

Algebra Linear Computacional COC473 - Lista 6

Bruno Dantas de Paiva
DRE: 118048097

October 21, 2020

1 Questão 1

```
def edo_solver(differential_function, t0, tf, delta, start_condition, control):
    x_incognita = [start_condition]
    t_incognita = [t0]
    steps = int((tf - t0) / (delta))
    if(control == 0):
        print("Euler_Method_Solution")
    elif(control == 1):
        print("Runge_Kutta_Second_Order_Method_Solution")
    else:
        print("Runge_Kutta_Fourth_Order_Method_Solution")

    print("_t", "____", "_x")
    print("0.0", "____", start_condition)

    for i in range(steps):
        t_incognita.append((i + 1) * delta)
        if(control == 0):
            x_incognita.append(x_incognita[i] + delta * differential_function(t_incognita[i]))
        elif(control == 1):
            K1 = differential_function(t_incognita[i], x_incognita[i])
            K2 = differential_function(t_incognita[i] + delta, x_incognita[i] + delta * K1)
            x_incognita.append(x_incognita[i] + delta / 2 * (K1 + K2))
        else:
            K1 = differential_function(t_incognita[i], x_incognita[i])
            K2 = differential_function(t_incognita[i] + delta / 2, x_incognita[i] + delta * K1 / 2)
            K3 = differential_function(t_incognita[i] + delta / 2, x_incognita[i] + delta * K2 / 2)
            K4 = differential_function(t_incognita[i] + delta, x_incognita[i] + delta * K3)
            x_incognita.append(x_incognita[i] + delta / 6 * (K1 + 2 * K2 + 2 * K3 + K4))

    if(i==0 or i == steps):
        continue

    print(round(t_incognita[i], 3), "____", round(x_incognita[i], 5))
    print(round(t_incognita[-1], 3), "____", round(x_incognita[-1], 5))
    print('\n')
```

Para a questão 1, é importante notar que existe um parâmetro chamado de control. Caso este seja 0, é executado o método de euler. Caso seja 1, o método de range-kutta de segunda ordem é executado e, caso contrário, será executado o range-kutta de quarta ordem.

2 Questão 2

2.1 Método de Euler

```
bdantas@Oracle:~/Área de Trabalho/UFRJ/ALC$ python3 main.py
Euler Method Solution
t      x
0.0    1
0.4    1.0
0.8    0.68
1.2    0.38406
1.6    0.24246
2.0    0.16721
2.4    0.12248
2.8    0.09368
3.2    0.07402
3.6    0.05999
4.0    0.04963
4.4    0.04175
4.8    0.03561
5.2    0.03074
5.6    0.02681
6.0    0.02359
6.4    0.02092
6.8    0.01868
7.2    0.01678
7.6    0.01516
8.0    0.01376
8.4    0.01255
8.8    0.01149
9.2    0.01056
9.6    0.00974
10.0   0.00901
```

Figure 1: Imagem contendo os resultados da solução de edo pelo método de Euler.

2.2 Método de Range-Kutta de Segunda Ordem

```
bdantas@Oracle:~/Área de Trabalho/UFRJ/ALC$ python3 main.py
Runge Kutta Second Order Method Solution
t      x
0.0    1
0.4    0.84
0.8    0.60638
1.2    0.42263
1.6    0.29652
2.0    0.21317
2.4    0.15788
2.8    0.12039
3.2    0.09426
3.6    0.07552
4.0    0.06172
4.4    0.05131
4.8    0.04328
5.2    0.03698
5.6    0.03194
6.0    0.02786
6.4    0.0245
6.8    0.02172
7.2    0.01938
7.6    0.0174
8.0    0.0157
8.4    0.01424
8.8    0.01298
9.2    0.01187
9.6    0.0109
10.0   0.01005
```

Figure 2: Imagem contendo os resultados da solução de edo pelo método de Range-Kutta de segunda ordem.

