Relatório − VERSÃO 35

Afonso Alemão, nº 96135 João Santos, nº 96237 Marcelo Santos, nº 86289 Rui Daniel, nº 96317

22 de Dezembro de 2020

Percentagem da contribuição para a realização do trabalho de cada elemento do grupo:

- Afonso Alemão (25%): Pergunta $G1-2b,\ G3-1,\ G3-2$
- João Santos (25%): Pergunta $G1-2c,\ G3-2c$
- Marcelo Santos (25%): Pergunta $G1-2a,\ G3-2a, 2b$
- Rui Daniel (25%): Pergunta $G1-1,\ G1-2b,\ 2c,\ G2,\ G3-2c$

  (As respostas de todas as perguntas provieram dum esforço coletivo, no qual todos os elementos do grupo deram a sua contribuição)

Grupo I

1. • Código da função **newtonquasi**:

```
% (Código em MatLab: Ficheiro newtonquasi.m)
function [z, fx, iter] = newtonquasi(f, x0, Delta, TolX, MaxIter)
  % caso o utilizador insira menos de 4 argumentos, funcao retorna sem resolver o problema
  if (nargin < 4 || nargin > 5)
      fprintf('Incorret input arguments\n');
      return;
  end
  % caso utilizador insira 4 argumentos, definimos que o numero maximo de iteradas seria 1000
  if nargin < 5
      MaxIter = 1000;
  end
  TolFun = eps;
  iter = zeros(1, 1); % inicializacao do vetor de iteradas
  iter(1) = x0; % aproximacao inicial
  fx = feval(f, x0); % fx = f(x(k))
  for k = 1 : MaxIter
    dx = - (fx * Delta) / (feval(f, iter(k) + Delta) - fx);
    iter(k + 1) = iter(k) + dx; % aplicacao do metodo para calculo de x(k + 1)
    fx = feval(f, iter(k + 1));
    criterio = (iter(k + 1) - iter(k)) / iter(k + 1); % criterio de paragem
    if abs(fx) < TolFun || abs(criterio) < TolX, break; end
  end
  z = iter(k + 1); % aproximacao final
  if k == MaxIter
    fprintf('The best in %d iterations\n', MaxIter)
  end
end
```

• Código para obter os valores da tabela e o número de iteradas:

```
% (Código em MatLab: Ficheiro scriptnewtonquasi.m)
f = @(x) (x ^ 3 - x ^ 2 - x + 1);
z = zeros(2, 1);
```

```
fx = zeros(2, 1);

[z(1), fx(1), iter1] = newtonquasi(f, -0.2, 1e-5, 1e-2, 1000);
[z(2), fx(2), iter2] = newtonquasi(f, -0.3, 1e-5, 1e-2, 1000);
fprintf('x0 = -0.2\nNumber of iterations = %d\n', size(iter1, 2));
disp(iter1);
fprintf('x0 = -0.3\nNumber of iterations = %d\n', size(iter2, 2));
disp(iter2);
```

x_n	x_n
-0.2000000000000000	-0.3000000000000000
2.199920003257039	8.798670201365423
1.694690306819625	6.009340485204284
1.387008629219663	4.164056127899828
1.208017409671792	2.953033630479767
1.108690345536816	2.169962618573130
1.055713188505415	1.676118425447672
1.028231592389497	1.375977672202824
1.014215892049550	1.201774916681658
1.007135457584903	1.105310385106001
–	1.053942616976839
–	1.027323454488433
–	1.013755707316576
–	1.006903780588617

x_0	-0.2	-0.3
Número de iteradas	10	14

Como se pode verificar pelas tabelas acima, ao utilizar como aproximação inicial $x_0 = -0.2$, foram necessárias 10 iterações para atingir a precisão desejada de $TolX = 10^{-2}$, enquanto que para $x_0 = -0.3$, foram necessárias 14 iteradas para atingir esta precisão.

2. (a)

$$w(r) = b\, e^{-kr}(k\sin(ar) + \cos(ar)) \tag{1}$$

$$f(r) = w(r) - 0.1 \;\Leftrightarrow f(r) = [1.5\, e^{-r}(\sin(2r) + \cos(2r))] - 0.1 \tag{2}$$

Fazendo uma representação gráfica da função (1) com os parâmetros $a = 2$, $b = 1.5$, $k = 1$ dados no enunciado, e da função (2), verifica-se que apenas será possível obter um intervalo em que $w(r) \geq 0.1 \iff f(r) \geq 0$, para $r > 0$, como a Figura 1 abaixo mostra. Devido a isto, na tabela de representação de intervalos, apenas um intervalo será colocado. A função $f(r)$ apenas tem 1 raiz para r > 0. Graficamente, podemos verificar que esta se encontra em $r \in [0.5; 1.5]$. Logo é neste intervalo que iremos executar o método da bisseção de modo a encontrarmos esta raiz.

Como $f(0) > 0$, o intervalo que pretendemos obter é $[0; c]$, $f(c) = 0$ e $c > 0$.

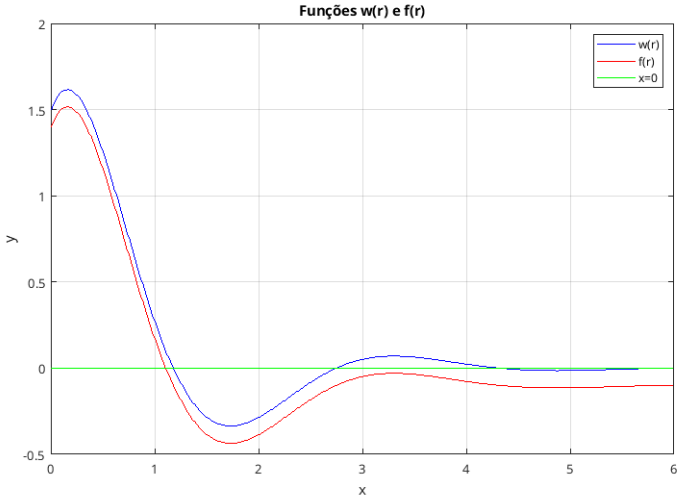


Figura 1: Gráficos de $w(r)$ e $f(r)$

- Código para criar o gráfico da Figura (1):

```
(Código em MATLAB: grafico.m)
w = @(r) 1.5 .* exp(- 1 .* r) .* (1 .* sin(2 .* r) + cos(2 .* r));
f = @(r) w(r) - 0.1;
i = 0 : 0.01 : 6;

plot(i, w(i), 'b', i, f(i), 'r', i, 0 .* i, 'g');
title('Funções w(r) e f(r)')
xlabel('x');
ylabel('y');
legend('w(r)', 'f(r)', 'x = 0');
grid;
```

- Código para obter os extremos dos intervalos:

```
% Utilizando com pequenas alterações a função bissecao.m da biblioteca de programas
% da professora Isabel Reis dos Santos (código na próxima página):
% (Código em MatLab: Ficheiro grupo1p2a.m)
w = @(r) 1.5 .* exp(- 1 .* r) .* (1 .* sin(2 .* r) + cos(2 .* r));
f = @(r) w(r) - 0.1;

% É chamada a função bissecao.m, dado na biblioteca de funções.
% Apenas se faz uma chamada pois só será necessário calcular uma raiz
% devido aos parâmetros dados: observamos graficamente que para r > 0
% só existe um único r tal que f(r) = 0
[z, f_z, n, iters, e_iters] = bissecao(f, 0.5, 1.5, 1e-20, 1e-15);
fprintf("Intervalo = [%d; %.15f]\n|e_n| = %e\n", 0, z, e_iters(end))
```

Intervalos
$[0.000000000000000 \times 10^0;\ 1.106577458467634 \times 10^0]$

(b) Como a função (3) pedida é diferente da função (2) utilizada para o cálculo de intervalos na alínea *2(a)*, não poderemos utilizar o valor exato da primeira raiz de (2). Logo iremos aplicar novamente o método da bisseção a esta nova função (3).
Pelo gráfico da Figura 2, podemos perceber que essa raiz está no intervalo $[0.5\,;\,1.4]$, por isso para obtermos o valor exato aplicamos para este o método da bisseção.

$$f(x) = 1.5\, e^{-x}(\sin(2x) + \cos(2x)) - 0.01 \tag{3}$$

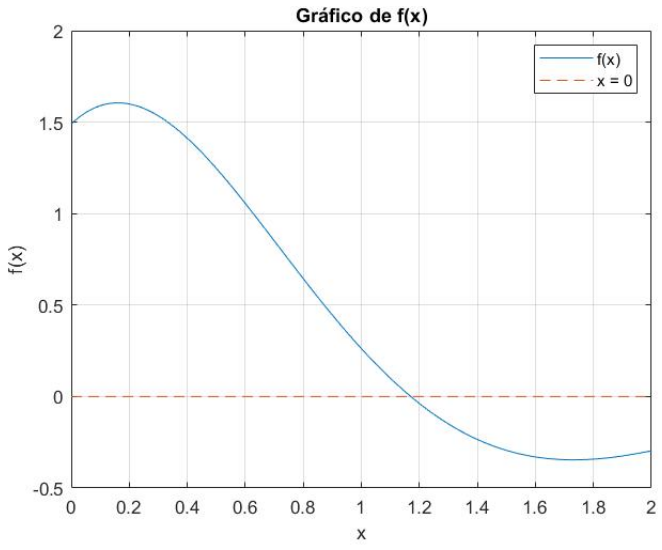


Figura 2: Gráfico da função (3)

