}{3}k_1^2 + k_2^2\Delta t\right)$$
$$k_3^2 = f_2\left(t_2 + \frac{2\Delta t}{3}, y_2 - \frac{\Delta t}{3}k_1^1 + k_2^1\Delta t, v_2 - \frac{\Delta t}{3}k_1^2 + k_2^2\Delta t\right)$$
$$k_4^1 = f_1\left(t_3 + \Delta t, y_3 + k_1^1\Delta t - k_2^1\Delta t + k_3^1\Delta t, v_3 + k_1^2\Delta t - k_2^2\Delta t + k_3^2\Delta t\right)$$
$$k_4^2 = f_2\left(t_3 + \Delta t, y_3 + k_1^1\Delta t - k_2^1\Delta t + k_3^1\Delta t, v_3 + k_1^2\Delta t - k_2^2\Delta t + k_3^2\Delta t\right)$$

2. Resultados

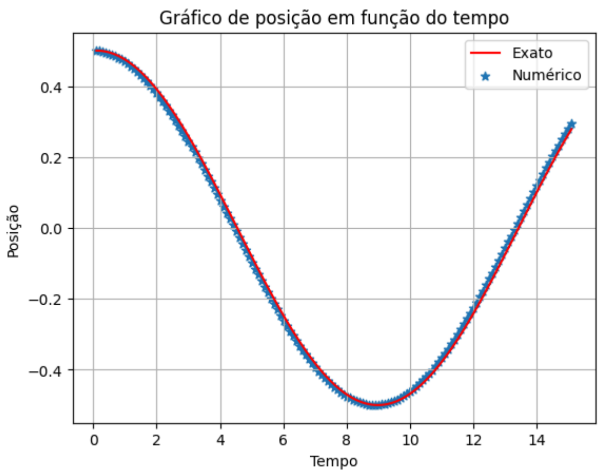
Para apresentar os resultados, a simulação será dividida em dois casos. No primeiro caso, vamos considerar $c = 0$, $k = \{0.25,\ 0.5,\ 0.75,\ 1.0\}$ e $t = 15$. No segundo caso, teremos $c = 4$, $k = \{1.0,\ 10.0,\ 50.0,\ 150.0\}$ e $t = 10$. Em ambos os casos, a massa m do bloco será igual a 2. Como já comentado anteriormente, para padronizar, será considerado a velocidade inicial do bloco como sendo nulo e a posição inicial como sendo 0.5. Os incrementos de tempo serão de 0.1.

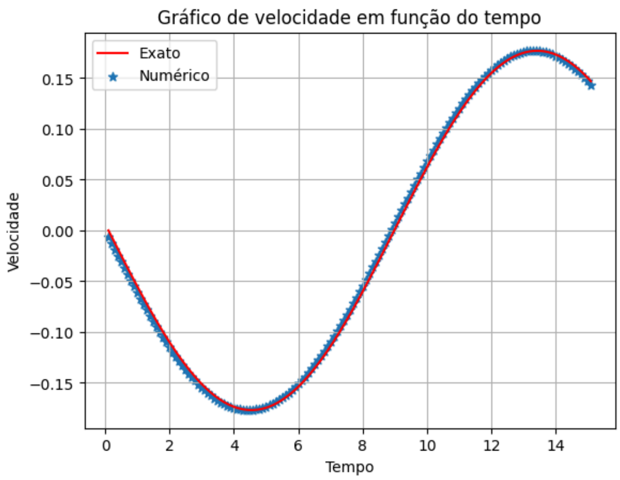
2.1 Primeiro caso

- **c = 0; k = 0.25; t = 15**

	Tempo	y_exato	y_numérico	v_exato	v_numérico
0	0.1	0.500000	0.499688	-0.000000	-0.006249
1	0.2	0.499688	0.498751	-0.006249	-0.012490
2	0.3	0.498751	0.497190	-0.012490	-0.018715
3	0.4	0.497190	0.495008	-0.018715	-0.024917
4	0.5	0.495008	0.492208	-0.024917	-0.031087
...	...	...	...	...	...
145	14.6	0.201199	0.217253	0.161833	0.159217
146	14.7	0.217253	0.233036	0.159217	0.156403
147	14.8	0.233036	0.248527	0.156403	0.153393
148	14.9	0.248527	0.263708	0.153393	0.150191
149	15.0	0.263708	0.278559	0.150191	0.146801

