Relatório – VERSÃO 59

Eduardo Montes, nº 94150 Manuel Salema, nº 96269 Pedro Resende, nº 92991 Rafael Olim, nº 93326

22 de Dezembro de 2020

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

- Eduardo Montes (25%): Todas as perguntas
- Manuel Salema (25%): Todas as perguntas
- Pedro Resende 25%): Todas as perguntas
- Rafael Olim (25%): Todas as perguntas

Grupo I

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

```
function [z,fz,iter] = newtonquasi(f,x0,TolX,MaxIter)
    %   Inputs
    %       f - função a aplicar o método
    %       x0 - aproximação inicial
    %       TolX - precisão mínima exigida
    %       MaxIter - número máximo de iteradas
    %
    %   Ouputs
    %       z - aproximação da raiz
    %       fz - valor da função na raiz
    %       iter - vector que contem as iteradas
    %

%Inicialização das variáveis auxiliares
Delta = 1e-5;
NIter = 0;

    iter(1) = x0;

while NIter < MaxIter

    %cálculo de x(n+1)
    iter(NIter+2) = iter(NIter+1) - ( Delta / ( f(iter(NIter+1) + Delta) -  f(iter(NIter+1)) ) ) *
    f(iter(NIter+1));

    NIter = NIter + 1;

    %verificação da precisão
    if( (abs(iter(NIter+1) - iter(NIter)) / abs(iter(NIter)) ) < TolX )
        z = iter(NIter+1);
        fz = f(z);
        break;
    end
end
```

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

```
Para x0 = -0.2
format long
[z, fz, iter] = newtonquasi(@(x) power(x,3) - power(x,2) - x + 1,-0.2,1e-2,1000)

Para x0 = -0.3
format long
[z, fz, iter] = newtonquasi(@(x) power(x,3) - power(x,2) - x + 1,-0.3,1e-2,1000)
```

x_n	x_n
-0.2000000000000000	-0.3000000000000000
2.199920003257039	8.798670201365423
1.694690306819625	6.009340485204284
1.387008629219663	4.164056127899828
1.208017409671791	2.953033630479767
1.108690345536816	2.169962618580354
1.055713188505415	1.676118425462648
1.028231592389497	1.375977672206094
1.014215892049550	1.201774916685671
1.007135457584903	1.105310385106001
	1.053942616976839
	1.027323454488433
	1.013755707316576
	1.006903780588617

x_0	-0.2	-0.3
Número de iteradas	9	13

2. (a) Código para obter os extremos dos intervalos:

```
function [] = findintervals(precisao)
    %   Inputs
    %       precisao - precisão com que se quer o cálculo do valor dos
    %                  extremos dos intervalos
    %
    %   Ouputs
    %       (none)
    %
    %   Explicação
    %       Analizando a função w temos que apenas podemos obter um intervalo porque para
    %       todo o r > 1 temos que w(r) nunca é > 0.1, isto pode ser vizualizado no
    %       gráfico aberto automaticamente ao correr a função. O intervalo em que
    %       w(r) >= 0.1 começa em 0 visto que w(0) > 0.1. Para calcular o limite direito
    %       do intervalo aplicou-se a função da bissecao à função w(r)-0.1 usando valores
    %       de r de 0 e 1 porque [w(0)-0.1] * [w(1)-0-1] < 0
    %       A função bissecao usada foi a que se encontra disponível na página da cadeira
    %       com uma ligeira alteração nos outputs que a função dá

    %gráfico da função w(r)
    fplot(@(r) 1*exp(-2*r)*(2*sin(4*r)+cos(4*r)), [0 10])

    % calculo do limite direito do intervalo em que w(r) - 0.1 >= 0
    zero =  bissecao (@(r) exp(-2*r)*(2*sin(4*r)+cos(4*r))-0.1, 0, 1, precisao, precisao, 1000);

    fprintf('O intervalo em que w(r) >=  0.1 é de 0 a %.15d\n', zero)

end
```

```
function [root] = bissecao(f, a, b, eps_abs, eps_step, iter_max)
   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));
```

Intervalos
$[0,\ 6.299126743959365 \times 10^{-1}]$
$[X.XXXXXXXXXXXXXXX \times 10^{X},\ X.XXXXXXXXXXXXXXX \times 10^{X}]$