- Código para obter os vectores **x** e **y**, ajuste dos pontos e gráfico da Figura 2:

```
% Utilizando com pequenas alterações a função  bissecao.m da biblioteca de programas
da professora Isabel Reis dos Santos:

% (Código em MatLab: bissecao.m)
function [root, fc, k, c, e] = bissecao(f, a, b, eps_abs, eps_step, iter_max)
   % caso os 3 parametros finais de entrada nao sejam inseridos pelo
   % utilizador, é lhes atribuido um significado
   % Sem os 3 parametros a, b, f, avisa o utilizador
   % da incorreta chamada à função
   if nargin < 3
       fprintf("Not enough input arguments")
       return
   end
   if nargin < 6
       iter_max = 1e3; end
   if nargin < 5
       eps_step = 1e-6; end
   if nargin < 4
       eps_abs = 1e-6; end
   root = [];
   k = 0;
   c = a;
   e = b - a;
   fa = feval(f, a);
   fc = fa;
   fb = feval(f, b);
   if ( fa * fb < 0 )
      while abs(b - a) >= eps_step
         k = k + 1;
         if k > iter_max
            disp('root not found with the desired accuracy')
            fprintf('Maximum number of iterations exceeded: %d\n', k)
            return
         end
         c(k) = (a + b) / 2;
         e(k) = (b - a) / 2;
         fc = feval(f, c(k));
         if abs(fc) < eps_abs
            break;
         elseif fa * fc < 0
            b = c(k);
            fb = fc;
         else
            a = c(k);
            fa = fc;
         end
      end
   else
      disp('The function must have opposite signs at the extreme points');
   end
   root = c(length(c));
end
```

```
% (Código em MatLab: Ficheiro grupo1p2b.m)
f = @(x) (1.5 * exp(- x) * (sin(2 * x)+ cos(2 * x)) - 0.01);

% gerador de grafico de f de modo a observar o intervalo a que a primeira
% raiz de f pertence
t = 0 : 0.01 : 2;
x1 = zeros(1, 1);
for i = 1 : length(t)
    x1(i) = 1.5 * exp(- t(i)) * (sin(2 * t(i))+ cos(2 * t(i))) - 0.01 ;
end
figure(1)
plot(t, x1, '-', t, t * 0, '--')
title('Gráfico de f(x)')
legend('f(x)', 'x = 0', 'Location', 'northeast');
grid;
xlabel('x')
ylabel('f(x)')

% graficamente observamos que a raiz está entre 0.5 e 1.4 (é aproximadamente 1.17)
[root, fc, k, c, errobissecao] = bissecao(f, 0.5, 1.4, 1e-18, 1e-15, 1000);
[z, fx, iter] = newtonquasi(f, 1.0, 1e-4, 1e-12, 10000);
% calculo dos modulos dos erros em cada iterada, comparativamente ao valor
% exato calculado no metodo da bissecao
e = iter;
e = abs(root - e);
% inicializacoes
x = zeros(1, 1);
y = zeros(1, 1);
% elementos de x: xi = ln |e(i)|
for i = 1 : length(e) - 1
    x(i) = log (e(i));
end
% elementos de y: yi = ln |e(i + 1)|
for i = 2 : length(e)
    y(i - 1) = log (e(i));
end

% geracao de grafico com reta de ajuste aos pontos (x, y)
[p, s] = polyfit(x, y, 1);
figure(2)
plot(x, y, 'o', x, polyval(p, x), '--');
title('Reta de ajuste de ln|e(k)| em funcao de ln|e(k+1)|')
xlabel('x = ln |e(k)|')
ylabel('y = ln |e(k+1)|')
legend('Pontos (x, y)', 'y(x)', 'Location', 'southeast');
grid;
fprintf("Coeficientes da reta de ajuste obtida\n")
for i = 1 : length(p)
    fprintf("%+.15e\n", p(i));
end
% calculo dos coeficientes assimptoticos de convergencia ao longo das
% iteradas
coeficienteassimptotico = zeros(1, 1);
for i = 1 : length(x) - 1
    coeficienteassimptotico(i) = e(i + 1) / e(i);
end
fprintf("\nCoeficientes Assimptoticos de Convergencia ao longo das iteradas:\n")
for i = 1 : length(coeficienteassimptotico)
```

```
    fprintf("%+.15e\n", coeficienteassimptotico(i));
end
% se elementos deste vetor tenderem para um numero real, ordem de
% convergencia é 1 e o coeficiente é o valor deste limite
```

A reta obtida é $y = Ax + B$ com $A = 1.007087346953520$ e $B = -5.606299758321017$

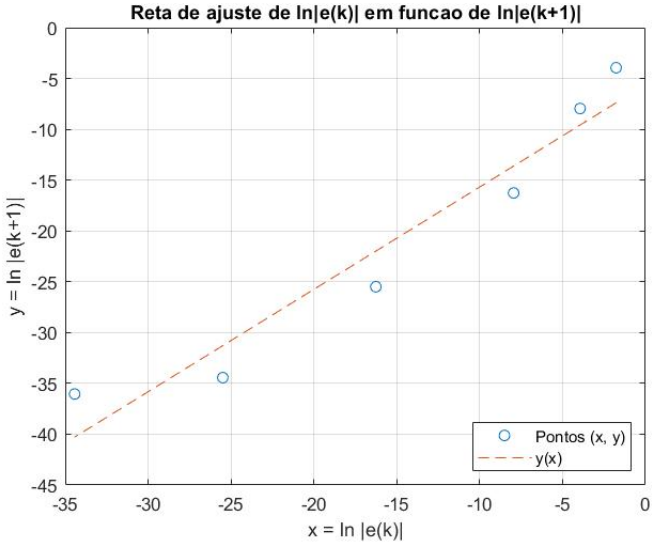


Figura 3: Gráfico da reta de ajuste aos pontos (x_i, y_i).

$$lim_{n\to\infty} \frac{|g^{(p)}|}{p!} = K_\infty$$

$$|e_{n+1}| = c\,|e_n|$$

$$|e_{n+1}| = c\,|e_n^p| \iff \ln|e_{n+1}| = \ln|c| + p\ \ln|e_n|$$

Ou seja, pela equação da reta obtida:

$$p = A = 1.007087346953520 \approx 1$$
$$c = e^B = e^{-5.606299758321017} \approx 3.674641294 \times 10^{-3}$$

c representa a média de K_k ao longo dos pontos (x_i, y_i). Observando o declive que esta reta teria no caso $n \to \infty$, verificamos que a ordenada na origem seria $< B$, ou seja, $K_\infty < 3.674641294 \times 10^{-3}$. Mais à frente, demonstramos experimentalmente que $K_\infty \approx 1.0 \times 10^{-4} < c$.
Determinação da ordem de convergência:

$$|e_{n+1}| \approx C|e_n^p| \quad e \quad |e_n| \approx C|e_{n-1}^p|$$

$$\frac{|e_{n+1}|}{|e_n|} \approx \left(\frac{|e_n|}{|e_{n-1}|}\right)^p$$

$$\ln\frac{|e_{n+1}|}{|e_n|} \approx p\ \ln\frac{|e_n|}{|e_{n-1}|}$$

$$p \approx \frac{\ln(|e_{n+1}|/|e_n|)}{\ln(|e_n|/|e_{n-1}|)}$$

$$\lim_{n\to\infty} \frac{\ln(|e_{n+1}|/|e_n|)}{\ln(|e_n|/|e_{n-1}|)} = p \tag{4}$$

Código para geração de valores de $\ln\frac{|e_{n+1}|}{|e_n|} = \ln|e_{n+1}| - \ln|e_n|$:

```
% (Código em MatLab: parte final de grupo1p2b.m)
% determinacao da ordem de convergencia
ordem = zeros(1);
zz = y - x;
for i = 2 : length(zz) - 1
    ordem(i) = zz(i) / zz(i - 1);
end

fprintf("n \t ln|e_n| \t\t\t\t ln|e_{n+1}| \t\t   ln|e_{n+1}| - ln|e_n|
        \t (ln|e_{n+1}| - ln|e_n|) / (ln|e_n| - ln|e_{n - 1}|)\n")

for i = 1 : length(zz) - 1
    fprintf("%d\t%+.15e\t%.15e\t%.15e\t%.15e\n", i, x(i), y(i), zz(i), ordem(i));
end
```

n	$\ln\lvert e_n\rvert$	$\ln\lvert e_{n+1}\rvert$	$\ln\lvert e_{n+1}\rvert - \ln\lvert e_n\rvert$	$\frac{\ln\lvert e_{n+1}\rvert - \ln\lvert e_n\rvert}{\ln\lvert e_n\rvert - \ln\lvert e_{n-1}\rvert}$
0	$-1.769026705090874 \times 10^{0}$	$-3.942431126752310 \times 10^{0}$	$-2.173404421661436 \times 10^{0}$	$\lvert e_{-1}\rvert$ indefinido
1	$-3.942431126752310 \times 10^{0}$	$-7.953799890991632 \times 10^{0}$	$-4.011368764239322 \times 10^{0}$	$1.845661453643714 \times 10^{0}$
2	$-7.953799890991632 \times 10^{0}$	$-1.626230047168131 \times 10^{1}$	$-8.308500580689678 \times 10^{0}$	$2.071238290221174 \times 10^{0}$
3	$-1.626230047168131 \times 10^{1}$	$-2.549246068146767 \times 10^{1}$	$-9.230160209786355 \times 10^{0}$	$1.110929718322313 \times 10^{0}$
4	$-2.549246068146767 \times 10^{1}$	$-3.443421547668306 \times 10^{1}$	$-8.941754795215392 \times 10^{0}$	$9.687540185634937 \times 10^{-1}$

Ora,

$$\frac{\ln|e_{n+1}|-\ln|e_n|}{\ln|e_n|-\ln|e_{n-1}|} = \frac{\ln(|e_{n+1}|/|e_n|)}{\ln(|e_n|/|e_{n-1}|)}$$

Pelos valores na tabela podemos concluir que esta expressão tende para 1 quando n tende para $+\infty$. Ou seja pela equação (3): $p = 1$.

