Projeto 2

Matemática Computacional

2024/2025

Grupo 9:

Afonso da Conceição Ribeiro	ist1102763
Margarida Rodrigues da Silva Freire	ist1109526
Maria Marta Veloza Silva	ist1109604
Sara Jacinto Costa	ist1110787

Licenciatura em Matemática Aplicada e Computação
Instituto Superior Técnico – Universidade de Lisboa

Índice

Grupo I

1.

Pretende-se aproximar o integral $I(f) = \int_a^b f(x)\,dx$ utilizando o **método de Romberg**.
Depois de analisar o enunciado, começa-se por apontar algumas informações relevantes para a resolução do exercício.
O algoritmo conhecido por **integração de Romberg** pode ser calculado recursivamente da seguinte forma:

$$R_{j,0} = T_{2^j}(f) \quad , \ j = 0, \ldots, n \quad ; \text{ para } k = 1, \ldots, n$$
$$R_{j,k} = \frac{4^k R_{j,k-1} - R_{j-1,k-1}}{4^k - 1} \quad , \ j = k, \ldots, n.$$

O objetivo deste exercício é mostrar que $R_{j,1}$ corresponde à regra de Simpson $S_{2^j}(f)$, $j = 1, ..., n$.

A regra de Simpson composta, para n subintervalos, com n par, é representada pela seguinte fórmula:

$$S_n(f) = \frac{h}{3}\left[f(x_0) + f(x_n) + 4\sum_{i=1}^{\frac{n}{2}} f(x_{2i-1}) + 2\sum_{i=1}^{\frac{n}{2}-1} f(x_{2i})\right],$$
$$\text{com } \ h = \frac{b-a}{n} \quad , \qquad x_i = a + ih \quad (i = 0, \ldots, n)$$

Sabe-se ainda que a regra dos trapézios para n subintervalos é dada por:

$$T_n(f) = h\left[\frac{f(x_0) + f(x_n)}{2} + \sum_{i=1}^{n-1} f(x_i)\right],$$
$$\text{com } \ h = \frac{b-a}{n} \quad , \qquad x_i = a + ih \quad (i = 0, \ldots, n)$$

Consoante os dados apresentados no enunciado, tem-se que:

$$S_{2^j}(f) = \frac{h_{2^j}}{3}\left[f(x_0) + f(x_{2^j}) + 4\sum_{i=1}^{2^{j-1}} f(x_{2i-1}) + 2\sum_{i=1}^{2^{j-1}-1} f(x_{2i})\right] , \qquad h_{2^j} = \frac{b-a}{2^j}$$

De seguida, calcula-se $R_{j,1}$.

$$R_{j,1} = \frac{4R_{j,0} - R_{j-1,0}}{3} = \frac{4\,T_{2^j}(f) - T_{2^{j-1}}(f)}{3}$$

Antes de prosseguir com os cálculos, para diferenciar os pontos consoante o número total de subintervalos a que correspondem (2^j ou 2^{j-1} subintervalos), considera-se a seguinte **notação**:

- $T_{2^j}(f) = h_{2^j}\left[\frac{f(x_0) + f(x_{2^j})}{2} + \sum_{i=1}^{2^j-1} f(x_i)\right] , \qquad h_{2^j} = \frac{b-a}{2^j}$
- $T_{2^{j-1}}(f) = h_{2^{j-1}}\left[\frac{f(x'_0) + f(x'_{2^{j-1}})}{2} + \sum_{i=1}^{2^{j-1}-1} f(x'_i)\right] , \qquad h_{2^{j-1}} = \frac{b-a}{2^{j-1}}$

Sabe-se que:

- $f(x_0) = f(x'_0)$
- $f(x_{2^j}) = f(x'_{2^{j-1}})$

Tendo em atenção os cálculos auxiliares apresentados mais abaixo, indicados pelos números entre parêntesis, tem-se que:

$$R_{j,1} = \frac{4R_{j,0} - R_{j-1,0}}{3} = \frac{4\,T_{2^j}(f) - T_{2^{j-1}}(f)}{3} =$$

$$= \frac{1}{3}\left[4h_{2^j}\left(\frac{f(x_0)+f(x_{2^j})}{2} + \sum_{i=1}^{2^j-1} f(x_i)\right) - h_{2^{j-1}}\left(\frac{f(x'_0)+f(x'_{2^{j-1}})}{2} + \sum_{i=1}^{2^{j-1}-1} f(x'_i)\right)\right] =$$

$$= \frac{h_{2^j}}{3}\left[2f(x_0) + 2f(x_{2^j}) + 4\sum_{i=1}^{2^j-1} f(x_i) - \frac{h_{2^{j-1}}}{h_{2^j}}\left(\frac{f(x'_0)+f(x'_{2^{j-1}})}{2} + \sum_{i=1}^{2^{j-1}-1} f(x'_i)\right)\right] =$$

$$\overset{(1)}{=} \frac{h_{2^j}}{3}\left[2f(x_0) + 2f(x_{2^j}) + 4\sum_{i=1}^{2^j-1} f(x_i) - 2\left(\frac{f(x'_0)+f(x'_{2^{j-1}})}{2} + \sum_{i=1}^{2^{j-1}-1} f(x'_i)\right)\right] =$$

$$= \frac{h_{2^j}}{3}\left[2f(x_0) + 2f(x_{2^j}) + 4\sum_{i=1}^{2^j-1} f(x_i) - f(x'_0) - f(x'_{2^{j-1}}) - 2\sum_{i=1}^{2^{j-1}-1} f(x'_i)\right] =$$

$$\overset{(2)}{=} \frac{h_{2^j}}{3}\left[f(x_0) + f(x_{2^j}) + 4\sum_{i=1}^{2^{j-1}} f(x_{2i-1}) + 4\sum_{i=1}^{2^{j-1}-1} f(x_{2i}) - 2\sum_{i=1}^{2^{j-1}-1} f(x'_i)\right] =$$

$$\overset{(3)}{=} \frac{h_{2^j}}{3}\left[f(x_0) + f(x_{2^j}) + 4\sum_{i=1}^{2^{j-1}} f(x_{2i-1}) + 4\sum_{i=1}^{2^{j-1}-1} f(x_{2i}) - 2\sum_{i=1}^{2^{j-1}-1} f(x_{2i})\right] =$$

$$= \frac{h_{2^j}}{3}\left[f(x_0) + f(x_{2^j}) + 4\sum_{i=1}^{2^{j-1}} f(x_{2i-1}) + 2\sum_{i=1}^{2^{j-1}-1} f(x_{2i})\right] =$$

$$= S_{2^j}(f)$$

Cálculos e explicações auxiliares:

1.

$$\frac{h_{2^{j-1}}}{h_{2^j}} = \frac{\frac{b-a}{2^{j-1}}}{\frac{b-a}{2^j}} = \frac{2^j(b-a)}{2^{j-1}(b-a)} = 2 \tag{1}$$

2.

$$\sum_{i=1}^{2^j-1} f(x_i) = \underbrace{\sum_{i=1}^{\frac{2^j}{2}} f(x_{2i-1})}_{\text{índices ímpares}} + \underbrace{\sum_{i=1}^{\frac{2^j-2}{2}} f(x_{2i})}_{\text{índices pares}} = \sum_{i=1}^{2^{j-1}} f(x_{2i-1}) + \sum_{i=1}^{2^{j-1}-1} f(x_{2i}) \tag{2}$$