150 rows × 5 columns

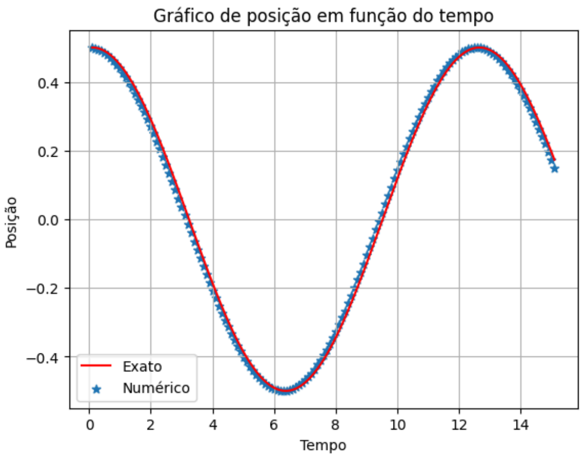


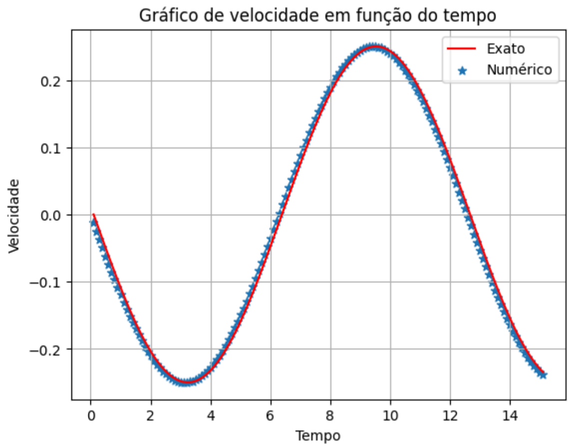


- **c = 0; k = 0.5; t = 15**

	Tempo	y_exato	y_numérico	v_exato	v_numérico
0	0.1	0.500000	0.499375	-0.000000	-0.012495
1	0.2	0.499375	0.497502	-0.012495	-0.024958
2	0.3	0.497502	0.494386	-0.024958	-0.037360
3	0.4	0.494386	0.490033	-0.037360	-0.049667
4	0.5	0.490033	0.484456	-0.049667	-0.061851
...	...	...	...	...	...
145	14.6	0.283962	0.263039	-0.205770	-0.212609
146	14.7	0.263039	0.241458	-0.212609	-0.218917
147	14.8	0.241458	0.219274	-0.218917	-0.224677
148	14.9	0.219274	0.196541	-0.224677	-0.229876
149	15.0	0.196541	0.173318	-0.229876	-0.234500