- A função iteradora é neste caso: (Figura 3)

```
% parte do codigo representado acima
% calculo dos coeficientes assimptoticos de convergencia ao longo das
% iteradas
coeficienteassimptotico = zeros(1, 1);
for i = 1 : length(x) - 1
    coeficienteassimptotico(i) = e(i + 1) / e(i);
end
fprintf("\nCoeficientes Assimptoticos de Convergencia ao longo das iteradas:\n")
for i = 1 : length(coeficienteassimptotico)
    fprintf("%+.15e\n", coeficienteassimptotico(i));
end
% se elementos deste vetor tenderem para um numero real, ordem de
% convergencia é 1 e o coeficiente é o valor deste limite
```

donde obtivemos um vetor de iteradas de $K_k = \frac{|e_{k+1}|}{|e_k|}$, com os seguintes valores :

k	K_k
1	$1.137895690725995 \times 10^{-1}$
2	$1.810859187271480 \times 10^{-2}$
3	$2.464132432754737 \times 10^{-4}$
4	$9.803752839550068 \times 10^{-5}$
5	$1.308112916306936 \times 10^{-4}$

logo a ordem de convergência é 1 e o factor assimptótico de convergência é

$$K_\infty = \lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|} \approx 1.308112916306936 \times 10^{-4} \approx 1.0 \times 10^{-4}$$

Logo como $K_\infty \approx \delta$ e $K_\infty \in \mathbb{R} : 0 < K_\infty < +\infty : p = 1$
A estimativa do coeficiente assimptótico de convergência é $\approx 1.0 \times 10^{-4} = \delta$

Comentário detalhado dos resultados: Sendo $p = 1$, e $0 < K_\infty < 1$, então a convergência é linear, ou seja, a ordem de convergência = 1.

O valor exato utilizado (valor calculado com o método da bisseção: *root* no código acima) foi $z = 1.170498853760944$

Sendo o nosso $g(x)$ para este caso igual a

$$g(x) = x - \frac{\delta}{f(x+\delta) - f(x)} f(x) \tag{5}$$

Calculamos $g'(x)$ que será igual a:

$$1 - \frac{\delta}{f(x+\delta) - f(x)} f'(x) + \frac{\delta(f'(x+\delta) - f'(x))}{(f(x+\delta) - f(x))^2} f(x)$$

Mas como $f(x) = 0$, obtemos:

$$g'(x) = 1 - \frac{\delta}{f(x+\delta) - f(x)} f'(x) \tag{6}$$

E coeficiente assimptótico:

$$K_\infty = \frac{|g^p(z)|}{p!} = |g'(z)| = \left|1 - \frac{\delta}{f(z+\delta) - f(z)} f'(z)\right| = \left|1 - \frac{10^{-4}}{f(z+10^{-4}) - f(z)} f'(z)\right| = 9.812600307501640 \times 10^{-5} \approx \delta$$

Com:

$$z = 1.170498853760944$$

$$f(x) = 1.5e^{-x}(\sin(2x) + \cos(2x)) - 0.01$$

$$f'(x) = -1.5e^{-x}(\sin(2x) + \cos(2x)) + 1.5e^{-x}(2\cos(2x) - 2\sin(2x))$$

Código em $MatLab$ para este cálculo no ficheiro *grupo1p2b.m*:

```
% calculo do coeficiente assimptotico de convergencia teoricamente
delt = 10 ^ (- 4);
fzd = 1.5 * exp(- 1 * (root + delt)) * (sin(2 * (root + delt)) + cos(2 * (root + delt))) - 0.01;
fz = 1.5 * exp(- root) * (sin(2 * root) + cos(2 * root)) - 0.01;
dfz = - 1.5 * exp(- root) * (sin(2 * root) + cos(2 * root)) +
     + 1.5 * exp(- root) * (2 * cos(2 * root) - 2 * sin(2 * root));
cof = abs(1 - (delt * dfz) / (fzd - fz));
```

(c) Como a equação pedida (3) é diferente da função (2) utilizada para o cálculo de intervalos na alínea *2(a)*, não poderemos utilizar o valor exato da primeira raiz calculado neste alínea. Iremos então utilizar a raiz obtida pelo método da bisseção aplicado à função (3), calculado na alínea *2(b)*:

$$z = 1.170498853760944$$

Código para obter os valores da tabela:

```
% (Código em MatLab: grupo1p2c.m)
% inicializacoes
z = zeros (1, 14);
num_iter = zeros (1, 14);
erros = zeros (1, 14);

% valor do zero de f calculado na alinea 2b) pelo metodo da bissecao
zero = 1.170498853760944;

f = @(r) 1.5 .* exp(- 1 .* r) .* (1 .* sin(2 .* r) + cos(2 .* r)) - 0.01;

% Para delta de 10 ^ -1 a 10 ^ (- 14)  fazer o calculo da raiz de f e determinar o
% erro

for i = 1 : 14
    delta = 10 ^ (- i);
    [z(i), fx, iter] = newtonquasi(f, 1.0, delta, 1e-5, 1000);
    num_iter(i) = length(iter);
    erros(i) = abs(z(i) - zero);
```

```
end

fprintf("Numero de iteradas ao longo dos diferentes delta:\n")
disp(num_iter)

fprintf("Modulo do erro ao longo dos diferentes delta:\n")
disp(erros)
```

O valor mais apropriado é $\delta = 10^{-7}$		
δ	$\|e_n\|$	Número de iteradas
10^{-1}	$1.323886833493049 \times 10^{-7}$	7
10^{-2}	$1.446182862352430 \times 10^{-8}$	5
10^{-3}	$2.140063681821403 \times 10^{-10}$	5
10^{-4}	$8.487655023259322 \times 10^{-12}$	5
10^{-5}	$1.152411499560913 \times 10^{-12}$	5
10^{-6}	$1.052491427344648 \times 10^{-13}$	5
10^{-7}	$2.664535259100376 \times 10^{-15}$	5
10^{-8}	$1.243449787580175 \times 10^{-14}$	5
10^{-9}	$2.464695114667848 \times 10^{-14}$	5
10^{-10}	$8.659739592076221 \times 10^{-14}$	5
10^{-11}	$1.973976537783528 \times 10^{-13}$	5
10^{-12}	$1.763789114761494 \times 10^{-11}$	5
10^{-13}	$9.814016266318504 \times 10^{-11}$	5
10^{-14}	$8.486413793917791 \times 10^{-10}$	5

O valor de $Delta$ que parece mais apropriado é $Delta = 10^{-7}$, pois é o que tem o $|e_n|$ mais baixo. Para além disso, o número de iteradas é baixo, não sendo maior do que para outros valores de $Delta$.
De $10^{-1} \leq Delta \leq 10^{-7}$, à medida que $Delta$ diminui, $|e_n|$ também diminui, atingindo um mínimo em 10^{-7}. Ou seja 10^{-7} é o valor de $Delta$ que minimiza o erro. Para $10^{-7} \leq Delta \leq 10^{-14}$, à medida que $Delta$ diminui, $|e_n|$ aumenta, o que demonstra que $Delta$ e $|e_n|$ não são proporcionais.

Grupo II

Como temos 13 pontos para modelar, utilizando o método dos mínimos quadrados, o polinómio de grau 12 é interpolador e todos os polinómios aproximadores de grau $m \in \mathbb{N}$ tal que : $1 \leq m \leq 11$ não são interpoladores, logo são esses que iremos calcular.
Isto porque quando utilizamos o método dos mínimos quadrados com $n+1$ pontos e $g \in P_n$ então obtemos o polinómio interpolador.
No método dos mínimos quadrados, encontramos a função aproximadora tal que $g(x_i) \approx f(x_i)$. Enquanto que na interpolação encontraríamos a função aproximadora tal que $g(x_i) = f(x_i)$, nos pontos que seriam interpoladores, e SSE seria igual a 0, tal como MSE.