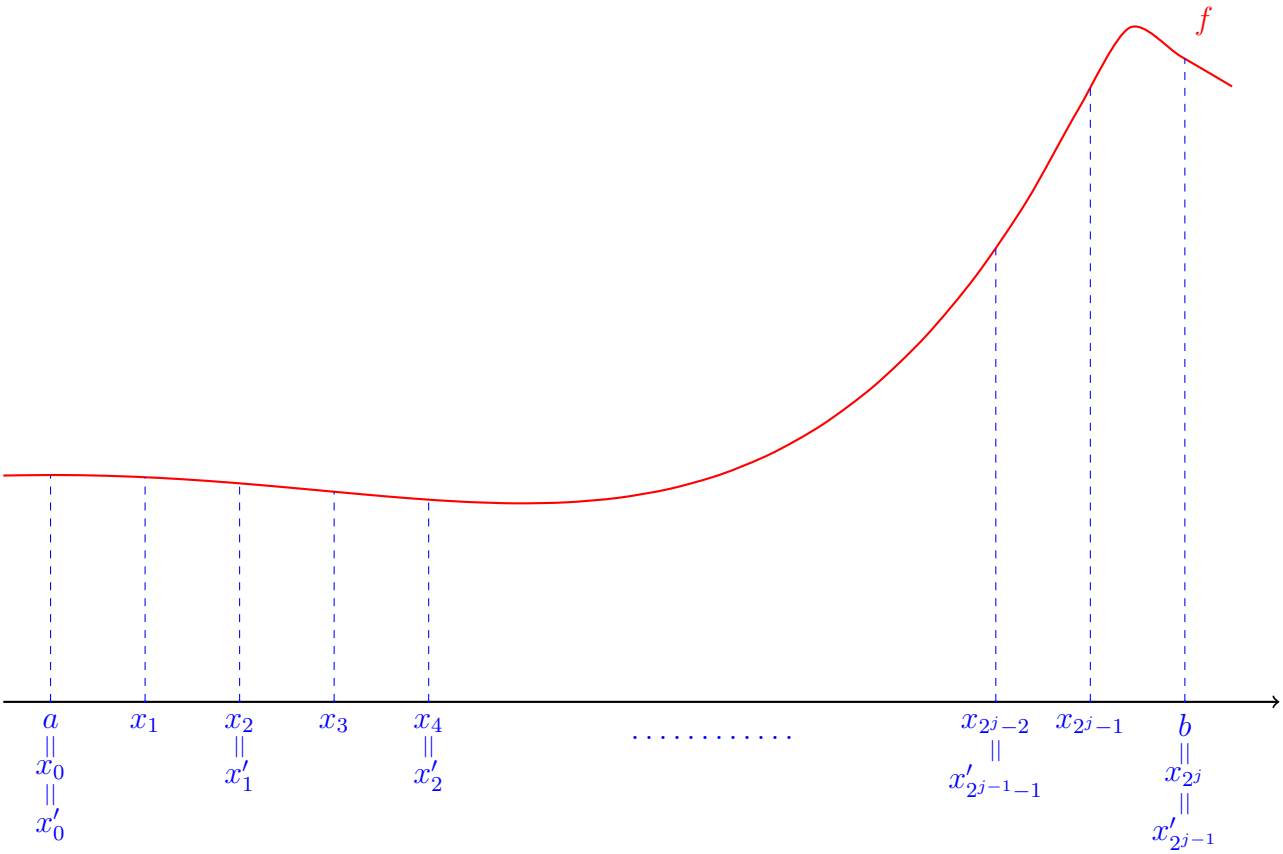
O último termo, $f(x_{2^j-1})$, tem índice ímpar.

- índice do último termo dos ímpares: $2^j - 1 = 2i - 1 \Leftrightarrow i = \frac{2^j}{2}$
- índice do último termo dos pares: $2^j - 2 = 2i \Leftrightarrow i = \frac{2^j-2}{2}$

3. Verifica-se que:

$$\sum_{i=1}^{2^{j-1}-1} f(x'_i) = \sum_{i=1}^{2^{j-1}-1} f(x_{2i}) \tag{3}$$

A observação do gráfico abaixo (onde está representada uma função f arbitrária) pode facilitar a compreensão da igualdade (3).



Pode-se observar que $x'_i = x_{2i}$. Ou seja:

$$\begin{aligned}\sum_{i=1}^{2^{j-1}-1} f(x'_i) &= f(x'_1) + f(x'_2) + \ldots + f(x'_{2^{j-1}-2}) + f(x'_{2^{j-1}-1}) = \\ &= f(x_2) + f(x_4) + \ldots + f(x_{2^j-4}) + f(x_{2^j-2}) = \\ &= \sum_{i=1}^{2^{j-1}-1} f(x_{2i})\end{aligned}$$

Assim, conclui-se que $R_{j,1} = S_{2^j}(f)$.

2.

Como se têm o integral:

$$I(f) = \int_a^b f(x)\,dx$$

Para se poder usar a regra dos trapézios, divide-se o intervalo $[a, b]$ em n subintervalos de largura constante h.

A função $f(x)$ é aproximada por uma linha entre dois pontos consecutivos: $(x_i, f(x_i))$ e $(x_{i+1}, f(x_{i+1}))$.
Cada pedaço da integral

$$\int_{x_i}^{x_{i+1}} f(x)\,dx$$

é aproximado pela área do trapézio:

- **Base:** h
- **Alturas:** $f(x_i)$ e $f(x_{i+1})$

Assim a área do trápézio é:

$$A = \frac{h}{2}\left[f(x_i) + f(x_{i+1})\right]$$

e a soma de todas as áreas é:

$$I \approx \sum_{i=0}^{n-1} \frac{h}{2}\left[f(x_i) + f(x_{i+1})\right]$$

Chegando assim à formúla dado que os pontos médios aparecem duas vezes:

$$T_n(f) = h\left[\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(x_i)\right]$$

Que também é a fórmula da regra dos trapézios.
Com isto, foi feito este código:

```
function I = trapezios(f, a, b, n)

    format long

    if mod(n, 1) ~= 0 || n <= 0
        error("O numero de divisoes deve ser um inteiro positivo");
    end

    h = (b - a) / n;

    xi = linspace(a, b, n + 1);

    y = f(xi);

    I = h * ((y(1)+y(end))/2 + sum(y(2:end-1)));
end
```

Para isso, foi necessário fornecer como *inputs*: $f(x)$, que é uma função real, os limites de integração a e b (valores reais), e n, um número natural. O *output* corresponde ao resultado da aplicação da regra dos trapézios ao integral $I(f)$.

3.

Utilizando o método de Romberg, descrito anteriormente e com auxílio da função `trapezios`, elaborada no exercício acima, procurou-se aproximar o valor do integral $I(f)$. Além disso, foi considerado como critério de paragem a condição:

$$|R_{k,k} - R_{k-1,k-1}| < \varepsilon,$$

onde ε representa a tolerância do erro admitido.

```
function [prev,current] = romberg(f, a, b, n, epsilon)

cond = true;
j = 0;
current = [];

while j <= n && cond
    prev = current;
    current = zeros(1, j + 1);
    current(1) = trapezios(f, a, b, 2^j);

    for i=1:j
        current(i + 1) = (4^i * current(i) - prev(i)) / (4^i - 1);
    end

    if j > 0
        cond = ~(abs(prev(end) - current(end)) < epsilon);
    end

    j = j + 1;

end

if j > n
    warning('O metodo nao convergiu dentro do numero de iteradas
   especificado')
else
    disp(['O metodo convergiu em ', num2str(j), ' iteradas'])
end
end
```

Neste código, é necessário fornecer como *inputs*: $f(x)$, que é uma função real; a e b, que são os limites reais de integração; n, um número natural; e ε, a tolerância de erro. O *output* corresponde aos valores da iteração anterior e aos valores da iteração atual.

4.

(a)

Com o objetivo de aproximar o valor de π, usando integração numérica, considera-se a aproximação

$$\pi = 4 \int_0^1 \frac{1}{1+x^2}\, dx \quad .$$

Pretende-se usar a integração de Romberg $(R_{j,k})$ para calcular aproximações de π através da regra dos trapézios, T_m, com $m = 1, 2, 4, \ldots, 32, 64, \ldots, 2^N$ subintervalos de integração em $[0, 1]$.

