Universidade de São Paulo – USP

Instituto de Física

Cálculo Numérico com Aplicações em Física - EP3

Aluno: Raphael Rolim
Professor: Arnaldo Gammal

Outubro/2024

Conteúdo

Preâmbulo

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Funções

```
def simpson_integral(function, int_function, a, b, n):
    def f(x):
        return eval(function)
    h = (b - a) / n
    x = np.linspace(a, b, n+1)
    y = f(x)
    integral = h/3 * (y[0] + y[-1] + 4*np.sum(y[1:-1:2]) + 2*np.sum(y[2:-1:2]))
    exact_integral = (eval(int_function.replace('x', 'b')) - eval(int_function.replace('x', 'a')))
    error = np.abs(integral - exact_integral)
    return integral, error

def trapz_integral(function, a, b, n):
    def f(x):
        return eval(function)
    h = (b - a) / n
    x = np.linspace(a, b, n+1)
    y = f(x)
    integral = h * (np.sum(y[1:-1]) - (y[0] + y[-1]) / 2)
    return integral

def random(zi, a=1103515245, c=12345, m=2147483647):
    zi = (a * zi + c) % m
    return zi / m, zi

def MC_integral(function, a, b, n, zi):
    def f(x):
        return eval(function)
    n_in = 0
    x_values = np.linspace(a, b, n+1)
    height = max([f(x) for x in x_values])
    A = (b - a) * height
    for _ in range(n):
        rand_x, zi = random(zi)
        x = a + (b - a) * rand_x
        rand_y, zi = random(zi)
        y = height * rand_y
        if y < f(x):
            n_in += 1
    integral = A * n_in / n
    return integral
```

Questão 1

Item a)

```
data_simps = {'p': [], 'N': [], 'I_{\text{num}}': [], 'Erro': []}
for p in range(1,26):
    N = 2**p
    integral, error = simpson_integral('np.float32(6-6*x**5)', 'np.float32(6*x-x**6)', 0, 1, N)
    data_simps['p'].append(p)
    data_simps['N'].append(N)
    data_simps['I_{\text{num}}'].append(integral)
    data_simps['Erro'].append(error)
df_simps = pd.DataFrame(data=data_simps)
df_simps.to_csv('simpson_integral.csv', index=False)
```

Tabela 1: Resultados do algoritmo de integração pelo método de Simpson com **precisão simples**.

p	N	I_{num}	Erro
1	2	4.875	0.125
2	4	4.9921875	0.0078125
3	8	4.99951171875	0.00048828125
4	16	4.999969482421875	3.0517578125e-05
5	32	4.999998092651367	1.9073486328125e-06
6	64	4.999999682108561	3.178914393942023e-07
7	128	4.99999984105428	1.5894571969710114e-07
8	256	5.0	0.0
9	512	5.0	0.0
10	1024	5.0	0.0
11	2048	4.99999984105428	1.5894571969710114e-07
12	4096	4.999999682108561	3.178914393942023e-07
13	8192	5.0	0.0
14	16384	5.0	0.0
15	32768	5.0	0.0
16	65536	5.0	0.0
17	131072	4.99999984105428	1.5894571969710114e-07
18	262144	4.99999984105428	1.5894571969710114e-07
19	524288	5.000000635782877	6.357828770120477e-07
20	1048576	5.000000476837158	4.76837158203125e-07
21	2097152	5.0000003178914385	3.1789143850602386e-07
22	4194304	4.999998410542806	1.5894571943064761e-06
23	8388608	5.0000011920928955	1.1920928955078125e-06
24	16777216	5.000001867612203	1.8676122026661801e-06
25	33554432	5.0000013311704	1.3311703996876645e-06

Seja a integral

$$\int_0^1 6 - 6x^5 \, \mathrm{d}x = \left[6x - x^6\right]_0^1,$$

a partir disso é possível encontrar os resultados na Tabela 1.

Item b)

```
data_simps_double = {'p': [], 'N': [], 'I_{\text{num}}': [], 'Erro': []}
for p in range(1, 26):
    N = 2**p
    integral, error = simpson_integral('np.float64(6-6*x**5)', 'np.float64(6*x-x**6)', 0, 1, N)
    data_simps_double['p'].append(p)
    data_simps_double['N'].append(N)
    data_simps_double['I_{\text{num}}'].append(integral)
    data_simps_double['Erro'].append(error)
df_simps_double = pd.DataFrame(data=data_simps_double)
df_simps_double.to_csv('simpson_integral_double.csv', index=False)
```

Tabela 2: Resultados do algoritmo de integração pelo método de Simpson com **precisão dupla**.