150 rows × 5 columns

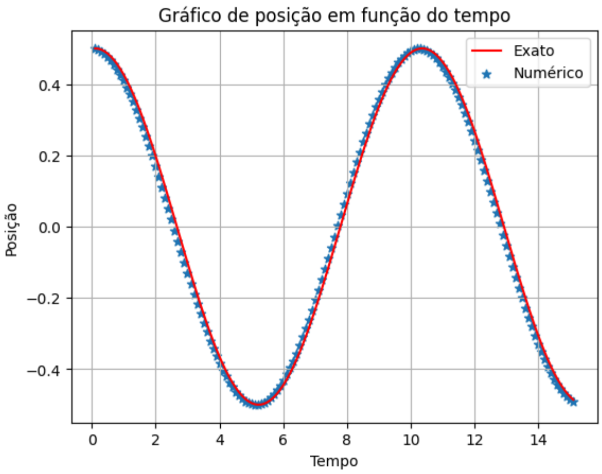


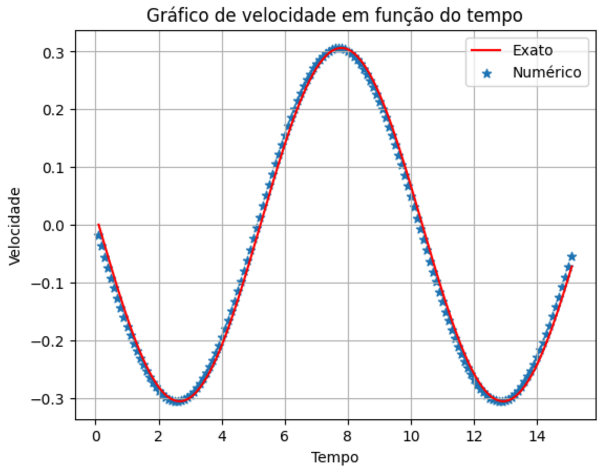


- **c = 0; k = 0.75; t = 15**

	Tempo	y_exato	y_numérico	v_exato	v_numérico
0	0.1	0.500000	0.499063	-0.000000	-0.018738
1	0.2	0.499063	0.496255	-0.018738	-0.037406
2	0.3	0.496255	0.491586	-0.037406	-0.055934
3	0.4	0.491586	0.485075	-0.055934	-0.074252
4	0.5	0.485075	0.476745	-0.074252	-0.092292
...	...	...	...	...	...
145	14.6	-0.427466	-0.442537	-0.158831	-0.142514
146	14.7	-0.442538	-0.455950	-0.142514	-0.125662
147	14.8	-0.455951	-0.467654	-0.125662	-0.108339
148	14.9	-0.467654	-0.477605	-0.108339	-0.090610
149	15.0	-0.477605	-0.485765	-0.090610	-0.072541

150 rows × 5 columns

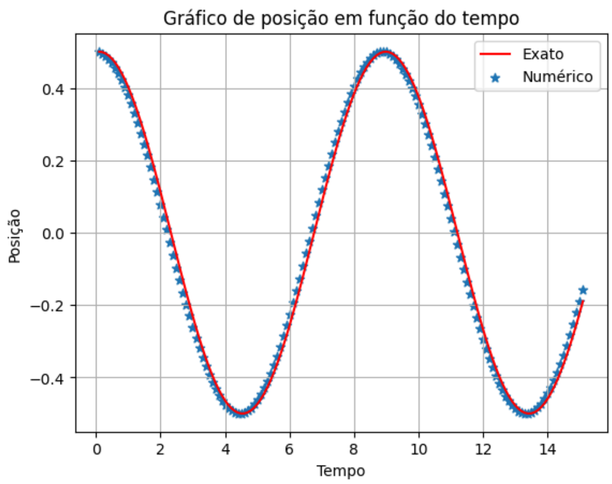


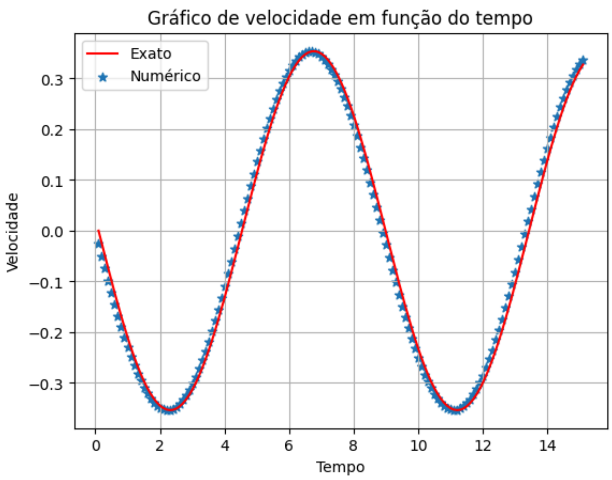


- **c = 0; k = 1.0; t = 15**

	Tempo	y_exato	y_numérico	v_exato	v_numérico
0	0.1	0.500000	0.498751	-0.000000	-0.024979
1	0.2	0.498751	0.495008	-0.024979	-0.049833
2	0.3	0.495008	0.488792	-0.049833	-0.074439
3	0.4	0.488792	0.480133	-0.074439	-0.098672
4	0.5	0.480133	0.469074	-0.098672	-0.122412
...	...	...	...	...	...
145	14.6	-0.338076	-0.311205	0.260485	0.276723
146	14.7	-0.311204	-0.282778	0.276724	0.291579
147	14.8	-0.282777	-0.252937	0.291580	0.304978
148	14.9	-0.252937	-0.221833	0.304978	0.316852
149	15.0	-0.221832	-0.189620	0.316852	0.327142