2.3 Método de Range-Kutta de Quarta Ordem

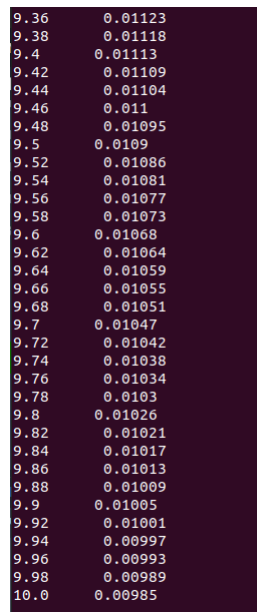
```
bdantas@Oracle:~/Área de Trabalho/UFRJ/ALC$ python3 main.py
Runge Kutta Fourth Order Method Solution
t      x
0.0    1
0.4    0.86166
0.8    0.60946
1.2    0.40998
1.6    0.28112
2.0    0.20018
2.4    0.14805
2.8    0.1132
3.2    0.08902
3.6    0.07167
4.0    0.05885
4.4    0.04914
4.8    0.04161
5.2    0.03567
5.6    0.03091
6.0    0.02703
6.4    0.02384
6.8    0.02117
7.2    0.01893
7.6    0.01702
8.0    0.01539
8.4    0.01398
8.8    0.01275
9.2    0.01168
9.6    0.01074
10.0   0.0099
```

Figure 3: Imagem contendo os resultados da solução de edo pelo método de Range-Kutta de quarta ordem.

É importante ressaltar, que dentre os métodos acima, nem todos os métodos convergiram, portanto, foi necessário para Range-Kutta de segunda ordem e euler, modificar o valor de delta para obter a solução mais próxima possível, conforme podemos observar abaixo

2.4 Método de Euler - Parte 2

Delta utilizado: 0.02

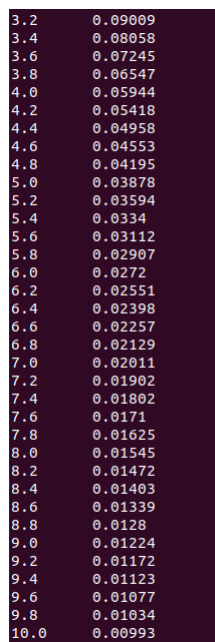


9.36	0.01123
9.38	0.01118
9.4	0.01113
9.42	0.01109
9.44	0.01104
9.46	0.011
9.48	0.01095
9.5	0.0109
9.52	0.01086
9.54	0.01081
9.56	0.01077
9.58	0.01073
9.6	0.01068
9.62	0.01064
9.64	0.01059
9.66	0.01055
9.68	0.01051
9.7	0.01047
9.72	0.01042
9.74	0.01038
9.76	0.01034
9.78	0.0103
9.8	0.01026
9.82	0.01021
9.84	0.01017
9.86	0.01013
9.88	0.01009
9.9	0.01005
9.92	0.01001
9.94	0.00997
9.96	0.00993
9.98	0.00989
10.0	0.00985

Figure 4: Imagem contendo os resultados da solução mais precisa de edo pelo método de Euler.

2.5 Método de Range-Kutta de Segunda Ordem - Parte 2

Delta utilizado: 0.3



3.2	0.09009
3.4	0.08058
3.6	0.07245
3.8	0.06547
4.0	0.05944
4.2	0.05418
4.4	0.04958
4.6	0.04553
4.8	0.04195
5.0	0.03878
5.2	0.03594
5.4	0.0334
5.6	0.03112
5.8	0.02907
6.0	0.0272
6.2	0.02551
6.4	0.02398
6.6	0.02257
6.8	0.02129
7.0	0.02011
7.2	0.01902
7.4	0.01802
7.6	0.0171
7.8	0.01625
8.0	0.01545
8.2	0.01472
8.4	0.01403
8.6	0.01339
8.8	0.0128
9.0	0.01224
9.2	0.01172
9.4	0.01123
9.6	0.01077
9.8	0.01034
10.0	0.00993

Figure 5: Imagem contendo os resultados da solução mais precisa de edo pelo método de Range-Kutta de segunda ordem.