Sabe-se que,

$$R_{j,0} = T_m \ , \qquad \text{quando} \quad m = 2^j \ , \quad k = 0 \quad \text{e} \quad j = 0, \ldots, n \ .$$

ou seja, a integração de Romberg corresponde à regra dos trapézios nestas condições.

Para a resolução do problema, considerou-se $N = 8$.
Para além disso, o valor real de π considerado foi o seguinte:

$$\pi \approx 3,141592653589793\ldots$$

Este valor tem exatamente 15 casas decimais, devido ao facto de os cálculos serem efetuados no MATLAB, que está associado a um sistema de ponto flutuante característico. Portanto, um maior número de casa decimais não iria alterar os resultados obtidos ao longo da resolução.

Com o auxílio da função `romberg` elaborada no exercício anterior, escreveu-se um script no MATLAB com o objetivo de construir uma tabela com todas as aproximações calculadas através do método de Romberg, definido recursivamente por:

$$R_{j,0} = T_m \ , \qquad m = 2^j \ , \qquad j = 0, \ldots, n$$

$$R_{j,k} = \frac{4^k R_{j,k-1} - R_{j-1,k-1}}{4^k - 1} \quad , \ k = 1, \ldots, n \quad , \qquad j = k, \ldots, n$$

O código seguinte foi utilizado para calcular os valores de $R_{j,k}$, com $k = 0, \ldots, 8, \quad j = k, \ldots, 8$, e também para calcular os erros associados a estas aproximações.

```
format long

f = @(x) (4./(1 + x.^2));
a = 0;
b = 1;
n = 8;
epsilon = 0.5*10^(-16);

% Resultado:
[I, table] = romberg(f,a,b,n,epsilon)

% I: resultado final da aplicação do método de Romberg
% table: matriz com todas as aproximações calculadas pelo método de Romberg

% Cálculo dos erros absolutos:
3.141592653589793 - table
```

Os resultados obtidos foram registados na tabela apresentada abaixo, sendo que os dígitos corretos nas aproximações obtidas para o número π estão assinalados a vermelho.

$m = 2^j$	j	k		
		0	1	2
1	0	3.000000000000000	—	—
2	1	3.100000000000000	3.133333333333333	—
4	2	3.131176470588235	3.141568627450981	3.142117647058824
8	3	3.138988494491089	3.141592502458707	3.141594094125888
16	4	3.140941612041389	3.141592651224822	3.141592661142563
32	5	3.141429893174975	3.141592653552836	3.141592653708038
64	6	3.141551963485655	3.141592653589216	3.141592653591641
128	7	3.141582481063752	3.141592653589784	3.141592653589822
256	8	3.141590110458282	3.141592653589793	3.141592653589794

$m = 2^j$	j	k		
		3	4	5
1	0	—	—	—
2	1	—	—	—
4	2	—	—	—
8	3	3.141585783761874	—	—
16	4	3.141592638396796	3.141592665277717	—
32	5	3.141592653590029	3.141592653649611	3.141592653638244
64	6	3.141592653589794	3.141592653589793	3.141592653589734
128	7	3.141592653589794	3.141592653589794	3.141592653589794
256	8	3.141592653589793	3.141592653589793	3.141592653589793

$m = 2^j$	j	k		
		6	7	8
1	0	—	—	—
2	1	—	—	—
4	2	—	—	—
8	3	—	—	—
16	4	—	—	—
32	5	—	—	—
64	6	3.141592653589723	—	—
128	7	3.141592653589794	3.141592653589794	—
256	8	3.141592653589793	3.141592653589793	3.141592653589793

De seguida, apresenta-se uma tabela com os erros absolutos correspondentes aos resultados das tabelas anteriores, assumindo que o valor teórico é dado por $\pi = 3.141592653589793$.

$m = 2^j$	j	k		
		0	1	2
1	0	0.141592653589793	—	—
2	1	0.041592653589793	0.008259320256460	—
4	2	0.010416183001558	0.000024026138812	0.000524993469031
8	3	0.002604159098704	0.000000151131086	0.000001440536095
16	4	0.000651041548404	0.000000002364971	0.000000007552770
32	5	0.000162760414818	0.000000000036957	0.000000000118245
64	6	0.000040690104138	0.000000000000577	0.000000000001848
128	7	0.000010172526041	0.000000000000009	0.000000000000029
256	8	0.000002543131511	0	0.000000000000001

$m = 2^j$	j	k		
		3	4	5
1	0	—	—	—
2	1	—	—	—
4	2	—	—	—
8	3	0.000006869827919	—	—
16	4	0.000000015192997	-0.000000011687924	—
32	5	0.000000000000236	0.000000000059817	0.000000000048451
64	6	0.000000000000001	0	0.000000000000059
128	7	0.000000000000001	0.000000000000001	0.000000000000001
256	8	0	0	0

$m = 2^j$	j	k		
		6	7	8
1	0	—	—	—
2	1	—	—	—
4	2	—	—	—
8	3	—	—	—
16	4	—	—	—
32	5	—	—	—
64	6	0.000000000000071	—	—
128	7	0.000000000000001	0.000000000000001	—
256	8	0	0	0

Estes valores foram calculados com o auxílio do MATLAB, no entanto, devido a erros de precisão característicos deste software, fizeram-se alguns ajustes em certos valores antes de os colocar na tabela (nomeadamente, o resultado das subtrações $3.141592653589793 - 3.141592653589794$, que é 0 quando é calculado no MATLAB, em vez de -0.000000000000001).

Como

$$R_{j,0} = T_m \,, \qquad \text{com} \;\; m = 2^j \,, \quad k = 0 \quad \text{e} \quad j = 0, \ldots, 8 \,,$$

então os valores da primeira coluna da tabela, com $k = 0$, correspondem às aproximações da regra dos trapézios.

Seja $f(x) = \frac{4}{1+x^2}$, verifica-se que $T_{256}(f) \approx 3.141590110458282$, cujo erro absoluto associado é 0.000002543131511. Por outro lado, tem-se que $R_{8,8} \approx 3.141592653589793$, que tem todos os dígitos corretos.

Através da análise das tabelas obtidas e tendo em conta outras observações efetuadas, conclui-se que a integração de Romberg é bastante eficaz no cálculo de aproximações do número π, mesmo para valores de N não muito grandes.

(b)

Pretende-se verificar que os valores obtidos para as aproximações $R_{j,0}$, $j = 0, 1, 2, \ldots, 5, 6, \ldots$, estão de acordo com a ordem de convergência da regra dos trapézios (ordem 2).

Começa-se por notar que

$$R_{j,0} = T_m \ , \qquad \text{com } \ m = 2^j \ \text{ e } \ j = 0, \ldots, n \ .$$

Sabe-se que a fórmula de erro associada à regra dos trapézios para m subintervalos é a seguinte:

$$E_m(f) = -\frac{(b-a)}{12} h^2 f''(\xi) \quad , \qquad \text{se } f \in C^2([a,b]) \quad , \qquad h = \frac{b-a}{m} \quad , \qquad \xi \in (a,b)$$

Na sequência do exercício anterior, considerando que $f(x) = \frac{4}{1+x^2}$, $a = 0$ e $b = 1$, pode-se assumir a seguinte aproximação da fórmula de erro da regra dos trapézios:

$$E \approx C\, h^p \ ,$$

tendo em conta que

- E é o erro absoluto;
- $h = \frac{b-a}{m}$, neste caso temos $h = \frac{1}{2^j}$ (corresponde à medida de cada subintervalo);
- p é a ordem de convergência ($p = 2$ é a ordem de convergência da regra dos trapézios);
- C é uma constante ($C := -\frac{(b-a)}{12} f''(\xi)$).