p	N	I_{num}	Erro
1	2	4.875	0.125
2	4	4.9921875	0.0078125
3	8	4.99951171875	0.00048828125
4	16	4.999969482421875	3.0517578125e-05
5	32	4.999998092651367	1.9073486328125e-06
6	64	4.9999998807907104	1.1920928955078125e-07
7	128	4.999999992549419	7.450580596923828e-09
8	256	4.999999999534339	4.656612873077393e-10
9	512	4.999999999970896	2.9103830456733704e-11
10	1024	4.999999999998181	1.8189894035458565e-12
11	2048	4.999999999999886	1.1368683772161603e-13
12	4096	4.999999999999993	7.105427357601002e-15
13	8192	4.999999999999999	8.881784197001252e-16
14	16384	5.0	0.0
15	32768	5.0	0.0
16	65536	5.0	0.0
17	131072	5.0	0.0
18	262144	5.0	0.0
19	524288	4.999999999999997	2.6645352591003757e-15
20	1048576	5.0	0.0
21	2097152	5.000000000000001	8.881784197001252e-16
22	4194304	5.0	0.0
23	8388608	4.999999999999989	1.0658141036401503e-14
24	16777216	5.000000000000001	8.881784197001252e-16
25	33554432	4.999999999999994	6.217248937900877e-15

```
df_simps_nonzero = df_simps[df_simps['Erro'] != 0]
plt.plot(df_simps_nonzero['p'], np.log2(df_simps_nonzero['Erro']), c='indigo', label='Precisão simples')
df_simps_double_nonzero = df_simps_double[df_simps_double['Erro'] != 0]
plt.plot(df_simps_double_nonzero['p'], np.log2(df_simps_double_nonzero['Erro']), c='orange', label='Precisão dupla')
plt.legend()
plt.xlabel(r'$p$')
plt.ylabel(r'$\log_2$(Erro)')
plt.grid()
plt.show()
```

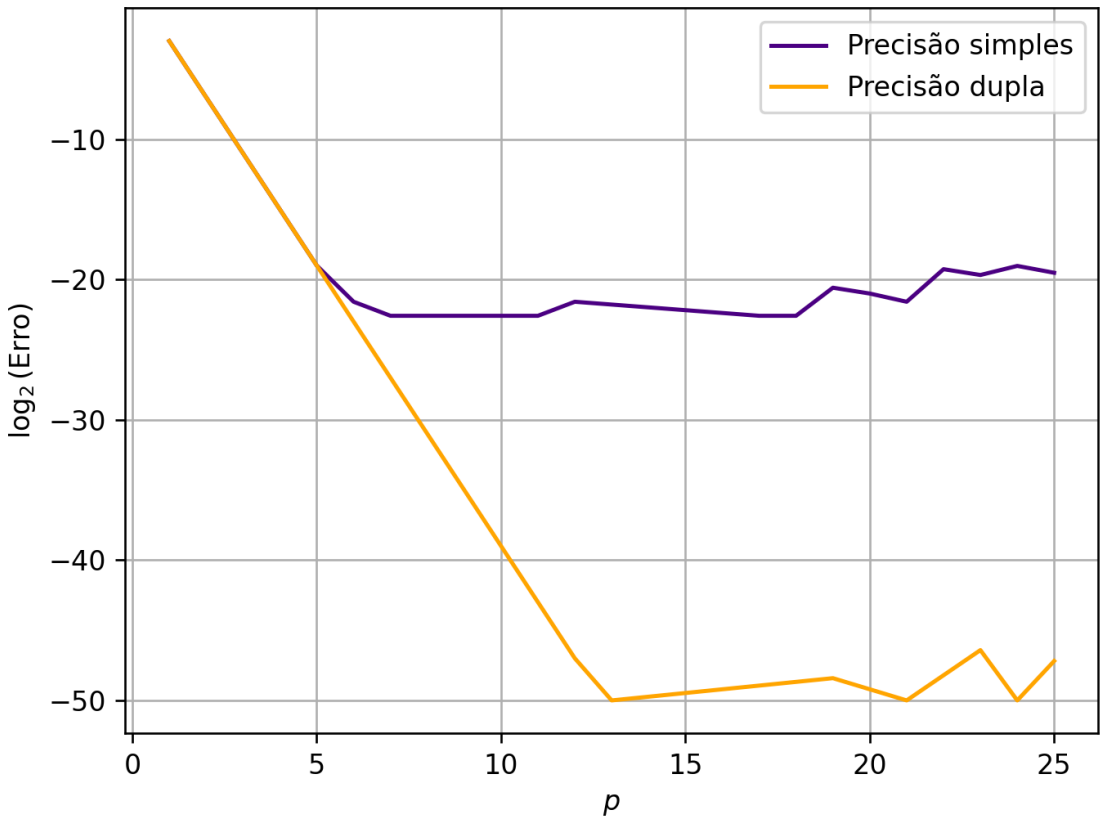

Figura 1: Gráfico do $\log_2$(Erro) em função das iterações p.