(b) • Código para obter os vectores $\mathbf{x}$ e $\mathbf{y}$, ajuste dos pontos e gráfico da Figura 1:

```
function [] = finderrorsnewtonquasi()
    %   Inputs
    %       (none)
    %
    %   Ouputs
    %       (none)

    %obtem o valor "correto" para o zero
    z = bissecao (@(r) exp(-2*r)*(2*sin(4*r)+cos(4*r))-0.001, 0, 1, power(10,-12)
    , power(10,-12), 1000);

    [a, b, c] = newtonquasi(@(r) exp(-2*r)*(2*sin(4*r)+cos(4*r))-0.001, 0.5, 1e-12, 100);

    %obtem os valores do x e do y
```

```
    x = log (abs(z - c(1:length(c)-1)));
    y = log( abs(z - c(2:length(c))));

    %obtem os dados da reta de ajuste
    p1 = polyfit(x,y,1);

    %faz o gráfico
    plot(x,y,'o')
    grid
    title(['Line Equation is y = ',num2str(p1(1)),'x + ','(',num2str(p1(2)),')']);
    hold on
    plot(x,polyval(p1,x),'-r')
    hold off
    legend('Dados','Reta de ajuste')
    xlabel('ln | ek - 1 |')
    ylabel('ln | ek |')

end
```

A recta obtida é $y = Ax + B$ com $A = 0.95537$ e $B = -5.3149$

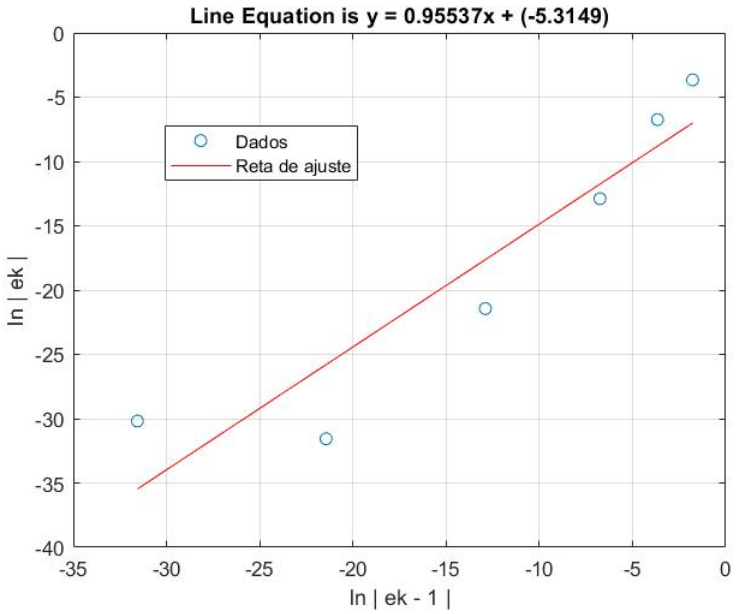

Figura 1: Gráfico da ordem de convergência.

- A função iteradora é neste caso

$$g(z) = z - \frac{\delta}{f(z+\delta) - f(z)} f(z)$$

donde
Delta é um valor arbitrário e

$$f(r) = e^{-2r}(2sin(4r) + cos(4r)) - 0.001$$

logo a ordem de convergência é 4 e o factor assimptótico de convergência é

$$K_\infty = \frac{g^{(4)}(z)}{4!}$$

A estimativa do coeficiente assimptótico de convergência é XX

Comentário detalhado dos resultados: Quanto mais vezes iteramos aplicando o método de Quasi-Newton, mais próximo do valor real irá ser o valor obtido. Isto acontece porque à metida que iteramos aplicando este método a distância ao valor real irá diminuir. É isso que podemos observar no gráfico. O erro de x das iteradas consecutivas está a diminuir. O método de Quasi-Newton

não calcula precisamente os valores porque faz uma aproximação da função f recorrendo ao valor de uma variação média de um intervalo pequeno, assim, os resultados obtidos têm certamente algum erro associado devido a isso. Também poderíamos ter usado um valor de precisão mais elevado para o cálculo do valor "certo", obtendo assim valores melhores. A ordem de convergência obtida foi de 4 o que quer dizer que o valor irá convergir muito rapidamente para o zero que se pretende calcular.