150 rows × 5 columns





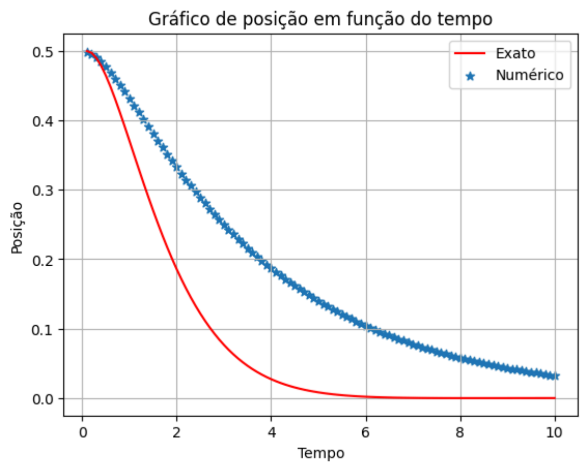
Em todas as simulações deste primeiro caso, tanto a posição numérica quanto a velocidade ficaram relativamente próximas das suas correspondentes exatas. Além disso, como o coeficiente de amortecimento foi nulo, temos gráficos trigonométricos típicos de sistemas massa-mola sem amortecimento, com nenhuma perda de amplitude das ondas. Aumentar o valor da constante elástica fez com que as molas oscilassem mais rapidamente, com maior frequência.

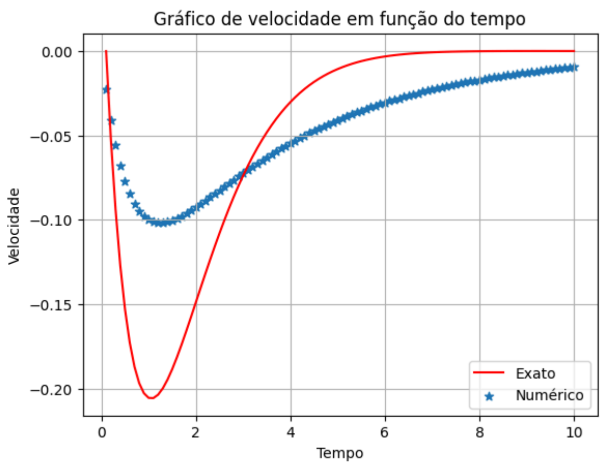
2.2 Segundo caso

- **c = 4; k = 1.0; t = 10**

	Tempo	y_exato	y_numérico	v_exato	v_numérico
0	0.1	0.500000	0.498830	-0.000000	-0.022640
1	0.2	0.497318	0.495612	-0.051855	-0.041073
2	0.3	0.489960	0.490732	-0.093771	-0.055979
3	0.4	0.478850	0.484514	-0.127117	-0.067929
4	0.5	0.464782	0.477229	-0.153098	-0.077405
...	...	...	...	...	...
95	9.6	-0.000079	0.036272	0.000053	-0.010624
96	9.7	-0.000074	0.035225	0.000051	-0.010317
97	9.8	-0.000069	0.034209	0.000050	-0.010019
98	9.9	-0.000064	0.033221	0.000047	-0.009730
99	10.0	-0.000060	0.032262	0.000045	-0.009449

100 rows × 5 columns





Com esses valores temos uma solução complexa, pois a condição para ter soluções reais não é satisfeita:

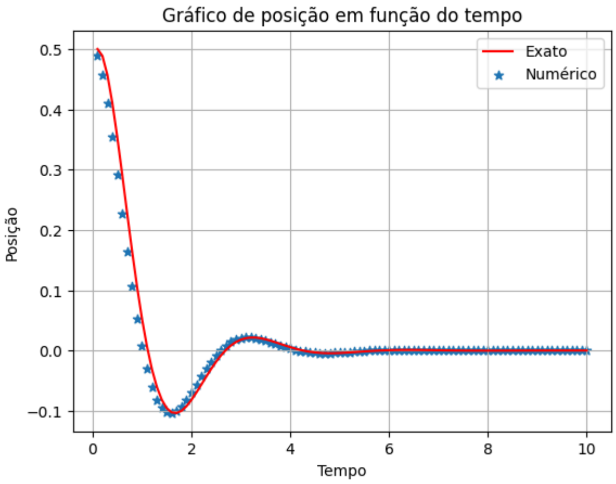
$$k > \frac{c^2}{4m} \Rightarrow 1 \not> \frac{4^2}{4 * 2} \Rightarrow 1 \not> 2$$

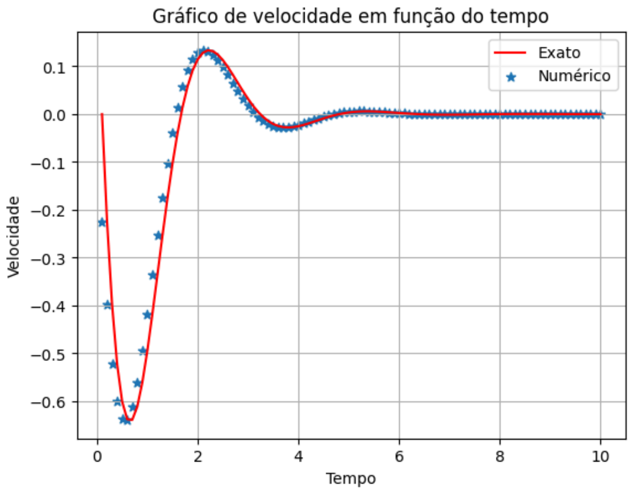
O gráfico apresenta um decaimento exponencial não oscilatório, porém a acurácia do método nesse caso é discutível, já que tem um erro considerável entre o valor real e o estimado.

- **c = 4; k = 10; t = 10**

	Tempo	y_exato	y_numérico	v_exato	v_numérico
0	0.1	0.500000	0.488344	-0.000000	-0.224708
1	0.2	0.488341	0.456762	-0.224704	-0.398545
2	0.3	0.456758	0.410291	-0.398536	-0.522885
3	0.4	0.410286	0.353727	-0.522872	-0.601089
4	0.5	0.353723	0.291453	-0.601073	-0.637992
...	...	...	...	...	...
95	9.6	0.000040	0.000038	-0.000014	-0.000029
96	9.7	0.000038	0.000034	-0.000029	-0.000040
97	9.8	0.000034	0.000030	-0.000040	-0.000047
98	9.9	0.000030	0.000025	-0.000047	-0.000051
99	10.0	0.000025	0.000020	-0.000051	-0.000052