Ao aplicar a função logaritmo a ambos os lados da igualdade obtida para a fórmula de erro, tem-se que:

$$\log(E) \approx \log(C) + p \log(h)$$

O que sugere uma relação linear entre $\log(E)$ e $\log(h)$, à qual se pode aplicar regressão linear para estimar o valor de p. Ou seja, sabendo que a fórmula geral da regressão linear é do tipo $y = ax + b$, neste caso tem-se que

$$\underbrace{\log(E)}_{y} \approx \underbrace{\log(C)}_{b} + \underbrace{p}_{a} \underbrace{\log(h)}_{x} \quad .$$

Para calcular uma estimativa de p através da regressão linear, utilizaram-se os valores dos erros absolutos calculados no exercício anterior (4.a.), presentes na tabela acima.

São necessários apenas os valores da primeira coluna da tabela, visto que estes correspondem aos resultados das aproximações de $R_{j,0}$. Considerou-se $n = 8$, sabendo que n representa o maior número tomado por j na aplicação do método de Romberg, ou seja, $j = 0, 1, \ldots, 8$.

Com o auxílio da função `polyfit`, criou-se o seguinte script:

```
format long

erros = [0.141592653589793;
         0.041592653589793;
         0.010416183001558;
         0.002604159098704;
         0.000651041548404;
         0.000162760414818;
         0.000040690104138;
         0.000010172526041;
         0.000002543131511];

j = linspace(0,8,9);

h = 1./(2.^j);

log_h = log(h);
log_E = log(erros);

pol = polyfit(log_h, log_E, 1);

pol

fprintf('A ordem de convergencia estimada p e aproximadamente: %.15f\n', pol
    (1));

% Output:
% pol =
%    1.984188170912992  -1.853917939033811

% A ordem de convergencia estimada p e aproximadamente: 1.984188170912992
```

Assim, podem-se concluir que a ordem de convergência calculada com o MATLAB,

$$p_8 = 1.984188170912992 \approx 2 \ ,$$

está próxima da ordem de convergência da regra dos trapézios.

Com o auxílio da função `romberg` elaborada no exercício 3., pode-se também calcular os valores de $R_{j,0}$, com $j = 0, 1, 2, \ldots, 17, 18$, para verificar se é possível ter uma melhor aproximação do coeficiente p.
A tabela abaixo contém os valores calculados para $R_{j,0}$, com $j = 0, 1, 2, \ldots, 17, 18$, que correspondem à primeira coluna da matriz calculada com o seguinte código

```
    romberg(@(x) (4./(1 + x.^2)), 0, 1, 18, 0.5*10^(-16))
```

e tem também os erros absolutos correspondentes a esses valores, tomando $\pi \approx 3,141592653589793$ como o valor de referência.

j	$R_{j,0}$	Erros absolutos
0	3.000000000000000	0.141592653589793
1	3.100000000000000	0.041592653589793
2	3.131176470588235	0.010416183001558
3	3.138988494491089	0.002604159098704
4	3.140941612041389	0.000651041548404
5	3.141429893174975	0.000162760414818
6	3.141551963485655	0.000040690104138
7	3.141582481063752	0.000010172526041
8	3.141590110458282	0.000002543131511
9	3.141592017806915	0.000000635782878
10	3.141592494644073	0.000000158945721
11	3.141592613853365	0.000000039736428
12	3.141592643655685	0.000000009934108
13	3.141592651106265	0.000000002483528
14	3.141592652968909	0.000000000620884
15	3.141592653434575	0.000000000155218
16	3.141592653550986	0.000000000038807
17	3.141592653580092	0.000000000009701
18	3.141592653587368	0.000000000002425

Recorrendo à função `polyfit`, escreveu-se o seguinte programa MATLAB para calcular uma estimativa da ordem de convergência.

```
format long

erros = [0.141592653589793;
         0.041592653589793;
         0.010416183001558;
         0.002604159098704;
         0.000651041548404;
         0.000162760414818;
         0.000040690104138;
         0.000010172526041;
         0.000002543131511;
         0.000000635782878;
         0.000000158945721;
         0.000000039736428;
         0.000000009934108;
         0.000000002483528;
         0.000000000620884;
         0.000000000155218;
         0.000000000038807;
         0.000000000009701;
         0.000000000002425];

j = linspace(0,18,19);

h = 1./(2.^j);

log_h = log(h);
log_E = log(erros);

pol = polyfit(log_h, log_E, 1);

pol
```

```
33
34 fprintf('A ordem de convergencia estimada p e aproximadamente: %.15f\n', pol
       (1));
35
36 % Output:
37 % pol =
38 %    1.996252042520623   -1.823824224489114
39
40 % A ordem de convergencia estimada p e aproximadamente: 1.996252042520623
```

Verifica-se que o resultado obtido corresponde às expectativas, visto que

$$p_{18} = 1.996252042520623 \approx 2 \,,$$

que é a ordem de convergência da regra dos trapézios. Desta forma, obtém-se uma melhor aproximação do que para $n = 8$, visto que o resultado obtido para p com $n = 18$ está bastante mais próximo de 2.

Foram também construídos dois gráficos, respetivamente para $n = 8$ e $n = 18$, que permitem observar a regressão linear calculada de forma visual. O código utilizado foi o seguinte:

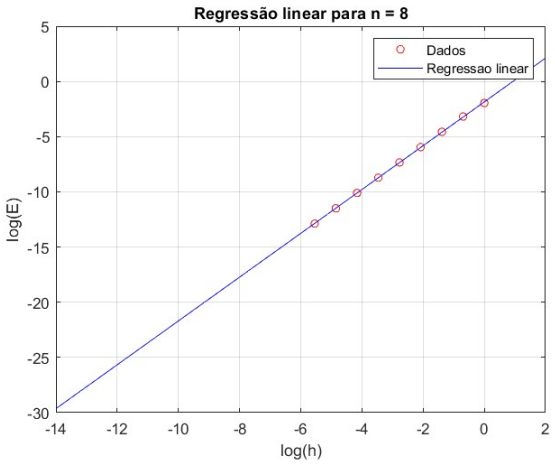
Para $n = 8$ (ficheiro `graf_b8`):

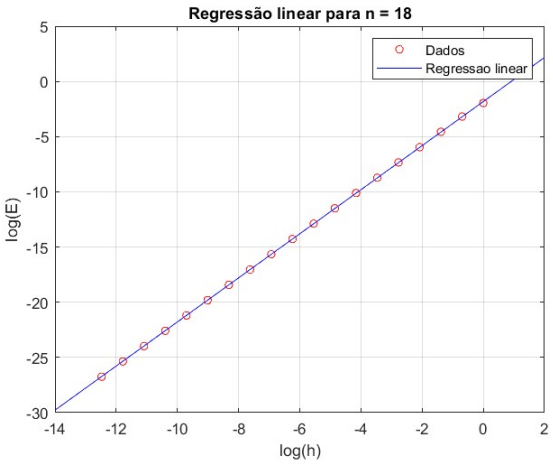
```
1  format long
2
3  % Dados do problema
4  erros = [0.141592653589793;
5           0.041592653589793;
6           0.010416183001558;
7           0.002604159098704;
8           0.000651041548404;
9           0.000162760414818;
10          0.000040690104138;
11          0.000010172526041;
12          0.000002543131511];
13
14 j = linspace(0,8,9);
15
16 h = 1./(2.^j);
17
18 log_h = log(h);
19 log_E = log(erros);
20
21  % Reta de regressao
22  reta = polyfit(log_h, log_E, 1);
23
24  % Representacao grafica dos dados e das funcoes de ajustamento
25  x = linspace(-14,2);
26  y = polyval(reta,x);
27  figure
28  plot(log_h,log_E,'ro','MarkerSize', 5)
29  hold on
30  plot(x,y,'b-')
31  grid on
32  hold off
33  xlabel('log(h)');
34  ylabel('log(E)');
35  title('Regressao linear para n = 8')
36  legend('Dados', 'Regressao linear');
```

Para $n = 18$ (ficheiro `graf_b18`):

```
format long

% Dados do problema
erros = [0.141592653589793;
         0.041592653589793;
         0.010416183001558;
         0.002604159098704;
         0.000651041548404;
         0.000162760414818;
         0.000040690104138;
         0.000010172526041;
         0.000002543131511;
         0.000000635782878;
         0.000000158945721;
         0.000000039736428;
         0.000000009934108;
         0.000000002483528;
         0.000000000620884;
         0.000000000155218;
         0.000000000038807;
         0.000000000009701;
         0.000000000002425];

j = linspace(0,18,19);

h = 1./(2.^j);

log_h = log(h);
log_E = log(erros);

 % Reta de regressao
 reta = polyfit(log_h, log_E, 1);

 % Representacao grafica dos dados e das funcoes de ajustamento
 x = linspace(-14,2);
 y = polyval(reta,x);
 figure
 plot(log_h,log_E,'ro','MarkerSize', 5)
 hold on
 plot(x,y,'b-')
 grid on
 hold off
 xlabel('log(h)');
 ylabel('log(E)');
 title('Regressao linear para n = 18')
 legend('Dados', 'Regressao linear');
```

Os gráficos elaborados são os seguintes:





Através da observação dos gráficos, pode-se verificar que o ajustameto dos pontos, recorrendo a regressão linear, é muito bem condicionado, oferecendo uma estimativa de p precisa e fiável.