(c) Código para obter os valores da tabela:

```
Nota: Os valores de Delta foram alterados dentro da função newtonquasi


function [] = findbetterdelta()
    %   Inputs
    %       (none)
    %
    %   Ouputs
    %       (none)

    z =  bissecao (@(r) exp(-2*r)*(2*sin(4*r)+cos(4*r))-0.001, 0, 1, power(10,-15),
    power(10,-15), 1000);

    [a, b, c] = newtonquasi(@(r) exp(-2*r)*(2*sin(4*r)+cos(4*r))-0.001, 0.5, 1e-5, 100)

    en = abs( z - a)

end
```

O valor mais apropriado de Delta é $\delta = 10^{-6}$ porque é o valor em que o erro é menor e o número de iterações também

O valor mais apropriado é $\delta = 10^{-6}$		
δ	$\lvert e_n \rvert$	Número de iteradas
10^{-1}	$7.711622341810553e-07$	8
10^{-2}	$5.392542301585479e-09$	5
10^{-3}	$5.901172883682193e-10$	4
10^{-4}	$4.890271521063028e-10$	4
10^{-5}	$3.973166240456294e-11$	4
10^{-6}	$9.754419494356625e-12$	4
10^{-7}	$1.474731448070088e-11$	4
10^{-8}	$1.526256898642941e-11$	4
10^{-9}	$1.520550352296368e-11$	4
10^{-10}	$1.528044357712588e-11$	4
10^{-11}	$1.459765641698141e-11$	4
10^{-12}	$9.844680626258651e-12$	4
10^{-13}	$3.989078289912129e-09$	4
10^{-14}	$5.392542301585479e-09$	5