3 Questão 3

```
def second_order_edo_solver(second_order_differential_function, t0, tf, delta, x0, derivada_x0):
    x_incognita = [x0]
    x_actual_line = derivada_x0
    t_incognita = [t0]
    steps = int((tf - t0) / delta)
    for i in range(steps):
        t_incognita.append((i + 1) * delta)

        if(control):
            x2_lines = second_order_differential_function(t_incognita[i], x_incognita[i],
                                                         x_actual_line)
            x_incognita.append(x_incognita[i] + x_actual_line * delta + x2_lines * delta / 2)
            x_actual_line = x_actual_line + x2_lines * delta

        else:
            K1 = delta / 2 * second_order_differential_function(t_incognita[i], x_incognita[i],
                                                                x_actual_line)
            Q = delta / 2 * (x_actual_line + K1 / 2)
            K2 = delta / 2 * second_order_differential_function(t_incognita[i] + delta / 2,
                                                                x_incognita[i] + Q,
                                                                x_actual_line + K1)
            K3 = delta / 2 * second_order_differential_function(t_incognita[i] + delta,
                                                                x_incognita[i] + Q + K2 * delta,
                                                                x_actual_line + K1 + K2 * delta)
            L = delta * (x_actual_line + K3)
            K4 = delta / 2 * second_order_differential_function(t_incognita[i] + delta,
                                                                x_incognita[i] + Q + K2 * delta,
                                                                x_actual_line + K1 + K2 * delta)

            x_incognita.append(x_incognita[i] + delta * (x_actual_line + (K1 + K2 + K3) / 3))
            x_actual_line = x_actual_line + (K1 + 2 * K2 + 2 * K3 + K4) / 3

    plt.plot(t_incognita, x_incognita)
    plt.ylabel('y\'\'(t)')
    plt.xlabel('T')
    plt.show()
```

O parametro control nesta função serve para alternar entre o método da Expansão em Série Taylor(True) e o método de Runge-Kutta-Nistron(False)

4 Questão 4

4.1 Expansão em Série de Taylor

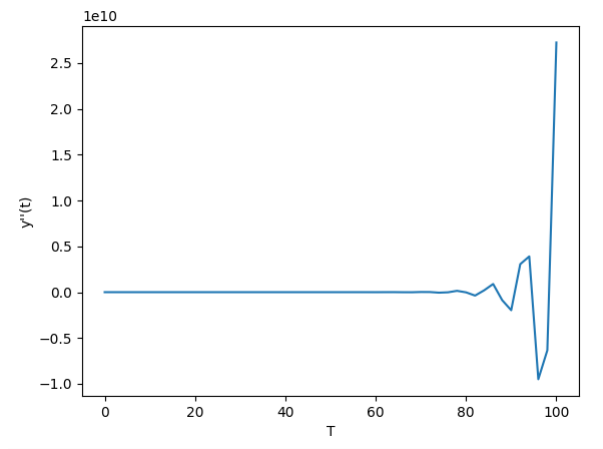


Figure 6: Gráfico resultado da equação diferencial de segunda ordem com $\delta = 2$

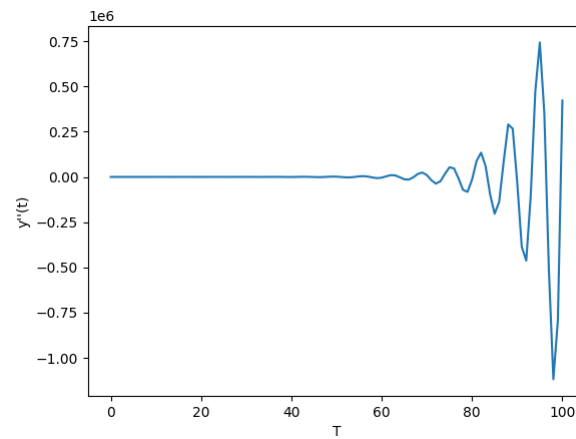


Figure 7: Gráfico resultado da equação diferencial de segunda ordem com $\delta = 1$