Assim, conclui-se que os valores obtidos para as aproximações $R_{j,0}$, $j = 0, 1, 2, \ldots, 8, \ldots, 18, \ldots$, estão de acordo com a ordem de convergência da regra dos trapézios, ou seja, ordem 2.

5.

A função de contagem de números primos $\pi :]0, +\infty[\longrightarrow \mathbb{N}$ é definida por

$$\pi(x) = \#\{p \text{ primo} : p \leq x\}.$$

Para estimar alguns valores da função de contagem de números primos, $\pi(10^k)$, $k = 1, 2, \ldots, 20, \ldots$, começou-se por utilizar o método de Romberg para aproximar o integral definido pela função Li, dada por

$$\pi(x) \approx \int_2^x \frac{dt}{\ln t} =: \mathrm{Li}(x),$$

$$\text{sabendo que, para } x \text{ muito grande, temos que, } \lim_{x \to +\infty} \frac{\pi(x)}{\mathrm{Li}(x)} = 1.$$

Para aplicar a função `romberg`, construída no exercício 3., de modo a obter o resultado pretendido, considera-se o seguinte *input*:

- função $f = \frac{1}{\log(t)}$;
- $a = 2$;
- $b = 10^k$, $k = 1, 2, 3, \ldots, 20, \ldots$;
- $n = 20$ (escolhe-se um grande número de iterações para obter uma boa aproximação);
- $epsilon = 0.5 \times 10^{-16}$ (escolhido de acordo com o sistema de ponto flutuante do MATLAB).

O código utilizado (guardado no ficheiro `aplicacao5`) é o seguinte:

```
f = @(t) (1./log(t));
a = 2;
b = 10^(k);  % alterar o valor de k = 1,2,3,...,20,...
             % consoante o pretendido
n = 20;
epsilon = 0.5*10^(-16);

I = romberg(f,a,b,n,epsilon)

```

Os resultados obtidos encontram-se na tabela apresentada abaixo.

$10^k = n$	$\pi(10^k)$ aproximado pelo método de Romberg
10^1	$5.120435724669805 \times 10^0$
10^2	$2.908097780396214 \times 10^1$
10^3	$1.765644942100347 \times 10^2$
10^4	$1.245092052119270 \times 10^3$
10^5	$9.628763837300592 \times 10^3$
10^6	$7.862651016218019 \times 10^4$
10^7	$6.649187353946728 \times 10^5$
10^8	$5.762237032575835 \times 10^6$
10^9	$5.084958323606025 \times 10^7$
10^{10}	$4.550593541887892 \times 10^8$
10^{11}	$4.118104999549170 \times 10^9$
10^{12}	$3.760834318044662 \times 10^{10}$
10^{13}	$3.460696198373203 \times 10^{11}$
10^{14}	$3.204982123458776 \times 10^{12}$
10^{15}	$2.984497441416977 \times 10^{13}$
10^{16}	$2.792423918922910 \times 10^{14}$
10^{17}	$2.623597787423314 \times 10^{15}$
10^{18}	$2.474036171297500 \times 10^{16}$
10^{19}	$2.340617512565436 \times 10^{17}$
10^{20}	$2.220860524709310 \times 10^{18}$

Para além do método de Romberg, escreveu-se uma outra função no MATLAB que permite calcular os valores de $\pi(n)$ correspondentes, com $n = 10^k$, recorrendo à função `primes` do MATLAB, que retorna um vetor linha com todos os números primos menores ou iguais a um certo número n.
A função construída foi denominada por `pi_conta_primos` e, dado um certo $k = 1, 2, 3, \ldots, 20, \ldots$, devolve $\pi(10^k)$. O código correspondente é o seguinte:

```
function count = pi_conta_primos(k)

format long

    n = 10^k;
    count = length(primes(n));

end

% Input: k
% Output: pi(10^k)
```

Os resultados obtidos para $k = 1, 2, 3, \ldots, 10$ estão na tabela seguinte:

$10^k = n$	$\pi(n)$ calculado com a função `pi_conta_primos`
10^1	4
10^2	25
10^3	168
10^4	1229
10^5	9592
10^6	78498
10^7	664579
10^8	5761455
10^9	50847534
10^{10}	455052511

Verificou-se que o programa demora algum tempo a correr para $k = 9$ (12 segundos) e $k = 10$ (1 minuto e 25 segundos). Para além disso, para valores de k superiores a 10 o MATLAB não devolve um resultado.

Este "erro" no programa deve-se ao facto de a função `primes` estar a criar um vetor que excede a memória máxima permitida no MATLAB. Neste caso, a memória máxima permitida é 15.4GB e o vetor que se pretende criar ocupa 46.6GB, com $k = 11$, por exemplo.

De modo a arranjar valores que possam ser comparados aos resultados obtidos através do método de Romberg, recorreu-se à função Li, que é uma boa aproximação da função de contagem de números primos, para valores de n muito grandes. Portanto, calcularam-se os valores de $\text{Li}(10^k)$, com $k = 11, 12, \ldots, 20$.

De seguida, mostra-se o código da função `li_funcao`, construída no MATLAB, que permitiu calcular estes valores e a tabela correspondente com os resultados observados.

```
function pi = li_funcao(k)

format long

    n = 10^k;
    pi = integral(@(t) 1./log(t), 2, n);
end

% Input: k
% Output: Li(10^k) = integral(@(t) 1./log(t), 2, 10^k)
```

$10^k = n$	$\pi(n)$ calculado com a função `li_funcao`
10^{11}	$4.118066445404169 \times 10^9$
10^{12}	$3.760795140173498 \times 10^{10}$
10^{13}	$3.460656541215641 \times 10^{11}$
10^{14}	$3.204942129000016 \times 10^{12}$
10^{15}	$2.984457197072858 \times 10^{13}$
10^{16}	$2.792383482191696 \times 10^{14}$
10^{17}	$2.623557198084439 \times 10^{15}$
10^{18}	$2.473995457989005 \times 10^{16}$
10^{19}	$2.340576696577419 \times 10^{17}$
10^{20}	$2.220819622293545 \times 10^{18}$

Para facilitar a comparação dos resultados, podem-se calcular os erros relativos percentuais associados aos valores conseguidos anteriormente. O erro relativo será calculado da seguinte forma:

$$\left|\delta_{\tilde{\pi}(n)}\right| = \frac{|\pi(n) - \tilde{\pi}(n)|}{|\pi(n)|} \times 100\,\% \ ,$$

tendo em conta que:

- $\tilde{\pi}(n)$ é a aproximação da função de contagem de números primos pelo método de Romberg;
- $\pi(n) = \pi(10^k) := \begin{cases} \texttt{pi\_conta\_primos(k)}, & \text{se } k = 1, 2, 3, \ldots, 10 \\ \texttt{li\_funcao(k)}, & \text{se } k = 11, 12, 13, \ldots, 20 \end{cases}$.