Grupo II

Código para obter as funções aproximadores, os valores da tabela e o gráfico da Figura 2 em formato `.jpg`:

```
format shortE;

%%Generate Data Table
x = 1:0.25:4;
y = [0.757, 0.578, 0.223, -0.247, -0.311, -0.536, 0.191, 0.761, 0.877, 1.57, 2.5, 2.87, 3.9];
```

```
%p(0) não é possível ser realizado dentro do for uma vez que não dá para
%criar um array P que inclua o 0 logo tem de ser feito à parte

[P0, e0] = polyfit(x, y, 0);
W0 = polyval(P0, x);
SSE0 = e0.normr^2;
MSEm0 = SSE0 / (13 - 1);
MSEmplus0 = SSE0 / (13 - 2);
divMSE0 = MSEm0 / MSEmplus0;

disp(P0);

for i = 1:11 %o i comportar-se à como o grau do polinómio

    %dentro dos cases o i das formulas sera previamente substituido
    %MSEm = SSE / (13 - (i + 1));
    %MSEmplus = SSE / (13 - ((i + 1) + 1));

   [P, e] = polyfit(x, y, i);
   disp(P);
   switch i
       case 1
           [P1] = polyval(P, x);
           AuxSSE1 = (P1 - y).^2; %Variável auxiliar no cálculo do SSE
           SSE1 = sum(AuxSSE1); %Realiza a soma de todos os valores do array AuxSSE1
           MSEm1 = SSE1 / 11; %Realiza o cálculo do MSE = nº de pontos - nº de parâmetros a estimar
           MSEmplus1 = SSE1 / 10; %Cálculo do MSE(m+1) = nº de pontos - nº de parâmetros a estimar + 1
           divMSE1 = MSEm1 / MSEmplus1; %Realiza o cálculo do ultimo valor pedido na tabela
       case 2
           [P2] = polyval(P, x);
           AuxSSE2 = (P2 - y).^2;
           SSE2 = sum(AuxSSE2);
           MSEm2 = SSE2 / 10;
           MSEmplus2 = SSE2 / 9;
           divMSE2 = MSEm2 / MSEmplus2;
       case 3
           [P3] = polyval(P, x);
           AuxSSE3 = (P3 - y).^2;
           SSE3 = sum(AuxSSE3);
           MSEm3 = SSE3 / 9;
           MSEmplus3 = SSE3 / 8;
           divMSE3 = MSEm3 / MSEmplus3;
       case 4
           [P4] = polyval(P, x);
           AuxSSE4 = (P4 - y).^2;
           SSE4 = sum(AuxSSE4);
           MSEm4 = SSE4 / 8;
           MSEmplus4 = SSE4 / 7;
           divMSE4 = MSEm4 / MSEmplus4;
       case 5
           [P5] = polyval(P, x);
           AuxSSE5 = (P5 - y).^2;
           SSE5 = sum(AuxSSE5);
           MSEm5 = SSE5 / 7;
           MSEmplus5 = SSE5 / 6;
           divMSE5 = MSEm5 / MSEmplus5;
       case 6
           [P6] = polyval(P, x);
```

```
            AuxSSE6 = (P6 - y).^2;
            SSE6 = sum(AuxSSE6);
            MSEm6 = SSE6 / 6;
            MSEmplus6 = SSE6 / 5;
            divMSE6 = MSEm6 / MSEmplus6;
        case 7
            [P7] = polyval(P, x);
            AuxSSE7 = (P7 - y).^2;
            SSE7 = sum(AuxSSE7);
            MSEm7 = SSE7 / 5;
            MSEmplus7 = SSE7 / 4;
            divMSE7 = MSEm7 / MSEmplus7;
        case 8
            [P8] = polyval(P, x);
            AuxSSE8 = (P8 - y).^2;
            SSE8 = sum(AuxSSE8);
            MSEm8 = SSE8 / 4;
            MSEmplus8 = SSE8 / 3;
            divMSE8 = MSEm8 / MSEmplus8;
        case 9
            [P9] = polyval(P, x);
            AuxSSE9 = (P9 - y).^2;
            SSE9 = sum(AuxSSE9);
            MSEm9 = SSE9 / 3;
            MSEmplus9 = SSE9 / 2;
            divMSE9 = MSEm9 / MSEmplus9;
        case 10
            [P10] = polyval(P, x);
            AuxSSE10 = (P10 - y).^2;
            SSE10 = sum(AuxSSE10);
            MSEm10 = SSE10 / 2;
            MSEmplus10 = SSE10 / 1;
            divMSE10 = MSEm10 / MSEmplus10;
        case 11
            [P11] = polyval(P, x);
            AuxSSE11 = (P11 - y).^2;
            SSE11 = sum(AuxSSE11);
            MSEm11 = SSE11 / 1;
            MSEmplus11 = SSE11 / 0;
            divMSE11 = MSEm11 / MSEmplus11;
    end
end
s = "--------------------------------"; %método de separação dos valores representados no output

disp(s);

%realização do display no output dos valores pedidos na tabela
disp(SSE0);
disp(SSE1);
disp(SSE2);
disp(SSE3);
disp(SSE4);
disp(SSE5);
disp(SSE6);
disp(SSE7);
disp(SSE8);
disp(SSE9);
disp(SSE10);
```

```
disp(SSE11);

disp(s);

disp(MSEm0);
disp(MSEm1);
disp(MSEm2);
disp(MSEm3);
disp(MSEm4);
disp(MSEm5);
disp(MSEm6);
disp(MSEm7);
disp(MSEm8);
disp(MSEm9);
disp(MSEm10);
disp(MSEm11);

disp(s);

disp(divMSE0);
disp(divMSE1);
disp(divMSE2);
disp(divMSE3);
disp(divMSE4);
disp(divMSE5);
disp(divMSE6);
disp(divMSE7);
disp(divMSE8);
disp(divMSE9);
disp(divMSE10);
disp(divMSE11);


f1 = figure;
plot(x, y, "o");
xlim([1 4])
hold on;

plot(x, W0);
plot(x, P1);
plot(x, P2);
plot(x, P3);
plot(x, P4);
plot(x, P5);

legend('(x, f(x))','grau 0','grau 1','grau 2','grau 3','grau 4','grau 5')
title('Polinómios não interpoladores com grau entre 0 e 5')
xlabel('x','FontSize',12);
ylabel('f(x)','FontSize',12);
hold off

f2 = figure;
plot(x, y, "o");
xlim([1 4])
hold on;

plot(x, P6);
plot(x, P7);
```

```
plot(x, P8);
plot(x, P9);
plot(x, P10);
plot(x, P11);

legend('(x, f(x))','grau 6','grau 7','grau 8','grau 9','grau 10','grau 11')
title('Polinómios não interpoladores com grau entre 6 e 11')
xlabel('x','FontSize',12);
ylabel('f(x)','FontSize',12);
hold off
```