Para p pequeno, é possível ver que há uma diminuição clara do erro com o decaimento aproximadamente proporcional a $\mathcal{O}(h^4)$, o que é esperado para o método de Simpson e podemos ver esse padrão em ambos os casos.

Já para p maior, é visível a estabilização do erro (problema de "Roundoff"), ou seja, independente do aumento dos valores de N, temos que o erro não irá mudar muito por conta do arredondamento de ponto flutuante. Podemos ver esse efeito em $p \approx 7$ para a precisão simples e $p \approx 13$ para a precisão dupla.

Sendo assim, podemos ver que para p maior, as curvas seguem o comportamento de $\mathcal{O}(\sqrt{N})$ por conta da estabilização dos erros (ou até leves alterações).

Questão 2

Utilizando $l = 1$ m, $g = 9.807$ m/s^2 e $N = 1000$ divisões trapezoidais, temos:

```
l = 1 # metros
g = 9.807 # m/s^2
theta_0 = np.linspace(0, np.pi, 10)
k = (1 - np.cos(theta_0)) / 2
T_gal = 2*np.pi*np.sqrt(l/g)
N = 1000
data_trapz = {'\theta_0':[], 'T/T_{\text{gal}}': []}
for i in range(len(theta_0)):
    integral = trapz_integral('1/np.sqrt(1-(k[i]*np.sin(x))**2)', 0, np.pi/2, N)
    T = 4*np.sqrt(l/g)*integral
    data_trapz['\theta_0'].append(theta_0[i])
    data_trapz['T/T_{\text{gal}}'].append(T / T_gal)

df_trapz = pd.DataFrame(data=data_trapz)
df_trapz.to_csv('trapz_integral_10.csv', index=False)
```

Tabela 3: Tabela com os resultados para 10 valores de θ_0.

θ_0	T/T_{Galileu}
0.0	0.9980000000000001
0.3490658503988659	0.9982269726483416
0.6981317007977318	1.001440621628954
1.0471975511965976	1.0141665645380695
1.3962634015954636	1.0452383774941036
1.7453292519943295	1.1059854466966184
2.0943951023931953	1.2140620213269666
2.443460952792061	1.4060048062650183
2.792526803190927	1.7937192552518408
3.141592653589793	Não Converge

```
l = 1 # metros
g = 9.807 # m/s^2
theta_0 = np.linspace(0, np.pi, 1000)
k = (1 - np.cos(theta_0)) / 2
T_gal = 2*np.pi*np.sqrt(l/g)
N = 1000
data_trapz = {'\theta_0':[], 'T/T_{\text{gal}}': []}
for i in range(len(theta_0)):
    integral = trapz_integral('1/np.sqrt(1-(k[i]*np.sin(x))**2)', 0, np.pi/2, N)
    T = 4*np.sqrt(l/g)*integral
    data_trapz['\theta_0'].append(theta_0[i])
    data_trapz['T/T_{\text{gal}}'].append(T / T_gal)

df_trapz = pd.DataFrame(data=data_trapz)
df_trapz.to_csv('trapz_integral.csv', index=False)
```

```
plt.plot(df_trapz['\theta_0'], df_trapz['T/T_{\text{gal}}'], color='indigo')
plt.xlabel(r'$\theta_0$ (rad)')
plt.ylabel(r'$T/T_{Galileu}$')
plt.grid()
plt.show()
```