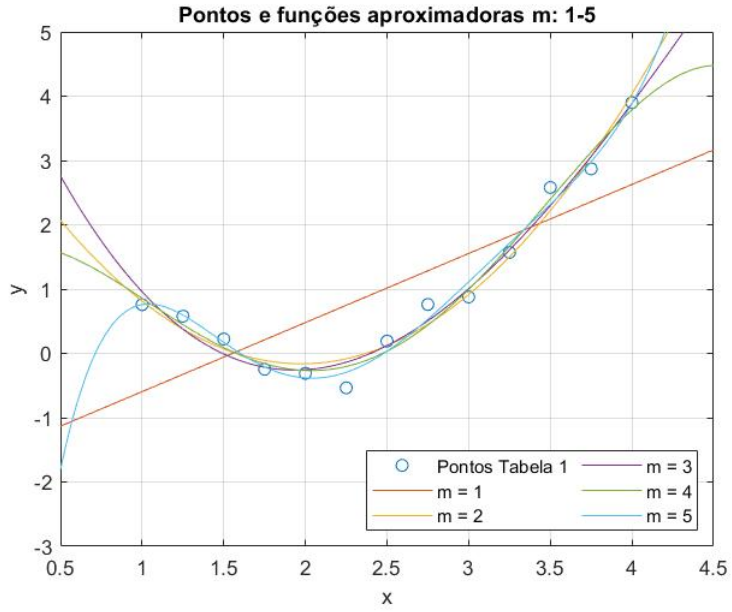
- Código para obter as funções aproximadores, os valores da tabela e o gráfico da Figura 4:

```
(Código em MatLab: Ficheiro grupo2.m)
% inicializacoes
x = [1, 1.25, 1.5, 1.75, 2, 2.25, 2.5, 2.75, 3, 3.25, 3.5, 3.75, 4];
y = [0.757, 0.578, 0.223, -0.247, -0.311, -0.536, 0.191, 0.761, 0.877, 1.57, 2.58, 2.87, 3.9];
MSE = zeros(1, 1);
MSEmplus1 = zeros(1, 1);
SSE = zeros(1, 1);
gl = zeros(1, 1);
p = zeros(1, 1);
e = zeros(1, 1);
DivMSE = zeros(1, 1); % Guarda abs(MSE(m) / MSE(m + 1))
NumeroDePontos = 13;

% geracao do metodo para m entre 1 e 5
figure(1);
z = 0.5 : 0.001 : 4.5; % pois os pontos a aproximar estao em [0.5; 4.5]
plot(x, y, 'o');
hold on;
for m = 1 : 5
    [p, e] = polyfit(x, y, m);
    plot(z, polyval(p, z));
    % Escreve os coeficientes do polinomio aproximador escolhido
    if m == 2
        fprintf("Coeficientes para m = 2\n")
       disp(p)
    end
    hold on;
    SSE(m) = e.normr ^ 2;
end
% elementos do grafico
ylim([-3 5])
xlim([0.5 4.5]) % pois os pontos a aproximar estao em [0.5; 4.5]
title('Pontos e funções aproximadoras m: 1-5');
xlabel('x');
ylabel('y');
legend('Pontos Tabela 1', 'm = 1', 'm = 2', 'm = 3', 'm = 4', 'm = 5', 'Location', 'southeast',
       'NumColumns', 2);
grid;
hold off;

% geracao do metodo para m entre 6 e 11
figure(2);
plot(x, y, 'o');
hold on;
for m = 6 : 11
    [p, e] = polyfit(x, y, m);
    plot(z, polyval(p, z));
    hold on;
    SSE(m) = e.normr ^ 2;
```

```
end
% elementos do grafico
title('Pontos e funções aproximadoras m: 6-11');
ylim([-3 5])
xlim([0.5 4.5]) % pois os pontos a aproximar estao em [0.5; 4.5]
grid;
xlabel('x');
ylabel('y');
legend('Pontos Tabela 1', 'm = 6', 'm = 7', 'm = 8', 'm = 9', 'm = 10', 'm = 11', 'Location',
        'southeast', 'NumColumns', 2);
hold off;

% calculo de vetor de MSE_m
for m = 1 : 11
    gl(m) = NumeroDePontos - (m + 1);
    MSE(m) = SSE(m) / gl(m);
end

% calculo de vetor de MSE_(m + 1)
for m = 1 : 10
    MSEmplus1(m) = MSE(m + 1);
end

% calculo de vetor de |MSE_m / MSE_(m + 1)|
for m = 1 : 10
    DivMSE(m) = abs(MSE(m) / MSE(m + 1));
end

DivMSE(11) = 0; % apenas para imprimir
fprintf("m \t SSE_m \t\t\t\t\t MSE_m \t\t\t\t\t MSE_m / MSE_(m+1)\n")

for i = 1 : 11
    fprintf("%d\t%+.15e\t%.15e\t%.15e\n", i, SSE(i), MSE(i), DivMSE(i));
end
```

grau m do polinómio	SSE_m	MSE_m	$\text{MSE}_m/\text{MSE}_{m+1}$
1	$8.802928527472528 \times 10^{0}$	$8.002662297702298 \times 10^{-1}$	$1.323785930694589 \times 10^{1}$
2	$6.045284295704297 \times 10^{-1}$	$6.045284295704297 \times 10^{-2}$	$1.080994758443972 \times 10^{0}$
3	$5.033101061438553 \times 10^{-1}$	$5.592334512709504 \times 10^{-2}$	$1.039155353309799 \times 10^{0}$
4	$4.305292366457000 \times 10^{-1}$	$5.381615458071250 \times 10^{-2}$	$1.236159250124705 \times 10^{0}$
5	$3.047447827025395 \times 10^{-1}$	$4.353496895750564 \times 10^{-2}$	$8.686083037169597 \times 10^{-1}$
6	$3.007222157873249 \times 10^{-1}$	$5.012036929788748 \times 10^{-2}$	$8.333996680979523 \times 10^{-1}$
7	$3.006982796877996 \times 10^{-1}$	$6.013965593755992 \times 10^{-2}$	$1.192383359154198 \times 10^{0}$
8	$2.017460424144772 \times 10^{-1}$	$5.043651060361930 \times 10^{-2}$	$1.875073051540019 \times 10^{0}$
9	$8.069527301167582 \times 10^{-2}$	$2.689842433722528 \times 10^{-2}$	$6.947718366081478 \times 10^{-1}$
10	$7.743095767537861 \times 10^{-2}$	$3.871547883768930 \times 10^{-2}$	$5.687605353791283 \times 10^{0}$
11	$6.806991067318353 \times 10^{-3}$	$6.806991067318353 \times 10^{-3}$	Não sabemos MSE_{12}

Justificação da escolha do modelo: Após uma análise, escolhemos o modelo com $m = 2$, pela sua simplicidade na forma de representar o comportamento aproximado do sistema de pontos, com uma função quadrática de $grau = 2$, que é relativamente baixo (função com pouca complexidade e de simples compreensão do comportamento do sistema). Para além disso, o SSE associado a este modelo é relativamente baixo como se pode observar na tabela acima. MSE é o SSE por grau de liberdade, também sendo este relativamente baixo.
O SSE descresce com o aumento do grau do polinómio, sendo $m = 11$ o valor de m que minimiza SSE. Já MSE em $1 \leq m \leq 5$ diminui como aumento de m, enquanto que para $5 \leq m \leq 11$ não apresenta um comportamento de crescimento/decrescimento consistente, sendo $m = 9$ o valor de m que minimiza MSE.
Prescindimos de utilizar um polinómio de grau mais elevado, com SSE e MSE associado mais baixos, pelo seguinte facto:



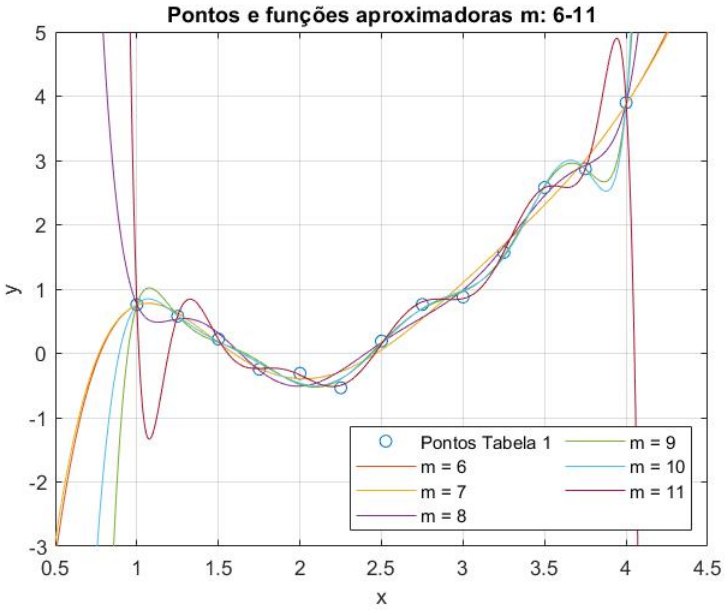


Figura 4: Pontos e funções aproximadoras.

Com um polinómio de $m = 2$, temos uma curva simples, de aparência suave, que passe através dos pontos (aproximadamente), e não um polinómio oscilatório que os "leve os pontos muito a sério".