Abaixo apresenta-se a tabela efetuada com base na informação acima.

$10^k = n$	$\tilde{\pi}(n)$ aproximado pelo método de Romberg	$\pi(n)$ calculado com a função `pi_conta_primos`	Erro relativo (%)
10^1	$5.120435724669805 \times 10^0$	4	28.010893116745116 %
10^2	$2.908097780396214 \times 10^1$	25	16.323911215848568 %
10^3	$1.765644942100347 \times 10^2$	168	5.097913220258747 %
10^4	$1.245092052119270 \times 10^3$	1229	1.309361441763238 %
10^5	$9.628763837300592 \times 10^3$	9592	0.383276035243870 %
10^6	$7.862651016218019 \times 10^4$	78498	0.163711383959066 %
10^7	$6.649187353946728 \times 10^5$	664579	0.051120392710700 %
10^8	$5.762237032575835 \times 10^6$	5761455	0.013573525712437 %
10^9	$5.084958323606025 \times 10^7$	50847534	0.004030158198521 %
10^{10}	$4.550593541887892 \times 10^8$	455052511	0.001503823981578 %
		$\pi(n)$ calculado com a função `li_funcao`	
10^{11}	$4.118104999549170 \times 10^9$	$4.118066445404169 \times 10^9$	0.000936219595096 %
10^{12}	$3.760834318044662 \times 10^{10}$	$3.760795140173498 \times 10^{10}$	0.001041744357362 %
10^{13}	$3.460696198373203 \times 10^{11}$	$3.460656541215641 \times 10^{11}$	0.001145943178410 %
10^{14}	$3.204982123458776 \times 10^{12}$	$3.204942129000016 \times 10^{12}$	0.001247899561052 %
10^{15}	$2.984497441416977 \times 10^{13}$	$2.984457197072858 \times 10^{13}$	0.001348464443002 %
10^{16}	$2.792423918922910 \times 10^{14}$	$2.792383482191696 \times 10^{14}$	0.001448108093747 %
10^{17}	$2.623597787423314 \times 10^{15}$	$2.623557198084439 \times 10^{15}$	0.001547110880778 %
10^{18}	$2.474036171297500 \times 10^{16}$	$2.473995457989005 \times 10^{16}$	0.001645650090574 %
10^{19}	$2.340617512565436 \times 10^{17}$	$2.340576696577419 \times 10^{17}$	0.001743843219350 %
10^{20}	$2.220860524709310 \times 10^{18}$	$2.220819622293545 \times 10^{18}$	0.001841771180093 %

Ao analisar a tabela apenas para os valores com $k = 1, 2, 3, \ldots, 10$, observa-se que o erro relativo percentual é consideravelmente grande para valores de n mais baixos. No entanto, à medida que o n vai aumentando, os erros relativos tornam-se cada vez mais pequenos, de uma forma relativamente rápida.

Portanto, pode-se afirmar que o método de Romberg pode fornecer uma boa aproximação da função $\pi(x)$, mas apenas para valores de x muito grandes.

Ao avaliar os valores correspondentes a $k = 11, 12, 13, \ldots, 20$, verifica-se que os erros relativos são igualmente pequenos, o que confirma, mais uma vez, que a utilização do método de Romberg pode dar uma boa aproximação do valor de $\pi(x)$.

No entanto, pode-se também observar que os erros relativos a partir de $k = 11$ vão aumentando muito ligeiramente. Esta observação pode ter várias explicações, como por exemplo:

- Todos os cálculos necessários foram efetuados com recurso ao MATLAB, que está associado a um sistema de ponto flutuante característico deste software. Portanto, é importante considerar possíveis erros de arredondamento que possam ter ocorrido ao longo dos cálculos efetuados, e que possam afetar o resultado final.
- Há que lembrar que o método de Romberg é calculado recursivamente, por isso os próprios cálculos efetuados pela função `romberg`, no MATLAB, podem ter sofrido pequenas oscilações devido a arredondamentos efetuados pelo próprio programa, o que aumenta a imprecisão dos resultados.
- A função Li trata-se apenas de uma aproximação da função de contagem de números primos, π, por isso pequenas oscilações da função Li em torno do valor real, dado pela função π, podem afetar os erros relativos calculados, sendo que estes pretendem avaliar a eficácia do método de Romberg relativamente à função π, e não em relação à sua função aproximada Li.

Assim, conclui-se que o método de Romberg é eficiente e fiável para estimar a quantidade de números primos até um certo valor x, para x suficientemente grande. No entanto, para valores de x mais pequenos, a contagem direta com o auxílio da função `primes` é mais adequada.

Grupo II

1.

O sistema de equações diferenciais ordinárias, tal como apresentado no enunciado, é dado por:

$$\begin{cases} (m_1+m_2)\ell_1\alpha_1''(t) \;+\; m_2\ell_2\alpha_2'(t)\cos(\alpha_1-\alpha_2) + \\ \qquad + m_2\ell_2(\alpha_2'(t))^2\sin(\alpha_1-\alpha_2) \;+\; (m_1+m_2)g\sin(\alpha_1) = 0 \\ \\ \ell_2\alpha_2''(t) \;+\; \ell_1\alpha_1''(t)\cos(\alpha_1-\alpha_2) - \\ \qquad - \ell_1(\alpha_1'(t))^2\sin(\alpha_1-\alpha_2) \;+\; g\sin(\alpha_2) = 0 \end{cases} \tag{1}$$

com condições iniciais:

$$\alpha_1(0) = a_1, \quad \alpha_1'(0) = b_1, \quad \alpha_2(0) = a_2, \quad \alpha_2'(0) = b_2.$$

Neste sistema, ℓ_1 e ℓ_2 são os comprimentos dos braços do pêndulo, m_1 e m_2 são as massas das duas esferas, $\alpha_1(t)$ e $\alpha_2(t)$ são os ângulos de cada pêndulo em relação à vertical em função do tempo e $g = 9.8\,\mathrm{m/s^2}$ representa a aceleração gravítica.
Para usar métodos de integração numérica como o `ode45` em MATLAB, é conveniente reescrever (1) como um sistema de primeira ordem. Para tal, define-se quatro novas variáveis:

$$x_1(t) := \alpha_1(t), \quad x_2(t) := \alpha_1'(t), \quad x_3(t) := \alpha_2(t), \quad x_4(t) := \alpha_2'(t).$$

Isto implica as igualdades:

$$x_1'(t) = \alpha_1'(t) = x_2(t), \quad x_3'(t) = \alpha_2'(t) = x_4(t).$$

Para obter $x_2'(t)$ e $x_4'(t)$, isola-se $\alpha_1''(t)$ e $\alpha_2''(t)$ no sistema original.