grau m do polinómio	SSE_m	MSE_m	$\text{MSE}_m/\text{MSE}_{m+1}$
0	2.1681×10^{1}	1.8068×10^{0}	9.1667×10^{-1}
1	8.7299×10^{0}	7.9363×10^{-1}	9.0909×10^{-1}
2	5.5201×10^{-1}	5.5201×10^{-2}	9.0000×10^{-1}
3	4.6316×10^{-1}	5.1462×10^{-2}	8.8889×10^{-1}
4	4.0540×10^{-1}	5.0674×10^{-2}	8.7500×10^{-1}
5	2.6629×10^{-1}	3.8042×10^{-2}	8.5714×10^{-1}
6	2.6156×10^{-1}	4.3593×10^{-2}	8.3333×10^{-1}
7	2.6053×10^{-1}	5.2107×10^{-2}	8.0000×10^{-1}
8	1.8322×10^{-1}	4.5804×10^{-2}	7.5000×10^{-1}
9	8.3752×10^{-2}	2.7917×10^{-2}	6.6667×10^{-1}
10	8.2473×10^{-2}	4.1237×10^{-2}	5.0000×10^{-1}
11	6.2875×10^{-3}	6.2875×10^{-3}	0

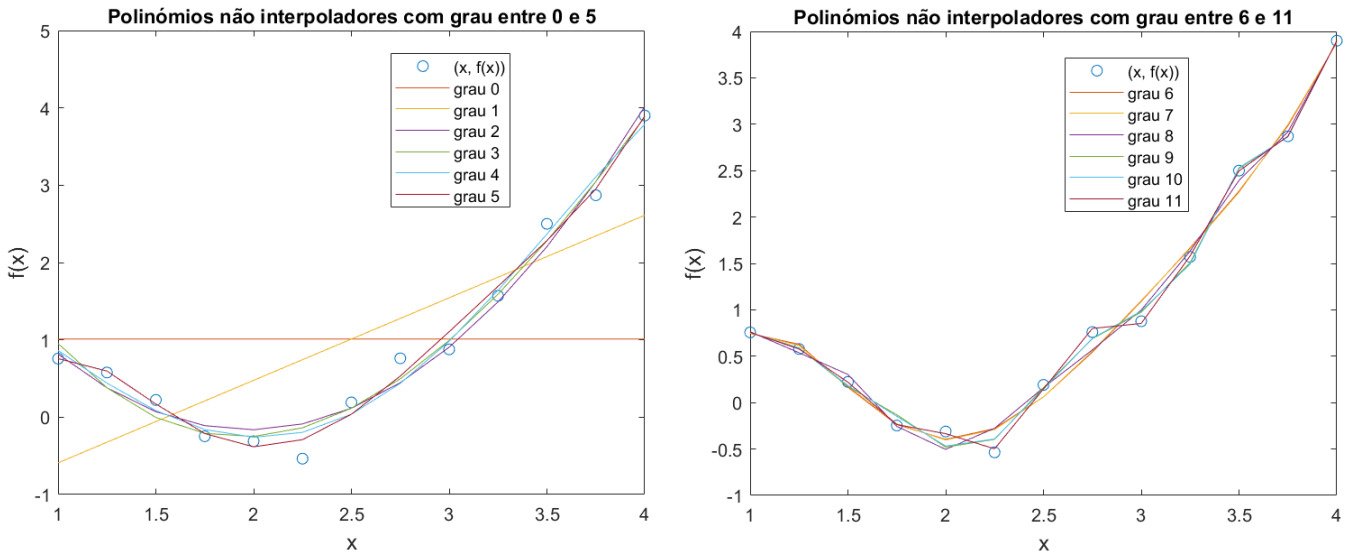

Figura 2: Pontos e funções aproximadoras.