100 rows × 5 columns

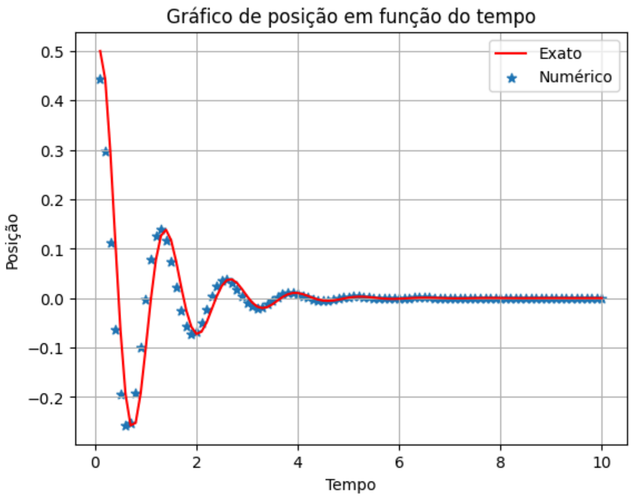


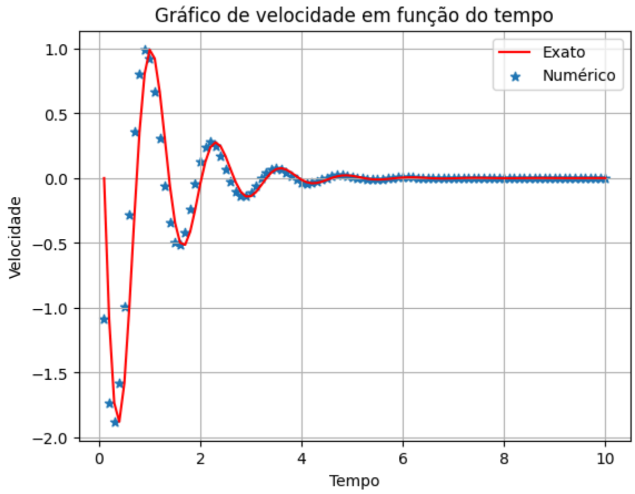


- **c = 4; k = 50; t = 10**

	Tempo	y_exato	y_numérico	v_exato	v_numérico
0	0.1	0.500000	0.442760	-0.000000	-1.086042
1	0.2	0.442659	0.297715	-1.086345	-1.734707
2	0.3	0.297483	0.112915	-1.734700	-1.881346
3	0.4	0.112609	-0.063469	-1.880584	-1.584318
4	0.5	-0.063742	-0.193854	-1.582706	-0.989785
...	...	...	...	...	...
95	9.6	-0.000027	-0.000033	-0.000105	-0.000020
96	9.7	-0.000033	-0.000031	-0.000016	0.000058
97	9.8	-0.000031	-0.000023	0.000060	0.000109
98	9.9	-0.000022	-0.000011	0.000110	0.000127
99	10.0	-0.000010	0.000002	0.000126	0.000113

100 rows × 5 columns

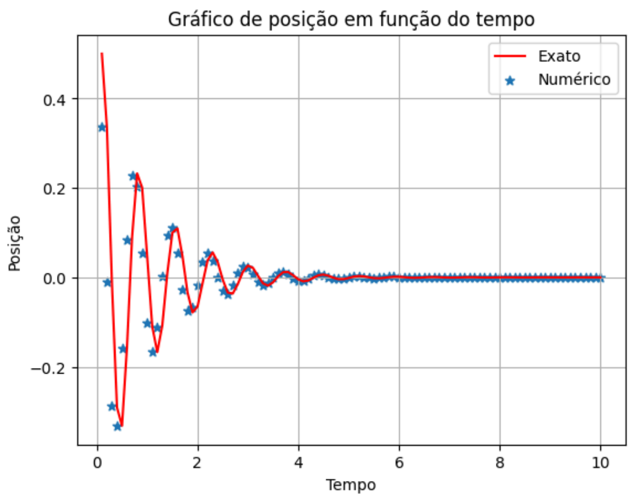


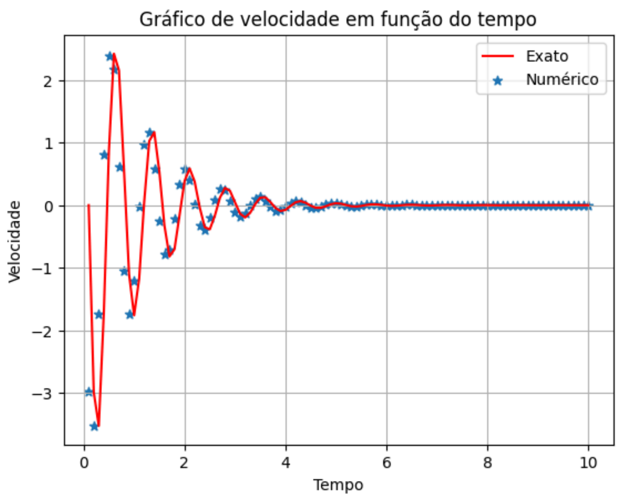


- **c = 4; k = 150; t = 10**

	Tempo	y_exato	y_numérico	v_exato	v_numérico
0	0.1	0.500000	0.336094	-0.000000	-2.976875
1	0.2	0.334960	-0.010396	-2.989867	-3.529408
2	0.3	-0.013985	-0.287164	-3.529181	-1.750175
3	0.4	-0.290750	-0.331963	-1.717882	0.811129
4	0.5	-0.331745	-0.158751	0.861695	2.392875
...	...	...	...	...	...
95	9.6	0.000038	0.000030	-0.000013	-0.000130
96	9.7	0.000024	0.000010	-0.000231	-0.000245
97	9.8	-0.000002	-0.000013	-0.000262	-0.000185
98	9.9	-0.000022	-0.000023	-0.000120	-0.000018
99	10.0	-0.000025	-0.000017	0.000073	0.000129