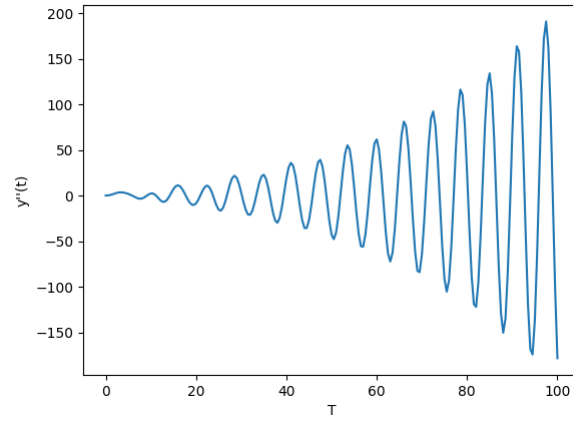


Figure 8: Gráfico resultado da equação diferencial de segunda ordem com $\delta = 0.5$

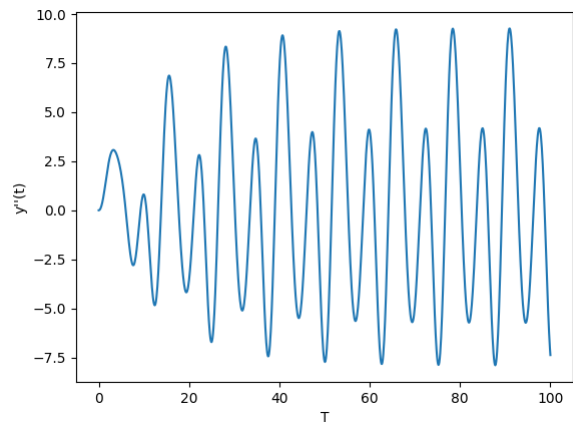


Figure 9: Gráfico resultado da equação diferencial de segunda ordem com $\delta = 0.1$

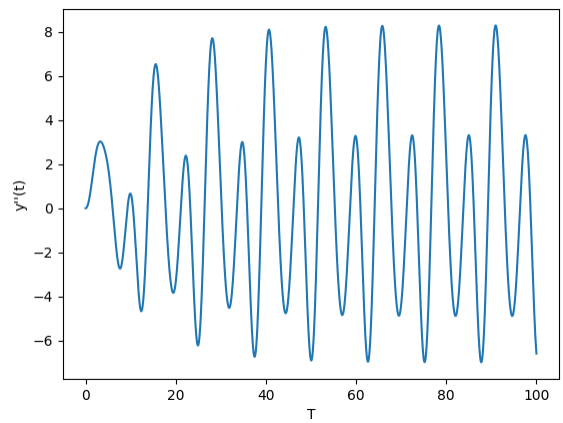


Figure 10: Gráfico resultado da equação diferencial de segunda ordem com $\delta = 0.05$

4.2 Runge- Kutta-Nystrom

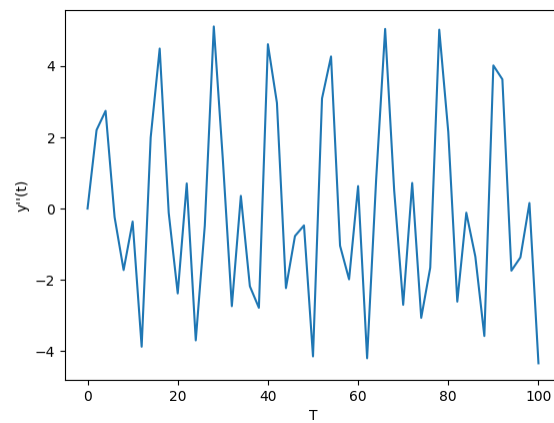


Figure 11: Gráfico resultado da equação diferencial de segunda ordem com $\delta = 2$

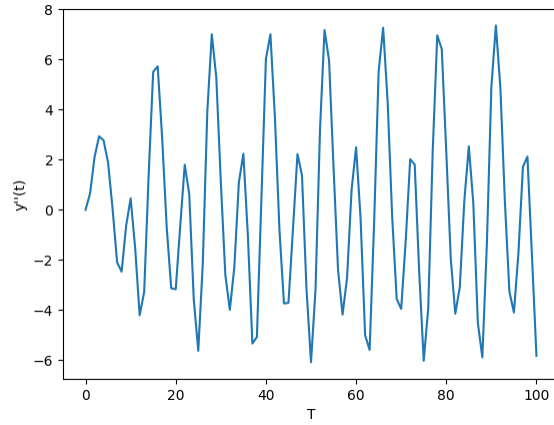


Figure 12: Gráfico resultado da equação diferencial de segunda ordem com $\delta = 1$