Justificação da escolha do modelo: Escolhemos este modelo, uma vez que achamos que é aquele que apresenta uma melhor relação entre complexidade e erro. Ou seja, aquele que é fácil de compreender por ter uma baixa complexidade, mas que tem também um erro relativamente baixo.
Modelo escolhido:

$$5.3971x^3 - 6.3309x^2 + 2.0197x - 0.1329$$

Grupo III

1. Código da função **integratrap**:

```
function t = integratrap(f, alpha, beta, MaxK)
```

```
%   Inputs:
%       f - função integranda
%       alpha - primeiro valor do intervalo de integração
%       beta - último valor do intervalo de integração
%       MaxK - número máximo de k
%
%   Output:
%       t - matriz de tamanho [MaxK x 4] da forma:
%           [n.º subintervalos  |Tn|  |T2n-Tn|  |T2n-Tn|/|T4n-T2n|]

   h(1) = (beta-alpha)/2;
   x = alpha:h(1):beta;
   y = feval(f,x);
   t(1, 1) = 2;
   t(1, 2) = h(1)*trapz(y);
   t(1, 3) = 0;
   t(1, 4) = 0;
   if MaxK > 1
    for k = 2:MaxK
       t(k, 1) = 2^k;
       h(k) = h(k-1)/2;
       x = alpha:h(k):beta;
       y = feval(f,x);
       t(k, 2) = h(k)*trapz(y);
       t(k, 3) = abs(t(k, 2) - t(k-1, 2));
       t(k, 4) = t(k-1, 3)/t(k, 3);
    end
   end
end
```

2. (a) Código para obter os gráficos da Figura 3

Script graficos.m

```
% 1.ª instrução - imprimir os gráficos lado a lado
% 2.ª instrução - função integranda
% 3.ª instrução - fazer gráfico com eixo dos xx entre alpha e beta
% 4.ª instrução - limite do eixo dos yy
% 5.ª instrução - título do gráfico

subplot(1, 3, 1)
y1 = @(x) exp(4-x).*sin(50.*(x-4));
fplot(y1, [4 10])
ylim([-1 1])
title('exp(4-x)*sin(50*(x-4))')

subplot(1, 3, 2)
y2 = @(x) 1./(2+sin(x-4));
fplot(y2, [4 2.*pi+4])
ylim([0 1.1])
title('1/(2 + sin(x-4))')

subplot(1, 3, 3)
y3 = @(x) exp(-x.^2 + 8.*x - 16);
fplot(y3, [4 6])
ylim([0 1.1])
title('exp(-x^2 + 8*x - 16)')
```

Command Window:

```
>> graficos
```

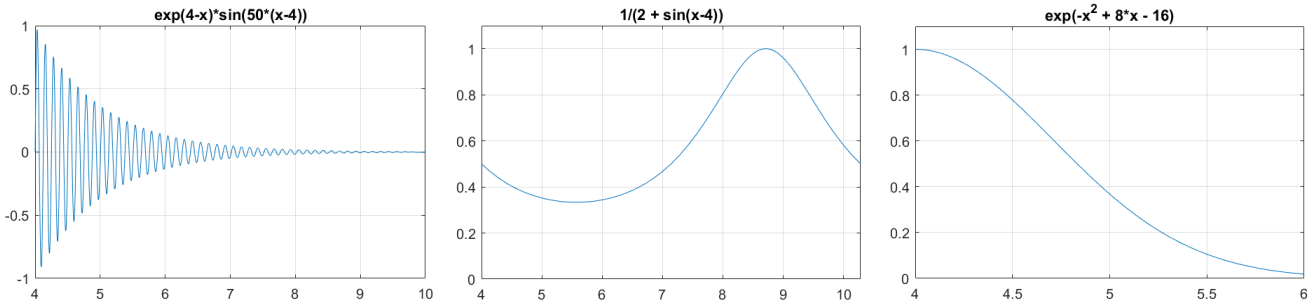

Figura 3: Gráfico das funções integrandas.

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

Script matrizes.m

```
MaxK = 19; % número máximo de k
alpha = 4; % o primeiro valor do intervalo de integração (igual para os 3 integrais)

% função 1
y1 = @(x) exp(4-x).*sin(50.*(x-4));
beta1 = 10;
% função 2
y2 = @(x) 1./(2+sin(x-4));
beta2 = 2.*pi+4;
% função 3
y3 = @(x) exp(-x.^2 + 8.*x - 16);
beta3 = 6;

% determinação da matriz para as 3 funções
int1 = integratrap(y1, alpha, beta1, MaxK)
int2 = integratrap(y2, alpha, beta2, MaxK)
int3 = integratrap(y3, alpha, beta3, MaxK)
```