100 rows × 5 columns





Para as simulações do segundo caso, exceto a primeira, o método de Runge-Kutta 4° ordem ⅜ novamente encontrou soluções bem próximas dos resultados reais das funções. Como nesse caso o coeficiente de amortecimento não era nulo, o comportamento dos gráficos apresentam decaimentos da posição e da velocidade à medida que o tempo avança. Assim como no primeiro caso, quanto maior o valor da constante elástica k, maior a frequência das oscilações.

3. Conclusão

Em resumo, os resultados obtidos através do emprego do método numérico de Runge-Kutta de 4ª ordem ⅜ demonstraram uma boa confiabilidade na predição do comportamento da solução do problema em questão. A utilização de incrementos de tempo de 0.1 durante os testes revelou resultados satisfatórios, porém, a implementação de incrementos menores poderia resultar em uma precisão ainda mais elevada. Este método se destaca pela sua eficácia em oferecer uma representação precisa do fenômeno estudado, oferecendo uma base sólida para análises e previsões futuras.

4. Código

```
"""
  Trabalho 1 - Métodos Númericos para Equações Diferenciais
  Aluno: Victor Luis Teixeira Reis - 202110049511

  Problema:
    dy/dt = dy_1/dt = f_1(t, y, v) = v
    dv/dt = dy_2/dt =  f_2(t, y, v) = -(A*v + B*y)
    Método: yi+1 = yi + (1/8)*(k1 + 3*k2 + 3*k3 + k4) * Δt
"""

from math import *
import pandas as pd
import matplotlib.pyplot as plt

def f_1(t, y, v):
  """Primeira função do sistema"""
  return v

def f_2(t, y, v):
  """Segunda função do sistema"""
  return -(A*v + B*y)

def analytic_position(y_0, t):
  """Expressão analítica para a posição do bloco"""
  return y_0*exp(P*t)*(cos(Q*t) - (P/Q)*sin(Q*t))

def analytic_velocity(y_0, t):
```

```
  """Expressão analítica para a velocidade do bloco"""
  return -y_0*exp(P*t)*((P**2 + Q**2)/Q)*sin(Q*t)

def calculate_k_matrix(k, actual_value, delta_t):
  """Função para calcular todos os k's do problema"""
  [t_i, y_i, v_i] =  actual_value
  third_delta_t = (1/3) * delta_t
  two_third_delta_t = (2/3) * delta_t

  k[0][0] = f_1(t_i, y_i, v_i)
  k[1][0] = f_2(t_i, y_i, v_i)
  k[0][1] = f_1(t_i + third_delta_t, y_i + k[0][0]*third_delta_t, v_i +
k[1][0]*third_delta_t)
  k[1][1] = f_2(t_i + third_delta_t, y_i + k[0][0]*third_delta_t, v_i +
k[1][0]*third_delta_t)
  k[0][2] = f_1(t_i + two_third_delta_t, y_i - k[0][0]*third_delta_t +
k[0][1]*delta_t, v_i - k[1][0]*third_delta_t + k[1][1]*delta_t)
  k[1][2] = f_2(t_i + two_third_delta_t, y_i - k[0][0]*third_delta_t +
k[0][1]*delta_t, v_i - k[1][0]*third_delta_t + k[1][1]*delta_t)
  k[0][3] = f_1(t_i + delta_t, y_i + k[0][0]*delta_t - k[0][1]*delta_t +
k[0][2]*delta_t, v_i + k[1][0]*delta_t - k[1][1]*delta_t +
k[1][2]*delta_t)
  k[1][3] = f_2(t_i + delta_t, y_i + k[0][0]*delta_t - k[0][1]*delta_t +
k[0][2]*delta_t, v_i + k[1][0]*delta_t - k[1][1]*delta_t +
k[1][2]*delta_t)


def runge_kutta(initial_value, delta_t, t):
  """Rotina principal do método Runge-Kutta 4° ordem 3/8"""