Reescrita ($\theta := \alpha_1 - \alpha_2$):

$$\begin{cases} (m_1+m_2)\ell_1\alpha_1'' + m_2\ell_2\alpha_2''\cos(\theta) = -m_2\ell_2(\alpha_2')^2\sin(\theta) - (m_1+m_2)g\sin(\alpha_1) \\ \\ \ell_1\alpha_1''\cos(\theta) + \ell_2\alpha_2'' = \ell_1(\alpha_1')^2\sin(\theta) - g\sin(\alpha_2). \end{cases}$$

Forma matricial:

$$\underbrace{\begin{bmatrix} (m_1+m_2)\ell_1 & m_2\ell_2\cos(\theta) \\ \ell_1\cos(\theta) & \ell_2 \end{bmatrix}}_{M} \begin{bmatrix} \alpha_1'' \\ \alpha_2'' \end{bmatrix} = \underbrace{\begin{bmatrix} -m_2\ell_2(\alpha_2')^2\sin(\theta) - (m_1+m_2)g\sin(\alpha_1) \\ \ell_1(\alpha_1')^2\sin(\theta) - g\sin(\alpha_2) \end{bmatrix}}_{\mathbf{c}}$$

Determinante da matriz:

$$\Delta = \det(M) = (m_1+m_2)\ell_1\ell_2 - m_2\ell_1\ell_2\cos^2(\theta) = \ell_1\ell_2\Big[(m_1+m_2) - m_2\cos^2(\theta)\Big]$$

Matriz inversa:

$$M^{-1} = \frac{1}{\Delta}\begin{bmatrix} \ell_2 & -m_2\ell_2\cos(\theta) \\ -\ell_1\cos(\theta) & (m_1+m_2)\ell_1 \end{bmatrix} = \frac{1}{\ell_1\ell_2\Big[(m_1+m_2) - m_2\cos^2(\theta)\Big]}\begin{bmatrix} \ell_2 & -m_2\ell_2\cos(\theta) \\ -\ell_1\cos(\theta) & (m_1+m_2)\ell_1 \end{bmatrix}$$

Isolar $\alpha_1''(t)$ e $\alpha_2''(t)$:

$$\begin{bmatrix}\alpha_1''\\ \alpha_2''\end{bmatrix} = \underbrace{\frac{1}{\ell_1\ell_2\big[(m_1+m_2)-m_2\cos^2(\theta)\big]}\begin{bmatrix}\ell_2 & -m_2\ell_2\cos(\theta)\\ -\ell_1\cos(\theta) & (m_1+m_2)\ell_1\end{bmatrix}}_{M^{-1}}\underbrace{\begin{bmatrix}-m_2\ell_2(\alpha_2')^2\sin(\theta)-(m_1+m_2)g\sin(\alpha_1)\\ \ell_1(\alpha_1')^2\sin(\theta)-g\sin(\alpha_2)\end{bmatrix}}_{\mathbf{c}} =$$

$$= \frac{1}{\ell_1\ell_2\big[(m_1+m_2)-m_2\cos^2(\theta)\big]}\begin{bmatrix}\ell_2\left(-m_2\ell_2(\alpha_2')^2\sin(\theta)-(m_1+m_2)g\sin(\alpha_1)\right)-m_2\ell_2\cos(\theta)\left(\ell_1(\alpha_1')^2\sin(\theta)-g\sin(\alpha_2)\right)\\ -\ell_1\cos(\theta)\left(-m_2\ell_2(\alpha_2')^2\sin(\theta)-(m_1+m_2)g\sin(\alpha_1)\right)+(m_1+m_2)\ell_1\left(\ell_1(\alpha_1')^2\sin(\theta)-g\sin(\alpha_2)\right)\end{bmatrix}$$

Sistema de equações diferenciais de primeira ordem:

$$\begin{cases} x_1'(t)\big[:=\alpha_1'(t)\big] = x_2(t),\\ x_2'(t)\big[:=\alpha_1''(t)\big] = \frac{\ell_2\left(-m_2\ell_2(\alpha_2')^2\sin(\theta)-(m_1+m_2)g\sin(\alpha_1)\right)-m_2\ell_2\cos(\theta)\left(\ell_1(\alpha_1')^2\sin(\theta)-g\sin(\alpha_2)\right)}{\ell_1\ell_2\big[(m_1+m_2)-m_2\cos^2(\theta)\big]}\\ x_3'(t)\big[:=\alpha_2'(t)\big] = x_4(t),\\ x_4'(t)\big[:=\alpha_2''(t)\big] = \frac{-\ell_1\cos(\theta)\left(-m_2\ell_2(\alpha_2')^2\sin(\theta)-(m_1+m_2)g\sin(\alpha_1)\right)+(m_1+m_2)\ell_1\left(\ell_1(\alpha_1')^2\sin(\theta)-g\sin(\alpha_2)\right)}{\ell_1\ell_2\big[(m_1+m_2)-m_2\cos^2(\theta)\big]}\end{cases}$$

2.

A seguinte função tem os seguintes argumentos:

- T, que define o instante final;
- m_1, m_2, ℓ_1, ℓ_2, que definem as massas das duas esferas e os comprimentos dos dois braços do duplo pêndulo;
- a_1, b_1, a_2, b_2, que definem as condições iniciais do problema.

O seu output são os gráficos das trajetórias dos ângulos de rotação e das velocidades angulares.

```
function duploPendulo(T, m1, m2, l1, l2, a1, b1, a2, b2)
    g = 9.81;
    X0 = [a1; b1; a2; b2];  % Vetor de condições iniciais: [α1(0); α1'(0); α
   2(0); α2'(0)]

    [t, X] = ode45(@(t,X) sistemaEDO(X, m1, m2, l1, l2, g), [0, T], X0);

    alpha1 = X(:, 1);   % α1(t)
    dalpha1 = X(:, 2);  % α1'(t)
    alpha2 = X(:, 3);   % α2(t)
    dalpha2 = X(:, 4);  % α2'(t)

    % Plot dos ângulos α1(t) e α2(t) em função do tempo t
    figure;
    subplot(2, 1, 1);
    plot(t, alpha1, "b-", t, alpha2, "r-");
    xlabel("t(s)"); ylabel("\alpha_1(t), \alpha_2(t) (rad)");
    title("Ângulos \alpha_1(t) e \alpha_2(t)");
    legend("\alpha_1(t)", "\alpha_2(t)");
    grid on;
```

```
20
21      % Plot das velocidades angulares α1'(t) e α2'(t) em função do tempo t
22      subplot(2, 1, 2);
23      plot(t, dalpha1, "b--", t, dalpha2, "r--");
24      xlabel("t(s)"); ylabel("\alpha'_1(t), \alpha'_2(t) (rad/s)");
25      title("Velocidades \alpha'_1(t) e \alpha'_2(t)");
26      legend("\alpha'_1(t)", "\alpha'_2(t)");
27      grid on;
28  end
```

Esta, por sua vez, chama a função `sistemaEDO`, que resolve o sistema de equações diferenciais com as condições iniciais e os parâmetros dados.

```
1   function dXdt = sistemaEDO(X, m1, m2, l1, l2, g)
2       alpha1 = X(1);          % Ângulo α1
3       dalpha1 = X(2);         % Velocidade α1'
4       alpha2 = X(3);          % Ângulo α2
5       dalpha2 = X(4);         % Velocidade α2'
6
7       theta = alpha1 - alpha2;
8
9       % Valores utilizados no cálculo matricial
10      detM = l1 * l2 * ( (m1+m2) - m2*cos(theta)^2 );
11      A = l2;
12      B = -m2 * l2 * cos(theta);
13      C = -l1 * cos(theta);
14      D = (m1 + m2) * l1;
15      c1 = -m2 * l2 * dalpha2^2 * sin(theta) - (m1 + m2) * g * sin(alpha1);
16      c2 = l1 * dalpha1^2 * sin(theta) - g * sin(alpha2);
17
18      % Expressão obtida da 1.   equação do sistema isolada para α1''
19      d2alpha1 = (A*c1 + B*c2) / detM;
20
21      % Expressão obtida da 2.   equação do sistema isolada para α2''
22      d2alpha2 = (C*c1 + D*c2) / detM;
23
24      % Devolve o vetor de derivadas [α1'; α1''; α2'; α2'']:
25      dXdt = [dalpha1; d2alpha1; dalpha2; d2alpha2];
26  end
```

3.

```
1   % Script principal para simular o movimento do duplo pêndulo
2   clc; clear;
3
4   g = 9.81;
5
6   % 3. (a)
7   [T, m1, m2, l1, l2, a1, b1, a2, b2] = deal(10, 1, 1, 2, 2, pi/4, 0, 0, 0);
8
9
10  % 3. (b)
11  % [T, m1, m2, l1, l2, a1, b1, a2, b2] = deal(100, 4, 2, 3, 4, pi/2, pi/6, pi
        /4, pi/12);
12
13
14  duploPendulo(T, m1, m2, l1, l2, a1, b1, a2, b2)
15  % animarDuploPendulo(T, m1, m2, l1, l2, a1, b1, a2, b2)
```