Command Window:

```
>> format longE
>> matrizes
```

Tabela com os valores obtidos:

Primeiro integral			
n	T_n	$\lvert T_{2n} - T_n \rvert$	$\lvert T_{2n} - T_n \rvert / \lvert T_{4n} - T_{2n} \rvert$
2	$-1.104920254759999 \times 10^{-1}$	0	0
4	$-2.005333115252731 \times 10^{-1}$	9.0041×10^{-2}	0
8	$-2.334762723742623 \times 10^{-1}$	3.2943×10^{-2}	2.7332×10^{0}
16	$-2.425825523165001 \times 10^{-1}$	9.1063×10^{-3}	3.6176×10^{0}
32	$2.335275830389911 \times 10^{-3}$	2.4492×10^{-1}	3.7181×10^{-2}
64	$-4.551915229915256 \times 10^{-2}$	4.7854×10^{-2}	5.1180×10^{0}
128	$9.875184325204907 \times 10^{-3}$	5.5394×10^{-2}	8.6389×10^{-1}
256	$1.765117356076461 \times 10^{-2}$	7.7760×10^{-3}	7.1238×10^{0}
512	$1.941858466744169 \times 10^{-2}$	1.7674×10^{-3}	4.3996×10^{0}
1024	$1.985083238254685 \times 10^{-2}$	4.3225×10^{-4}	4.0889×10^{0}
2048	$1.995831331113539 \times 10^{-2}$	1.0748×10^{-4}	4.0216×10^{0}
4096	$1.998514752230553 \times 10^{-2}$	2.6834×10^{-5}	4.0054×10^{0}
8192	$1.999185382830347 \times 10^{-2}$	6.7063×10^{-6}	4.0013×10^{0}
16384	$1.999353026444874 \times 10^{-2}$	1.6764×10^{-6}	4.0003×10^{0}
32768	$1.999394936471402 \times 10^{-2}$	4.1910×10^{-7}	4.0001×10^{0}
65536	$1.999405413923214 \times 10^{-2}$	1.0477×10^{-7}	4.0000×10^{0}
131072	$1.999408033282739 \times 10^{-2}$	2.6194×10^{-8}	4.0000×10^{0}
262144	$1.999408688122408 \times 10^{-2}$	6.5484×10^{-9}	4.0000×10^{0}
524288	$1.999408851832310 \times 10^{-2}$	1.6371×10^{-9}	4.0000×10^{0}

Segundo integral			
n	T_n	$\lvert T_{2n} - T_n \rvert$	$\lvert T_{2n} - T_n \rvert / \lvert T_{4n} - T_{2n} \rvert$
2	$3.141592653589793 \times 10^{0}$	0	0
4	$3.665191429188091 \times 10^{0}$	5.2360×10^{-1}	0
8	$3.627791516645356 \times 10^{0}$	3.7400×10^{-2}	1.4000×10^{1}
16	$3.627598733591012 \times 10^{0}$	1.9278×10^{-4}	1.9400×10^{2}
32	$3.627598728468436 \times 10^{0}$	5.1226×10^{-9}	3.7634×10^{4}
64	$3.627598728468436 \times 10^{0}$	0	Inf
128	$3.627598728468435 \times 10^{0}$	8.8818×10^{-16}	0
256	$3.627598728468435 \times 10^{0}$	0	Inf
512	$3.627598728468436 \times 10^{0}$	8.8818×10^{-16}	0
1024	$3.627598728468435 \times 10^{0}$	8.8818×10^{-16}	1.0000×10^{1}
2048	$3.627598728468435 \times 10^{0}$	4.4409×10^{-16}	2.0000×10^{1}
4096	$3.627598728468434 \times 10^{0}$	8.8818×10^{-16}	5.0000×10^{-1}
8192	$3.627598728468438 \times 10^{0}$	4.4409×10^{-15}	2.0000×10^{-1}
16384	$3.627598728468434 \times 10^{0}$	4.4409×10^{-15}	1.0000×10^{0}
32768	$3.627598728468433 \times 10^{0}$	4.4409×10^{-16}	1.0000×10^{1}
65536	$3.627598728468430 \times 10^{0}$	3.1086×10^{-15}	1.4286×10^{-1}
131072	$3.627598728468435 \times 10^{0}$	4.8850×10^{-15}	6.3636×10^{-1}
262144	$3.627598728468437 \times 10^{0}$	1.3323×10^{-15}	3.6667×10^{0}
524288	$3.627598728468436 \times 10^{0}$	4.4409×10^{-16}	3.0000×10^{0}