Modelo escolhido:

$$m = 2: \; y(x) = 1.023888111888112x^2 - 4.045374625374625x + 3.834618381618382 \tag{7}$$

Grupo III

1. Código da função **integratrap**:

```
% (Código em MatLab: Ficheiro integratrap.m)

function [n, TN, dif1, dif2] = integratrap(f, alpha, beta, maxk)
    if maxk < 1
        fprintf('maxk invalid')
    end
    dif1 = zeros(1);
    dif2 = zeros(1, 1);
    TN = zeros(1);
    n = zeros(1);
    h = zeros(1);
    h = (beta - alpha) / 2;
    maxk = 2 ^ maxk;
    % soma do primeiro com o ultimo termo de f(xi)
    suminicial = (feval(f, alpha) + feval(f, beta)) / 2;
    k = 2;
    i = 1;
    while k <= maxk
        integral = suminicial;
        h = (beta - alpha) / k;
        % soma de todos os f(xi), exceto para o primeiro e para o ultimo
        % termo de xi
        for q = 1 : (k - 1)
             x = alpha + q * h; % calcula xi
             integral = integral + feval(f, x); % adiciona f(xi)
        end
        % pelo metodo dos trapezios, o integral sera a soma de
        % [0.5 * (f(x0) + f(xn)) + (soma de todos os outros f(xi))] * h
        TN(i) = integral * h;
        % calculo de |T_(2n) - T_n|
        if k > 2
             dif1(i) = abs(TN(i) - TN(i - 1));
        end
        % calculo de |T_(2n) - T_n| / |T_(4n) - T_(2n)|
        if k > 4
            dif2(i) =  dif1(i - 1) / dif1(i);
        end
        n(i) = k;
        i = i + 1;
        k = 2 ^ i;
    end
end
```

2. (a) Para esta alínea, foi implementado o script abaixo para a representação das três funções lado a lado. É de notar que na alínea seguinte, será feita uma representação mais detalhada de cada função, mostrando também o comportamento do aumento dos subintervalos, sendo uma importante ajuda visual para a compreensão e explicação deste método iterativo usando a função **integratrap**.

Código para obter os gráficos da Figura 5

```
% (Código em MatLab: Ficheiro grupo3p2a.m)
% numero de pontos utilizados para produzir o grafico
N = 5000;
ints = [linspace(5, 11, N); linspace(5, 2 * pi + 5, N); linspace(5, 7, N)];
f_1 = @(x) exp(5 - x) .* sin(50 .* (x - 5));
f_2 = @(x) 1 ./ (2 + sin(x - 5));
f_3 = @(x) exp(- x .^ 2 + 10 .* x - 25);
f = {f_1, f_2, f_3}; % array das funções para iterar

for n = 1 : 3
    subplot(1, 3, n);
    range = ints(n, :);
    plot(range, f{n}(range));
    xlabel('x');
    ylabel('y');
    xlim([range(1) range(end)]);
    title(sprintf("Função %d", n));
    grid;
end
```

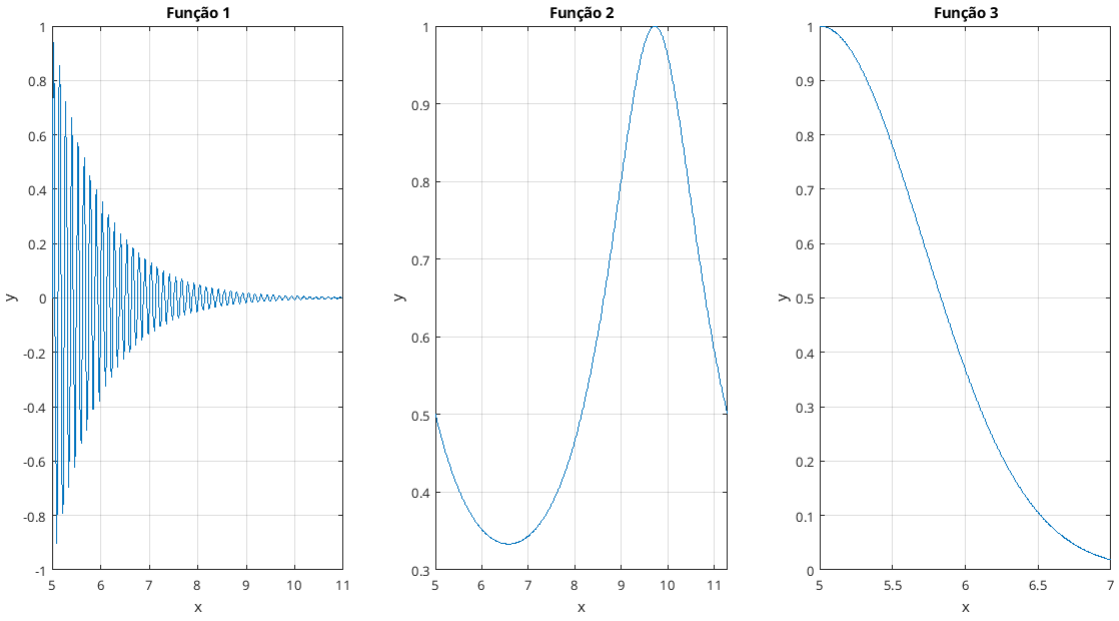


Figura 5: Gráfico das funções integrandas.

(b) Código para calcular os valores dos integrais utilizando a função **integratrap**:

```
% (Código em MatLab: Ficheiro grupo3p2b.m)
I = [5 11; 5 2 * pi + 5; 5 7]; % matriz com os valores a, b de cada função
f_1 = @(x) exp(5 - x) .* sin(50 .* (x - 5));
f_2 = @(x) 1 ./ (2 + sin(x - 5));
f_3 = @(x) exp(- x .^ 2 + 10 .* x - 25);
f = {f_1, f_2, f_3};

for i = 1 : 3
    r = I(i, :); % linha i da matriz I
    [n, t, dif1, dif2] = integratrap(f{i}, r(1), r(2), 19);
    fprintf("função %s\n", func2str(f{i}));
    fprintf("\t  n\t T_(n) \t\t\t    |(T_(n) - T_(n/2)| \t\t |(T_(n/2) -
            - T_(n/4)| / |(T_(n) - T_(n/2)| \n")
    for j = 1 : length(n)
        fprintf(" %6d\t%+.15e\t%.15e\t%.15e\n", n(j), t(j), dif1(j), dif2(j));
    end
    fprintf("\n");
end
```

Tabelas com os valores obtidos:

Primeiro integral			
n	T_n	$\lvert T_n - T_{n/2}\rvert$	$\lvert T_{n/2} - T_{n/4}\rvert / \lvert T_n - T_{n/2}\rvert$
2	$-1.104920254759999 \times 10^{-1}$	$0.000000000000000 \times 10^{0}$	$0.000000000000000 \times 10^{0}$
4	$-2.005333115252731 \times 10^{-1}$	$9.004128604927317 \times 10^{-2}$	$0.000000000000000 \times 10^{0}$
8	$-2.334762723742623 \times 10^{-1}$	$3.294296084898921 \times 10^{-2}$	$2.733248127332061 \times 10^{0}$
16	$-2.425825523165001 \times 10^{-1}$	$9.106279942237810 \times 10^{-3}$	$3.617609062970854 \times 10^{0}$
32	$2.335275830389911 \times 10^{-3}$	$2.449178281468900 \times 10^{-1}$	$3.718095988004719 \times 10^{-2}$
64	$-4.551915229915253 \times 10^{-2}$	$4.785442812954244 \times 10^{-2}$	$5.117976281816490 \times 10^{0}$
128	$9.875184325204893 \times 10^{-3}$	$5.539433662435742 \times 10^{-2}$	$8.638866542270385 \times 10^{-1}$
256	$1.765117356076462 \times 10^{-2}$	$7.775989235559725 \times 10^{-3}$	$7.123767143482944 \times 10^{0}$
512	$1.941858466744171 \times 10^{-2}$	$1.767411106677091 \times 10^{-3}$	$4.399649411607105 \times 10^{0}$
1024	$1.985083238254685 \times 10^{-2}$	$4.322477151051454 \times 10^{-4}$	$4.088884787389016 \times 10^{0}$
2048	$1.995831331113535 \times 10^{-2}$	$1.074809285884944 \times 10^{-4}$	$4.021622447644323 \times 10^{0}$
4096	$1.998514752230554 \times 10^{-2}$	$2.683421117019233 \times 10^{-5}$	$4.005369410966144 \times 10^{0}$
8192	$1.999185382830339 \times 10^{-2}$	$6.706305997847273 \times 10^{-6}$	$4.001340108668785 \times 10^{0}$
16384	$1.999353026444884 \times 10^{-2}$	$1.676436145448684 \times 10^{-6}$	$4.000334886630821 \times 10^{0}$
32768	$1.999394936471403 \times 10^{-2}$	$4.191002651900144 \times 10^{-7}$	$4.000083714307865 \times 10^{0}$
65536	$1.999405413923212 \times 10^{-2}$	$1.047745180962623 \times 10^{-7}$	$4.000020928800296 \times 10^{0}$
131072	$1.999408033282673 \times 10^{-2}$	$2.619359460928616 \times 10^{-8}$	$4.000005331804195 \times 10^{0}$
262144	$1.999408688122415 \times 10^{-2}$	$6.548397414596341 \times 10^{-9}$	$4.000000756047699 \times 10^{0}$
524288	$1.999408851832359 \times 10^{-2}$	$1.637099442119982 \times 10^{-9}$	$3.999999783835007 \times 10^{0}$