  actual_value = initial_value
  y_0 = initial_value[1]

  k = [
    [0.0, 0.0, 0.0, 0.0], # [k_11, k_12, k_13, k_14]
    [0.0, 0.0, 0.0, 0.0], # [k_21, k_22, k_23, k_24]
  ]

  while actual_value[0] < t:
    calculate_k_matrix(k, actual_value, delta_t)

    actual_value[1] = actual_value[1] + ((1/8)*(k[0][0] + 3*k[0][1] +
3*k[0][2] + k[0][3]) * delta_t)
    actual_value[2] = actual_value[2] + ((1/8)*(k[1][0] + 3*k[1][1] +
3*k[1][2] + k[1][3]) * delta_t)

    exact_position = analytic_position(y_0, actual_value[0])
```

```
    exact_velocity = analytic_velocity(y_0, actual_value[0])

    actual_value[0] = round(actual_value[0] + delta_t, 1)

    add_table_data(actual_value[0], actual_value[1], exact_position,
actual_value[2], exact_velocity)

  return [actual_value[1], actual_value[2]]

def add_table_data (time, yn, ye, vn, ve):
  """Função que adiciona os valores em uma tabela"""
  table_data['Tempo'].append(time)
  table_data['y_exato'].append(ye)
  table_data['y_numérico'].append(yn)
  table_data['v_exato'].append(ve)
  table_data['v_numérico'].append(vn)

def plot_position_graph (t, y_exact, y_numeric):
  """Função que plota os valores exatos e numéricos da posição"""
  plt.plot(t, y_exact, label="Exato", color="red")
  plt.scatter(t, y_numeric, label="Numérico", marker='*')
  plt.xlabel('Tempo')
  plt.ylabel('Posição')
  plt.title('Gráfico de posição em função do tempo')
  plt.grid(True)
  plt.legend()
  plt.show()


def plot_velocity_graph (t, v_exact, v_numeric):
  """Função que plota os valores exatos e numéricos da velocidade"""
  plt.plot(t, v_exact, label="Exato", color="red")
  plt.scatter(t, v_numeric, label="Numérico", marker='*')
  plt.xlabel('Tempo')
  plt.ylabel('Velocidade')
  plt.title('Gráfico de velocidade em função do tempo')
  plt.grid(True)
  plt.legend()
  plt.show()


# Valores e condições iniciais
initial_value = [0.0, 0.5, 0.0] # [t_0, y_0, v_0]
delta_t = 0.1
t = 10.0
c = 4
```

```
m = 2.0
k = 1.0

A = c/m
B = k/m

P = - (A/2)
Q = sqrt(B - ((A**0.5)/4))

table_data = {
  'Tempo': [],
  'y_exato': [],
  'y_numérico': [],
  'v_exato': [],
  'v_numérico': [],
}

# Executando o método e plotando os resultados
print(f'===========Runge Kutta System Method============')

y, v = runge_kutta(initial_value, delta_t, t)

df = pd.DataFrame(table_data)
display(df)

plot_position_graph(table_data["Tempo"], table_data["y_exato"],
table_data["y_numérico"])
plot_velocity_graph(table_data["Tempo"], table_data["v_exato"],
table_data["v_numérico"])
```

5. Referências

1. **Notas de aula de Métodos Numéricos de Equações Diferenciais do professor Helio Pedro Amaral Souto.** Disponível em: <https://ead.iprj.uerj.br/moodle/pluginfile.php/8304/mod_resource/content/21/Notas_de_Aula.pdf>. Acesso em: 05 abr. 2024
2. **Primeiro trabalho computacional de Métodos Numéricos de Equações Diferenciais do professor Helio Pedro Amaral Souto.** Disponível em: <https://ead.iprj.uerj.br/moodle/mod/assign/view.php?id=19693>. Acesso em: 05 abr. 2024