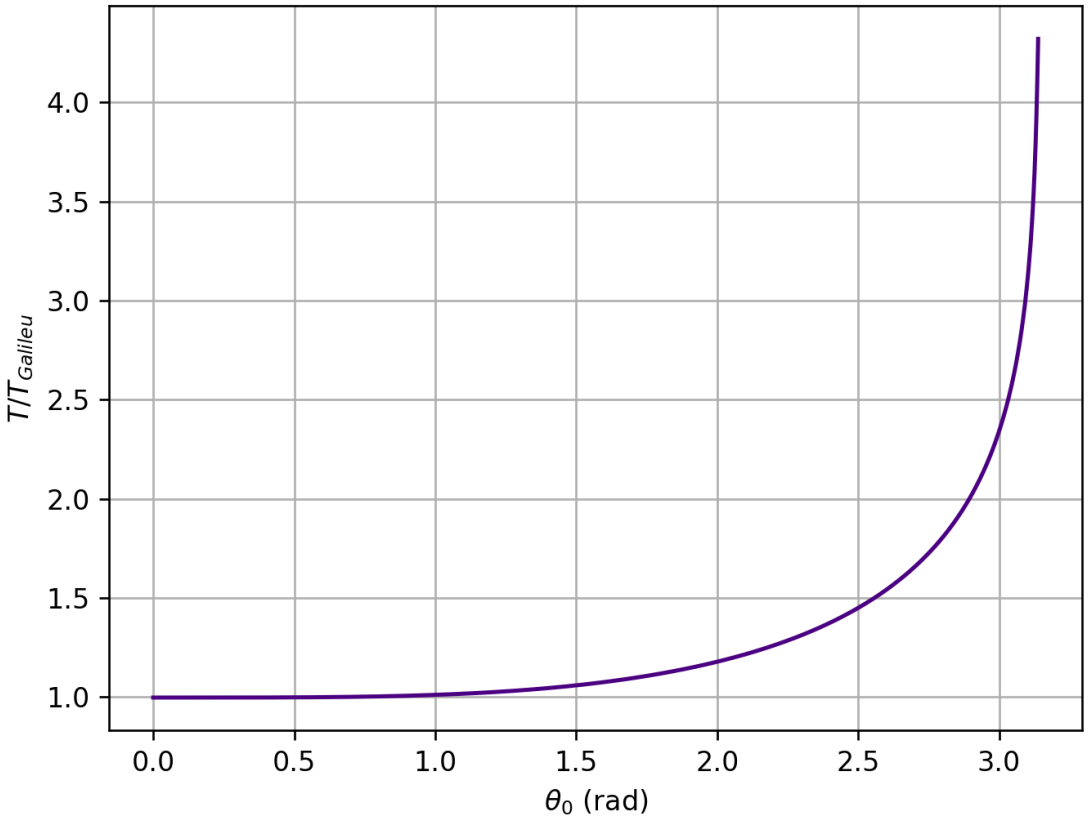

Figura 2: Gráfico de T/T_{Galileu} com os resultados para 1000 valores de θ_0.

Questão 3

Item a)

```
print(random(12610413)[0])
```

O resultado é $U_i = 0.4246220939907348$.

Item b)

```
print(MC_integral('1-x**2', 0, 1, 100, 12610413))
```

Para uma única tentativa, o resultado da integral é 0.65.

Item c)

```
I = []
seed = 12610413
MC_data = {'N_t': [], 'I_m':[], 'sigma': [], 'sigma_m': []}
for p in range(1, 18):
    N_t = 2**p
    seeds = np.zeros(N_t)
    for n in range(N_t):
        _, seed = random(seed)
        seeds[n] = seed
    I = np.array([MC_integral('1-x**2', 0, 1, 100, s) for s in seeds])
    I_m = np.sum(I) / N_t
    sigma = np.sqrt(np.sum((I - I_m)**2) / (N_t - 1))
    sigma_m = sigma / np.sqrt(N_t)
    MC_data['N_t'].append(N_t)
    MC_data['I_m'].append(I_m)
    MC_data['sigma'].append(sigma)
    MC_data['sigma_m'].append(sigma_m)
df_MC = pd.DataFrame(data=MC_data)
df_MC.to_csv('MC_integral.csv', index=False)
```

Tabela 4: Tabela com os resultados do método de Monte Carlo para integração.

N_t	I_m	σ	σ_m
2	0.65	0.014142135623730963	0.010000000000000009
4	0.6874999999999999	0.055602757725374256	0.027801378862687128
8	0.66	0.03505098327538656	0.01239239398064105
16	0.6625	0.04538722287164086	0.011346805717910215
32	0.665625	0.04376900508866534	0.007737340075995941
64	0.6615625	0.047248708725112006	0.005906088590639001
128	0.670078125	0.044536035020580025	0.003936466546276712
256	0.6677734375	0.04869395659949729	0.0030433722874685805
512	0.66564453125	0.04837854857482247	0.0021380499850762345
1024	0.6670800781249999	0.04556337291843686	0.0014238554037011518
2048	0.6681396484375	0.047061228089843934	0.0010399160472904837
4096	0.666083984375	0.0473946028399622	0.0007405406693744093
8192	0.6665917968749999	0.04707698566718133	0.000520132121923216
16384	0.6670953369140624	0.04701858077300506	0.00036733266228910203
32768	0.66574951171875	0.0471141075214481	0.0002602711321716256
65536	0.6661865234375001	0.04706484197931488	0.00018384703898169876
131072	0.6664447784423827	0.04695991718663383	0.00012970967143213858