Segundo integral			
n	T_n	$\lvert T_n - T_{n/2}\rvert$	$\lvert T_{n/2} - T_{n/4}\rvert / \lvert T_n - T_{n/2}\rvert$
2	$3.141592653589793 \times 10^{0}$	$0.000000000000000 \times 10^{0}$	$0.000000000000000 \times 10^{0}$
4	$3.665191429188091 \times 10^{0}$	$5.235987755982983 \times 10^{-1}$	$0.000000000000000 \times 10^{0}$
8	$3.627791516645356 \times 10^{0}$	$3.739991254273489 \times 10^{-2}$	$1.400000000000026 \times 10^{1}$
16	$3.627598733591012 \times 10^{0}$	$1.927830543446696 \times 10^{-4}$	$1.939999999993204 \times 10^{2}$
32	$3.627598728468436 \times 10^{0}$	$5.122575696958620 \times 10^{-9}$	$3.763400791893205 \times 10^{4}$
64	$3.627598728468436 \times 10^{0}$	$0.000000000000000 \times 10^{0}$	Inf
128	$3.627598728468435 \times 10^{0}$	$8.881784197001252 \times 10^{-16}$	$0.000000000000000 \times 10^{0}$
256	$3.627598728468434 \times 10^{0}$	$1.332267629550188 \times 10^{-15}$	$6.666666666666666 \times 10^{-1}$
512	$3.627598728468439 \times 10^{0}$	$4.884981308350689 \times 10^{-15}$	$2.727272727272727 \times 10^{-1}$
1024	$3.627598728468433 \times 10^{0}$	$6.217248937900877 \times 10^{-15}$	$7.857142857142857 \times 10^{-1}$
2048	$3.627598728468435 \times 10^{0}$	$2.664535259100376 \times 10^{-15}$	$2.333333333333333 \times 10^{0}$
4096	$3.627598728468424 \times 10^{0}$	$1.110223024625157 \times 10^{-14}$	$2.400000000000000 \times 10^{-1}$
8192	$3.627598728468446 \times 10^{0}$	$2.176037128265307 \times 10^{-14}$	$5.102040816326531 \times 10^{-1}$
16384	$3.627598728468437 \times 10^{0}$	$8.437694987151190 \times 10^{-15}$	$2.578947368421053 \times 10^{0}$
32768	$3.627598728468429 \times 10^{0}$	$8.881784197001252 \times 10^{-15}$	$9.500000000000000 \times 10^{-1}$
65536	$3.627598728468440 \times 10^{0}$	$1.110223024625157 \times 10^{-14}$	$8.000000000000000 \times 10^{-1}$
131072	$3.627598728468436 \times 10^{0}$	$3.552713678800501 \times 10^{-15}$	$3.125000000000000 \times 10^{0}$
262144	$3.627598728468451 \times 10^{0}$	$1.509903313490213 \times 10^{-14}$	$2.352941176470588 \times 10^{-1}$
524288	$3.627598728468513 \times 10^{0}$	$6.217248937900877 \times 10^{-14}$	$2.428571428571429 \times 10^{-1}$

Terceiro integral			
n	T_n	$\lvert T_n - T_{n/2}\rvert$	$\lvert T_{n/2} - T_{n/4}\rvert / \lvert T_n - T_{n/2}\rvert$
2	$8.770372606158094 \times 10^{-1}$	$0.000000000000000 \times 10^{0}$	$0.000000000000000 \times 10^{0}$
4	$8.806186341245393 \times 10^{-1}$	$3.581373508729890 \times 10^{-3}$	$0.000000000000000 \times 10^{0}$
8	$8.817037913321337 \times 10^{-1}$	$1.085157207594389 \times 10^{-3}$	$3.300326886893367 \times 10^{0}$
16	$8.819862452657772 \times 10^{-1}$	$2.824539336434562 \times 10^{-4}$	$3.841890936325893 \times 10^{0}$
32	$8.820575578012112 \times 10^{-1}$	$7.131253543402050 \times 10^{-5}$	$3.960789388911675 \times 10^{0}$
64	$8.820754296107942 \times 10^{-1}$	$1.787180958301438 \times 10^{-5}$	$3.990224666549545 \times 10^{0}$
128	$8.820799002925637 \times 10^{-1}$	$4.470681769452867 \times 10^{-6}$	$3.997557979887612 \times 10^{0}$
256	$8.820810181335849 \times 10^{-1}$	$1.117841021192056 \times 10^{-6}$	$3.999389613279148 \times 10^{0}$
512	$8.820812976045025 \times 10^{-1}$	$2.794709176301424 \times 10^{-7}$	$3.999847392605732 \times 10^{0}$
1024	$8.820813674728968 \times 10^{-1}$	$6.986839429234948 \times 10^{-8}$	$3.999961935016792 \times 10^{0}$
2048	$8.820813849400391 \times 10^{-1}$	$1.746714228811896 \times 10^{-8}$	$3.999989989196659 \times 10^{0}$
4096	$8.820813893068272 \times 10^{-1}$	$4.366788153298273 \times 10^{-9}$	$3.999997635544990 \times 10^{0}$
8192	$8.820813903985197 \times 10^{-1}$	$1.091692514165743 \times 10^{-9}$	$4.000016576678018 \times 10^{0}$
16384	$8.820813906714446 \times 10^{-1}$	$2.729249048982751 \times 10^{-10}$	$3.999973965632194 \times 10^{0}$
32768	$8.820813907396778 \times 10^{-1}$	$6.823319687043750 \times 10^{-11}$	$3.999884475829415 \times 10^{0}$
65536	$8.820813907567422 \times 10^{-1}$	$1.706434993309358 \times 10^{-11}$	$3.998581671025751 \times 10^{0}$
131072	$8.820813907610036 \times 10^{-1}$	$4.261369035418738 \times 10^{-12}$	$4.004429044108068 \times 10^{0}$
262144	$8.820813907620528 \times 10^{-1}$	$1.049271780573235 \times 10^{-12}$	$4.061263358374775 \times 10^{0}$
524288	$8.820813907623183 \times 10^{-1}$	$2.654543251878749 \times 10^{-13}$	$3.952739439565036 \times 10^{0}$

Comentário detalhado:

- **Integral 1**

  Sendo a função 1 o produto de uma função exponencial com uma função sinusoidal de baixo período ($T = \frac{\pi}{25} \approx 0.126$), ou seja, de frequência razoavelmente elevada ($f = \frac{25}{\pi} \approx 7.958$), para um número pequeno de subdivisões n (*ver Figura 6 abaixo*), na integração pelo método do Trapézios, as áreas criadas pelos nós (a cinzento) não traduzem o verdadeiro formato (a vermelho) e consequente valor do integral da função.

  Isto ocorre devido ao facto de que, para baixo n, o tamanho de cada uma das subdivisões é maior, sendo $x_{i+1} - x_i > T$, o que faz com que x_i e x_{i+1} pertençam a oscilações de diferentes períodos, fazendo com que a área do trapézio respetivo a esse subintervalo seja uma representação irrealista.

  Ao aumentar o número de subintervalos n, começamos a obter valores mais precisos para o integral, visto que a distância entre nós, h, é suficientemente pequena para que se consigam criar áreas de trapézios que acompanham a função integranda de forma consistente nos subintervalos pretendidos: neste caso $x_{i+1} - x_i < T$, ficando tanto x_i como x_{i+1} na mesma oscilação na maior parte dos casos. Aliás, para n suficientemente elevado até podemos ter vários subintervalos dentro da mesma oscilação, o que aumenta imenso a precisão do cálculo.

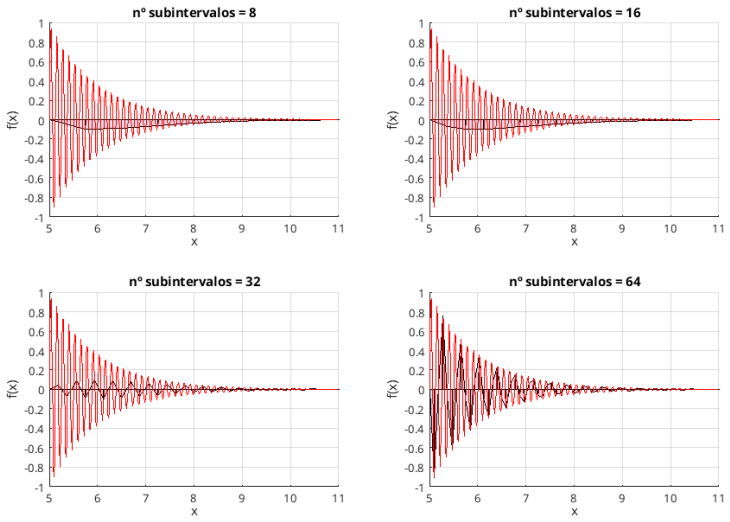
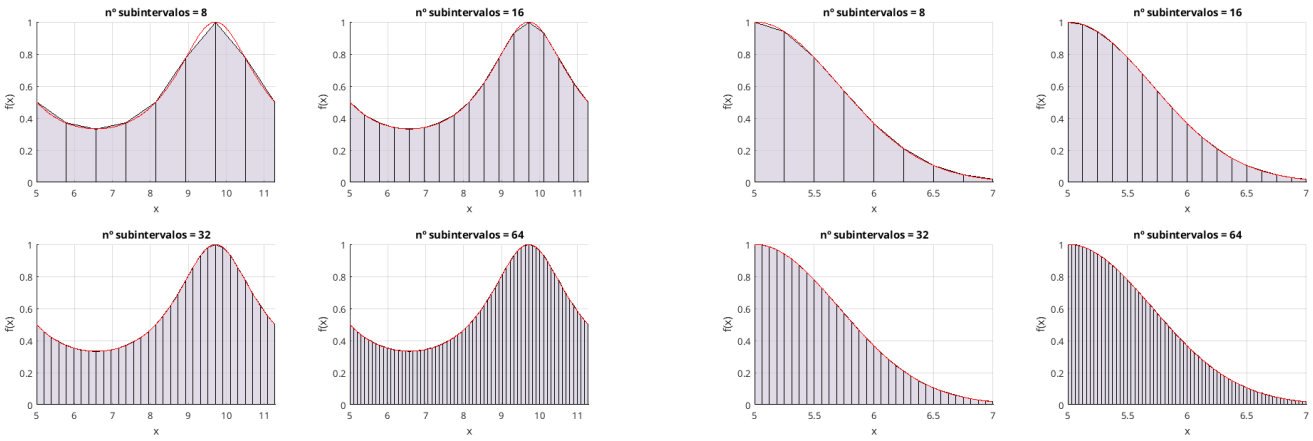


Figura 6: subintervalos do integral 1

- **Integrais 2 e 3**

  Como as funções 2 e 3 apresentam curvas mais simples, de aparência mais suave que a função 1, o cálculo do valor do integral pelo método dos Trapézios é bastante rápido, sendo possível criar áreas de subintervalos que se ajustam de forma muito próxima à função dada a partir de n bastante pequeno, como se verifica abaixo na Figura 7.