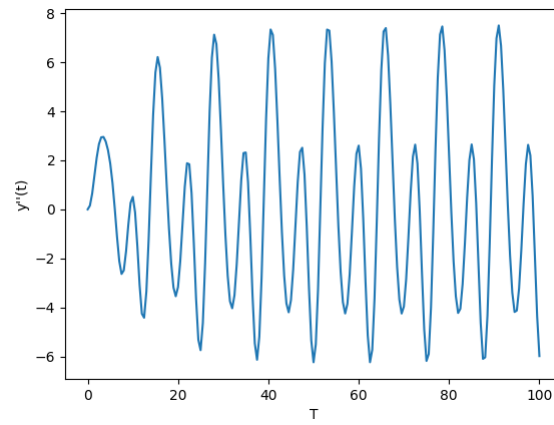


Figure 13: Gráfico resultado da equação diferencial de segunda ordem com $\delta = 0.5$

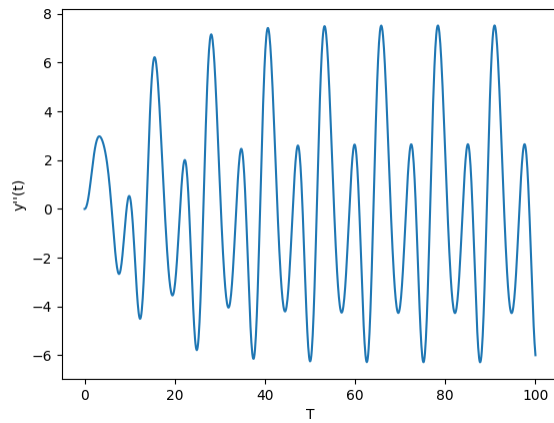


Figure 14: Gráfico resultado da equação diferencial de segunda ordem com $\delta = 0.1$

5 Questão 5

5.1 Expansão em Série de Taylor

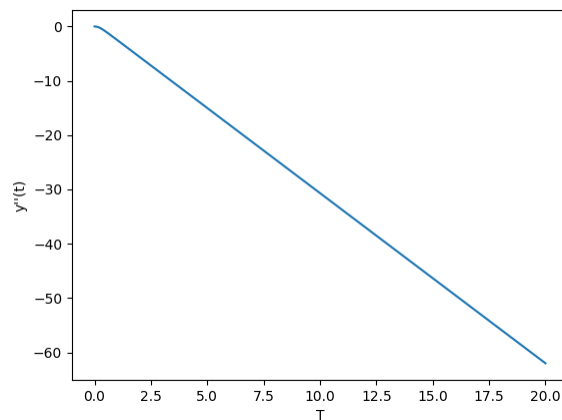


Figure 15: Gráfico resultado da equação diferencial de segunda ordem com $\delta = 0.05$

5.2 Runge- Kutta-Nystron

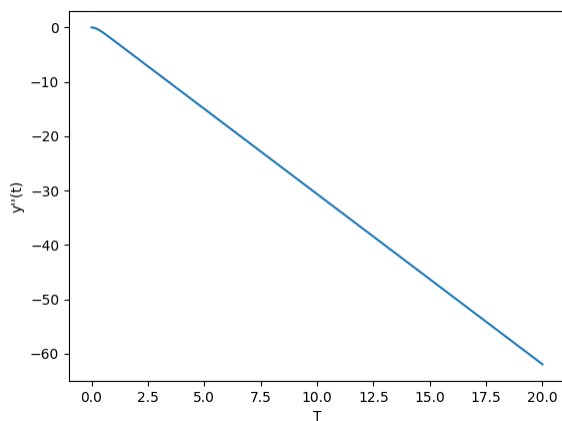


Figure 16: Gráfico resultado da equação diferencial de segunda ordem com $\delta = 0.05$