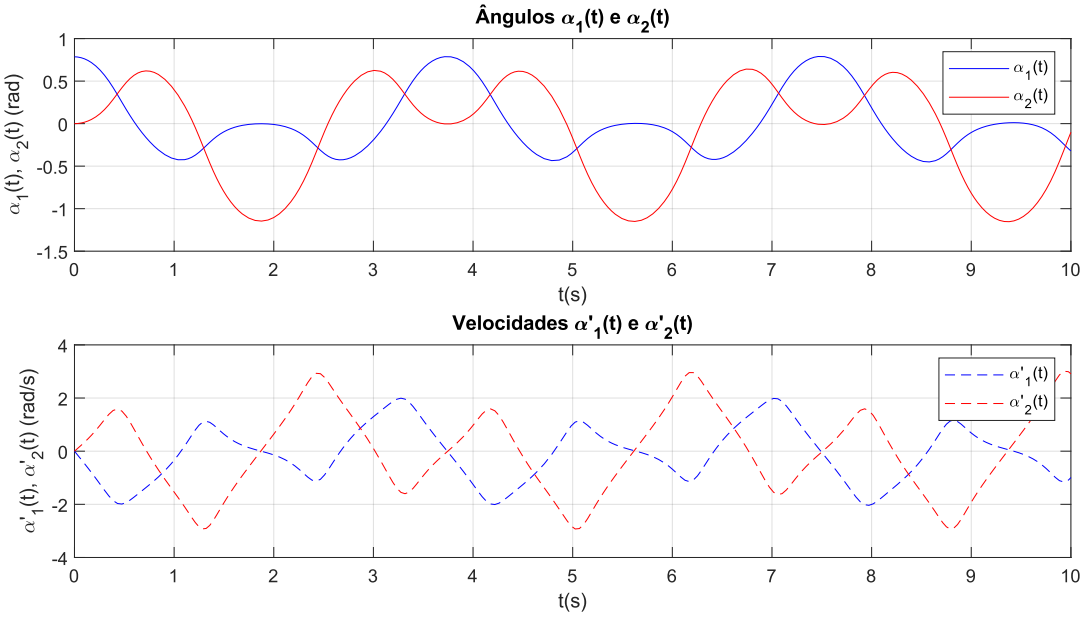
Utiliza-se o script acima para definir os parâmetros e visualizar os dois gráficos. Para a alínea (a), os parâmetros são os definidos na linha 7 e, para a alínea (b), os parâmetros são os definidos na linha 11, sendo que uma destas linhas deve ficar comentada consoante a alínea.

(a)

Os parâmetros a considerar são os seguintes:

$$\begin{cases} m_1 = m_2 = 1 \\ \ell_1 = \ell_2 = 2 \\ a_1 = \frac{\pi}{4} \\ a_2 = b_1 = b_2 = 0 \\ T = 10 \end{cases}$$

Obteve-se o seguinte gráfico:



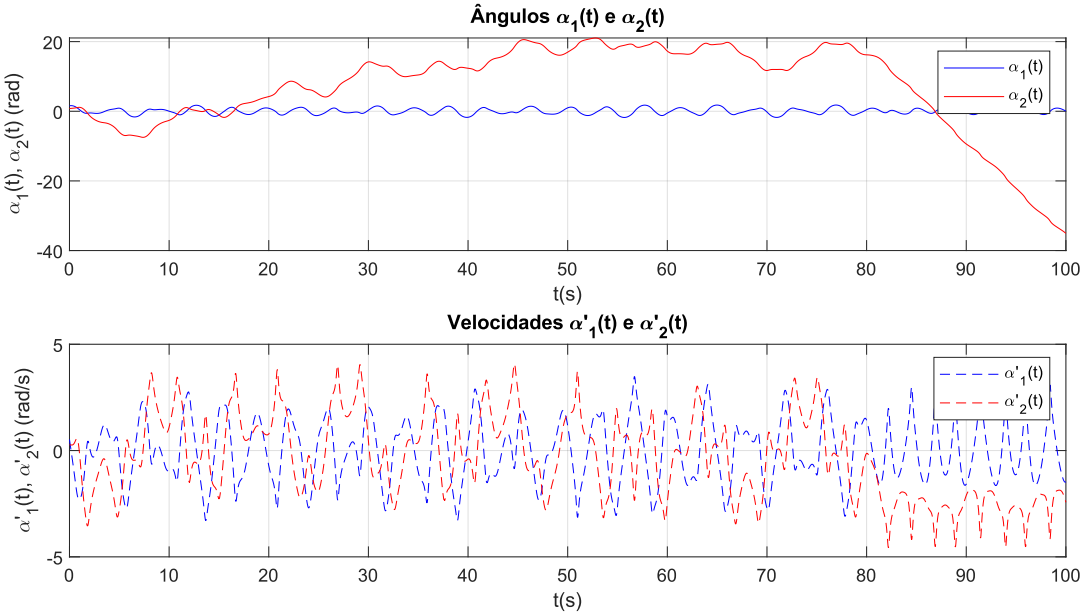
Como se pode observar, os gráficos são periódicos.

(b)

Escolhem-se os seguintes parâmetros:

$$\begin{cases} m_1 = 4, \quad m_2 = 2 \\ \ell_1 = 3, \quad \ell_2 = 4 \\ a_1 = \frac{\pi}{2}, \quad a_2 = \frac{\pi}{6}, \quad b_1 = \frac{\pi}{4}, \quad b_2 = \frac{\pi}{12} \\ T = 100 \end{cases}$$

Obteve-se o seguinte gráfico:



Este gráfico já não é periódico.

Animação do sistema de dois pêndulos acoplados

Decidiu-se fazer, ainda, uma simulação visual do comportamento dos pêndulos. Os gráficos obtidos são dinâmicos, por isso, quando são visualizados no MATLAB, é possível observar uma representação dinâmica do percurso percorrido pelo pêndulo.
O código utilizado para a simulação apresenta-se a seguir, e, para visualizar a animação, utiliza-se o mesmo script, trocando o comentário da linha 16 para a 15:

```
1  function animarDuploPendulo(T, m1, m2, l1, l2, a1, b1, a2, b2)
2      g = 9.81;
3      X0 = [a1; b1; a2; b2];
4
5      [t, X] = ode45(@(t, X) sistemaEDO(X, m1, m2, l1, l2, g), [0 T], X0);
6
7      alpha1 = X(:,1);
8      alpha2 = X(:,3);
9
10     % Coordenadas das massas
11     x1 =  l1 * sin(alpha1);
12     y1 = -l1 * cos(alpha1);
13     x2 = x1 + l2 * sin(alpha2);
14     y2 = y1 - l2 * cos(alpha2);
15
16     figure;
17     axis equal;
18     axis([-l1-l2, l1+l2, -l1-l2, l1+l2]);
19     grid on;
20     title('Animação do sistema de dois pêndulos acoplados');
21     xlabel('x(m)');
22     ylabel('y(m)');
23     hold on;
24
```

```
    % Animação
    for i = 1:2:length(t) % Ajustar o passo para controlar a velocidade
        cla;
        plot([0 x1(i)], [0 y1(i)], 'b-', 'LineWidth', 2);
        plot([x1(i) x2(i)], [y1(i) y2(i)], 'r-', 'LineWidth', 2);
        plot(x1(i), y1(i), 'bo', 'MarkerSize', 10, 'MarkerFaceColor', 'b');
        plot(x2(i), y2(i), 'ro', 'MarkerSize', 10, 'MarkerFaceColor', 'r');
        plot(0, 0, 'ko', 'MarkerSize', 5, 'MarkerFaceColor', 'k');
        axis equal;
        axis([-l1-l2, l1+l2, -l1-l2, l1+l2]);
        drawnow;
    end
end
```

Apresentam-se três imagens que mostram a evolução inicial do pêndulo da alínea (a):