(a) Integral 2

(b) Integral 3

Figura 7: subintervalos dos integrais 2 e 3

- **Comentário final**

  Em relação ao desempenho da função **integratrap**, esta mostrou-se bastante rápida no cálculo dos três integrais com $MaxK = 19$. Foi feito um teste de velocidade de execução, em que se repetiu o cálculo dez vezes para cada uma das três funções, e se obteve a média das mesmas, usando as funções *tic* e *toc* para obter o tempo de execução. A média obtida foi de $0.2404904s$, um valor bastante aceitável, tendo em conta a quantidade de cálculos e iterações feitas na rotina da função criada.

  Comparativamente com a mesma função feita com base no código do ficheiro *trapezios.m*, da biblioteca fornecida pela professora, é visível uma grande diferença em relação à memória utilizada, que neste vai aumentando exponencialmente, assim como do tempo computacional, o que tornaria esta solução insustentável para o problema em questão.

  Assim, optámos por não guardar em memória o valor de x_i e de y_i de cada nó, mas sim, de calcular estes valores iterativamente, poupando imensa memória, aumentando a eficiência.

  A regra dos Trapézios tem um desempenho pior do que se esperava para o integral 1, pois só a partir de $n = 512$ (elevado) é que os algarismos significativos de T_n começam a estabilizar, pelos motivos discutidos na página anterior.

  Já para os integrais 2 e 3, tem o desempenho esperado, pois a cada n_{i+1}, a partir de $n = 4$, $\frac{E_n(f)}{E_{2n}(f)} \approx 2^2$.

  Se pretendermos um resultado com poucos algarismos significativos, este método é bastante rápido. Já para obter resultados com um maior número de algarismos significativos seria mais eficiente e rápido a utilização, por exemplo, do método de Simpson, pois neste $E_n(f) \approx Ch^4$, enquanto que para o método dos Trapézios $E_n(f) \approx Ch^2$. Sabendo que $\frac{E_n(f)}{E_{2n}(f)} \approx 2^p$, pela coluna relativa a $|T_{n/2} - T_{n/4}|/|T_n - T_{n/2}|$, podemos verificar que os valores desta tendem para $4 = 2^2$, ou seja $p = 2$ no método dos Trapézios, enquanto que no método de Simpson seria $p = 4$. Isto significaria que o erro iria diminuir de iterada para iterada mais rapidamente, ou seja, iria convergir para o integral mais rapidamente: obteríamos um resultado com precisão $< \epsilon$ com um menor número de subintervalos n, do que seria necessário no método dos Trapézios.

Código usado para criar as figuras 6 e 7

```
% (Código em MatLab: Ficheiro subtrapezios_plot.m)
% (reaproveitando as variáveis I, f_1, f_2 e f_3 definidas acima no script grupo3p2b.m)
figure(1);
clf;
set(gcf, 'Position', [0 0 1000 1000]);
func = f_1;          % mudar aqui a função a desenhar
interv = I(1, :);    % mudar aqui nº da linha do intervalo da função a desenhar
r_linha = linspace(interv(1), interv(2), 500);
for i = 3 : 6
    h = (interv(2) - interv(1)) / (2 ^ i);
    % o round(interv(2)) + 1 é usado devido ao ponto final do segundo
    % intervalo ser um valor irracional
    xi = interv(1) : h : round(interv(2)) + 1;
    yi = feval(func, xi);
    subplot(2, 2, i - 2);
    hold on;
    title(sprintf("nº subintervalos = %d", 2 ^ i));
    for j = 1 : length(xi) - 1
        axis on;
        xlim(interv);
        % criar os pontos para cada trapézio e encher o "interior" da função
        pts_x = [ xi(j) xi(j) xi(j + 1) xi(j + 1) ];
        pts_y = [ 0 yi(j) yi(j + 1) 0];
        plot(pts_x, pts_y, 'k');
        fill(pts_x, pts_y, 'y', 'FaceAlpha', 0.7, 'EdgeAlpha', 0.3, 'FaceColor', '#D4CCDC');
    end
    % plot da função escolhida
    plot(r_linha, func(r_linha), 'r');
    xlabel('x');
    ylabel('f(x)');
    grid;
    hold off;
end
```

(c) Código para calcular o número mínimo de subintervalos para que o erro seja menor que 10^{-6}.

```
(Código em MatLab: Ficheiro grupo3p2c_extra.m)
% (utilizando a função integratrap feita na pergunta 1 do grupo III)
% (reaproveitando o código na pergunta 2 b, e executando pequenas alterações)
I = [5 11; 5 2 * pi + 5; 5 7]; % matriz com os valores a, b de cada função
f_1 = @(x) exp(5 - x) .* sin(50 .* (x - 5));
f_2 = @(x) 1 ./ (2 + sin(x - 5));
f_3 = @(x) exp(- x .^ 2 + 10 .* x - 25);
f = {f_1, f_2, f_3};
dif3 = zeros(19, 1);

for i = 1 : 3
    r = I(i, :); % linha i da matriz I
    [n, t, dif1, dif2] = integratrap(f{i}, r(1), r(2), 19);
    dif3 = dif1 / 3;
    fprintf("função %s\n", func2str(f{i}));

    fprintf("\t  n\t |(T_(n) - T_(n/2)| \t\t |e_(n)| \n")
    for j = 1 : length(n)
        fprintf(" %6d\t%.15e\t%.15e\n", n(j), dif1(j), dif3(j));
    end
    fprintf("\n");
end
```

Para o cálculo experimental: Utilizando a fórmula: $|\epsilon_{2n}^T| = \frac{1}{3}|T_{2n} - T_n|$

Primeiro integral		
n	$\lvert T_n - T_{n/2}\rvert$	$\lvert\epsilon_n^T\rvert$
2	$0.000000000000000 \times 10^{0}$	$0.000000000000000 \times 10^{0}$
4	$9.004128604927317 \times 10^{-2}$	$3.001376201642439 \times 10^{-2}$
8	$3.294296084898921 \times 10^{-2}$	$1.098098694966307 \times 10^{-2}$
16	$9.106279942237810 \times 10^{-3}$	$3.035426647412603 \times 10^{-3}$
32	$2.449178281468900 \times 10^{-1}$	$8.163927604896334 \times 10^{-2}$
64	$4.785442812954244 \times 10^{-2}$	$1.595147604318081 \times 10^{-2}$
128	$5.539433662435742 \times 10^{-2}$	$1.846477887478581 \times 10^{-2}$
256	$7.775989235559725 \times 10^{-3}$	$2.591996411853242 \times 10^{-3}$
512	$1.767411106677091 \times 10^{-3}$	$5.891370355590302 \times 10^{-4}$
1024	$4.322477151051454 \times 10^{-4}$	$1.440825717017152 \times 10^{-4}$
2048	$1.074809285884944 \times 10^{-4}$	$3.582697619616479 \times 10^{-5}$
4096	$2.683421117019233 \times 10^{-5}$	$8.944737056730776 \times 10^{-6}$
8192	$6.706305997847273 \times 10^{-6}$	$2.235435332615757 \times 10^{-6}$
16384	$1.676436145448684 \times 10^{-6}$	$5.588120484828948 \times 10^{-7}$
32768	$4.191002651900144 \times 10^{-7}$	$1.397000883966715 \times 10^{-7}$
65536	$1.047745180962623 \times 10^{-7}$	$3.492483936542076 \times 10^{-8}$
131072	$2.619359460928616 \times 10^{-8}$	$8.731198203095388 \times 10^{-9}$
262144	$6.548397414596341 \times 10^{-9}$	$2.182799138198780 \times 10^{-9}$
524288	$1.637099442119982 \times 10^{-9}$	$5.456998140399941 \times 10^{-10}$

A partir desta tabela, concluímos experimentalmente que o número mínimo de subintervalos para que o erro seja menor que 10^{-6}, para o primeiro integral, é de 16384.

Segundo integral		
n	$\lvert T_n - T_{n/2}\rvert$	$\lvert\epsilon_n^T\rvert$
2	$0.000000000000000 \times 10^{0}$	$0.000000000000000 \times 10^{0}$
4	$5.235987755982983 \times 10^{-1}$	$1.745329251994328 \times 10^{-1}$
8	$3.739991254273489 \times 10^{-2}$	$1.246663751424496 \times 10^{-2}$
16	$1.927830543446696 \times 10^{-4}$	$6.426101811488986 \times 10^{-5}$
32	$5.122575696958620 \times 10^{-9}$	$1.707525232319540 \times 10^{-9}$
64	$0.000000000000000 \times 10^{0}$	$0.000000000000000 \times 10^{0}$
128	$8.881784197001252 \times 10^{-16}$	$2.960594732333751 \times 10^{-16}$
256	$1.332267629550188 \times 10^{-15}$	$4.440892098500626 \times 10^{-16}$
512	$4.884981308350689 \times 10^{-15}$	$1.628327102783563 \times 10^{-15}$
1024	$6.217248937900877 \times 10^{-15}$	$2.072416312633626 \times 10^{-15}$
2048	$2.664535259100376 \times 10^{-15}$	$8.881784197001252 \times 10^{-16}$
4096	$1.110223024625157 \times 10^{-14}$	$3.700743415417189 \times 10^{-15}$
8192	$2.176037128265307 \times 10^{-14}$	$7.253457094217689 \times 10^{-15}$
16384	$8.437694987151190 \times 10^{-15}$	$2.812564995717063 \times 10^{-15}$
32768	$8.881784197001252 \times 10^{-15}$	$2.960594732333751 \times 10^{-15}$
65536	$1.110223024625157 \times 10^{-14}$	$3.700743415417189 \times 10^{-15}$
131072	$3.552713678800501 \times 10^{-15}$	$1.184237892933500 \times 10^{-15}$
262144	$1.509903313490213 \times 10^{-14}$	$5.033011044967377 \times 10^{-15}$
524288	$6.217248937900877 \times 10^{-14}$	$2.072416312633625 \times 10^{-14}$

A partir desta tabela, concluímos experimentalmente que o número mínimo de subintervalos para que o erro seja menor que 10^{-6}, para o segundo integral, é de 32.

Terceiro integral		
n	$\lvert T_n - T_{n/2}\rvert$	$\lvert\epsilon_n^T\rvert$
2	$0.000000000000000 \times 10^{0}$	$0.000000000000000 \times 10^{0}$
4	$3.581373508729890 \times 10^{-3}$	$1.193791169576630 \times 10^{-3}$
8	$1.085157207594389 \times 10^{-3}$	$3.617190691981298 \times 10^{-4}$
16	$2.824539336434562 \times 10^{-4}$	$9.415131121448539 \times 10^{-5}$
32	$7.131253543402050 \times 10^{-5}$	$2.377084514467350 \times 10^{-5}$
64	$1.787180958301438 \times 10^{-5}$	$5.957269861004792 \times 10^{-6}$
128	$4.470681769452867 \times 10^{-6}$	$1.490227256484289 \times 10^{-6}$
256	$1.117841021192056 \times 10^{-6}$	$3.726136737306855 \times 10^{-7}$
512	$2.794709176301424 \times 10^{-7}$	$9.315697254338080 \times 10^{-8}$
1024	$6.986839429234948 \times 10^{-8}$	$2.328946476411649 \times 10^{-8}$
2048	$1.746714228811896 \times 10^{-8}$	$5.822380762706321 \times 10^{-9}$
4096	$4.366788153298273 \times 10^{-9}$	$1.455596051099424 \times 10^{-9}$
8192	$1.091692514165743 \times 10^{-9}$	$3.638975047219143 \times 10^{-10}$
16384	$2.729249048982751 \times 10^{-10}$	$9.097496829942504 \times 10^{-11}$
32768	$6.823319687043750 \times 10^{-11}$	$2.274439895681250 \times 10^{-11}$
65536	$1.706434993309358 \times 10^{-11}$	$5.688116644364527 \times 10^{-12}$
131072	$4.261369035418738 \times 10^{-12}$	$1.420456345139580 \times 10^{-12}$
262144	$1.049271780573235 \times 10^{-12}$	$3.497572601910785 \times 10^{-13}$
524288	$2.654543251878749 \times 10^{-13}$	$8.848477506262498 \times 10^{-14}$

A partir desta tabela, concluímos experimentalmente que o número mínimo de subintervalos para que o erro seja menor que 10^{-6}, para o terceiro integral, é de 256.

	Teórico	Experimental	Teórico / Experimental
1º caso	208968	16384	12.754394531250000
2º caso	4547	32	142.0937500000000
3º caso	1155	256	4.511718750000000

Comentário detalhado: Para calcular o número mínimo de subintervalos para que o erro seja menor que 10^{-6}, procedemos da seguinte forma:

$$E_n^T(f) = \frac{-(b-a)h^2}{12} f''(\xi)$$

$$|E_n^T(f)| \leq \left|\frac{(b-a)h^2}{12}M\right| \leq \frac{(b-a)^3}{12n^2}M < \epsilon$$

$$h = \frac{b-a}{n}, \quad M = \max_{x\in[a;b]} |f''(x)|$$

$$n^2 > \frac{(b-a)^3 M}{12\epsilon} \iff n > \left(\frac{(b-a)^3 M}{12\epsilon}\right)^{\frac{1}{2}}$$

$$n = \left\lceil \left(\frac{(b-a)^3 M}{12\epsilon}\right)^{\frac{1}{2}} \right\rceil$$

A partir deste resultado, fizemos o cálculo teórico de n para as 3 funções pedidas, a partir do seguinte código:

```
(Código em Octave: Ficheiro calculo_der.m)
% GNU/Octave

pkg load symbolic;
syms x;

% array com as funções dadas
f ={exp(5 - x) * sin(50 * (x - 5)), 1 / (2 + sin(x - 5)), exp(- x ^ 2 + 10 * x - 25)};

% calcula e imprime a segunda derivada de cada função
for i = 1 : 3
    fprintf('Função %d\n', i);
    d2f = function_handle(diff(diff(f{i})))
end

(Código em MatLab: Ficheiro erro.m)
% funcao que calcula teoricamente o numero minimo de subintervalos para
% atingir uma certa precisao
function [n] = erro (b, a, Maxddx, erroMax)
    dif = abs(((b - a) ^ 3 ) / 12);
    erron = dif / erroMax;
    n = sqrt(erron * Maxddx);

    % ultilizamos n - 0.5 para a função arredondar n para baixo, e somamos
    % 1 para conseguir majorar o n
    n = round(n - 0.5) + 1;
end

(Código em MatLab: Ficheiro grupo3p2cerrosscript.m)
% segundas derivadas das funcoes calculadas anteriormente em calculo_der.m
ddf1 = @(x) - 2499 * exp (5 - x) .* sin (50 * x - 250) -
        - 100 * exp (5 - x) .* cos (50 * x - 250);
ddf2 = @(x) sin (x - 5) ./ (sin (x - 5) + 2) .^ 2 +
        + 2 * cos (x - 5) .^ 2 ./ (sin (x - 5) + 2) .^ 3;
ddf3 = @(x) (10 - 2 * x) .^ 2 .* exp (- x .^ 2 + 10 * x - 25) -
        -2 * exp (- x .^ 2 + 10 * x - 25);

% calculo do maximo do modulo das segundas derivadas nos respetivos
% intervalos de integracao
N = 1000;
limits = [5 11; 5 2 * pi + 5; 5 7];
ints = [linspace(5, 11, N); linspace(5, 2 * pi + 5, N); linspace(5, 7, N)];
f = {ddf1, ddf2, ddf3};
Maxddx = zeros(3, 1);
num_intervalos = zeros(3, 1);

% calculo do numero minimo de subintervalos para atingir a precisao pedida
```

```
for n = 1 : 3
    range = ints(n, :);
    Maxddx(n) = max(abs(f{n}(range)));
    num_intervalos(n) = erro (limits(n, 2), limits(n, 1), Maxddx(n), 10 ^ (- 6));
end

fprintf("Número mínimo de intervalos:\n")
disp(num_intervalos)
```

A fórmula teórica do erro determina o número de subintervalos (n) utilizados, através de uma majoração do módulo do erro, ou seja, o n obtido é uma majoração, o que significa que no máximo são necessários n intervalos para atingir a precisão ϵ. Os resultados computacionais estão de acordo com a fórmula teórica do erro, pois o valor de n experimental é menor que o valor de n teórico, respeitando a majoração de n, e respeitando a majoração do erro. As elevadas diferenças entre o valor teórico e experimental, são devido à estimativa de erro anterior ser manifestamente irrealista. Tal acontece devido aos valores elevados de M nos intervalos em causa. Aqui utilizámos o método dos Trapézios, obtendo valores de n elevados. A utilização da Regra de Simpson teria vantagem sobre esta, e obteríamos valores de n mais baixos, pois seria:

$$n > \left\lceil \left(\frac{(b-a)^5 M}{180\epsilon} \right)^{\frac{1}{4}} \right\rceil \quad e \quad n\ par$$

Bibliografia

[1] M. M. Graça e P. T. Lima, *Apontamentos de Matemática Computacional*, set. de 2019.

[2] I. R. d. Santos, *Matemática Computacional*, 1 de set. de 2020.

[3] ——, *Biblioteca de programas*, 2017. URL: `https://web.tecnico.ulisboa.pt/~isabel.santos/an/matlab/` (acedido em 08/12/2020).

[4] MATLAB, *MATLAB R2019a*, versão 9.6.0.1072779, 2019.

[5] ——, *MATLAB R2020b*, versão 9.9.0.1467703, 2020.

[6] J. W. Eaton, *GNU Octave*, versão 5.2.0, 2020.