Terceiro integral			
n	T_n	$\lvert T_{2n} - T_n \rvert$	$\lvert T_{2n} - T_n \rvert / \lvert T_{4n} - T_{2n} \rvert$
2	$8.770372606158094 \times 10^{-1}$	0	0
4	$8.806186341245394 \times 10^{-1}$	3.5814×10^{-3}	0
8	$8.817037913321336 \times 10^{-1}$	1.0852×10^{-3}	3.3003×10^{0}
16	$8.819862452657772 \times 10^{-1}$	2.8245×10^{-4}	3.8419×10^{0}
32	$8.820575578012112 \times 10^{-1}$	7.1313×10^{-5}	3.9608×10^{0}
64	$8.820754296107942 \times 10^{-1}$	1.7872×10^{-5}	3.9902×10^{0}
128	$8.820799002925631 \times 10^{-1}$	4.4707×10^{-6}	3.9976×10^{0}
256	$8.820810181335853 \times 10^{-1}$	1.1178×10^{-6}	3.9994×10^{0}
512	$8.820812976045016 \times 10^{-1}$	2.7947×10^{-7}	3.9998×10^{0}
1024	$8.820813674728977 \times 10^{-1}$	6.9868×10^{-8}	4.0000×10^{0}
2048	$8.820813849400373 \times 10^{-1}$	1.7467×10^{-8}	4.0000×10^{0}
4096	$8.820813893068260 \times 10^{-1}$	4.3668×10^{-9}	4.0000×10^{0}
8192	$8.820813903985225 \times 10^{-1}$	1.0917×10^{-9}	4.0000×10^{0}
16384	$8.820813906714478 \times 10^{-1}$	2.7293×10^{-10}	4.0000×10^{0}
32768	$8.820813907396784 \times 10^{-1}$	6.8231×10^{-11}	4.0000×10^{0}
65536	$8.820813907567340 \times 10^{-1}$	1.7056×10^{-11}	4.0005×10^{0}
131072	$8.820813907610002 \times 10^{-1}$	4.2663×10^{-12}	3.9978×10^{0}
262144	$8.820813907620663 \times 10^{-1}$	1.0660×10^{-12}	4.0020×10^{0}
524288	$8.820813907623327 \times 10^{-1}$	2.6645×10^{-13}	4.0008×10^{0}

Comentário detalhado: A fórmula do erro para o método dos trapézios é dada pela equação $E_n \approx Ch^2$, onde *C* é uma constante e *h* a distância entre os pontos. Se tivermos o dobro dos subintervalos, a distância entre os pontos vai ser metade e a fórmula do erro é dada por $E_{2n} \approx C\frac{h^2}{4}$. Desta forma, o quociente entre os erros $\frac{E_n}{E_{2n}} \approx 4$, isto é, quando se duplica o número de subintervalos, o erro fica (aproximadamente) 4 vezes menor.

Observando os resultados obtidos, é possível verificar que o método está a ter um comportamento esperado (ordem 2) no primeiro e no terceiro integral, uma vez que o erro converge para 4. No entanto, é muito mais eficiente para o terceiro integral, já que o erro converge para valores próximos de 4 com apenas 64 subintervalos, enquanto que no primeiro integral são precisos 1024 subintervalos para que isso aconteça. Também é possível observar que, para cada *k*, o erro é cerca de 2 a 3 ordens de grandeza inferior no terceiro integral, o que evidencia, mais uma vez, uma maior eficiência neste caso.

Esta diferença na eficiência reside na complexidade das respetivas funções integrandas. Ao visualizar os gráficos, verifica-se que a função correspondente ao primeiro integral possui muitas oscilações quando comparada à do terceiro integral. Consequentemente, serão necessários mais subintervalos, de maneira a melhor aproximar os polinómios interpoladores e minimizar o erro.

Assim, podemos concluir que de maneira mais ou menos eficiente, o método dos trapézios se comporta da maneira esperada. No entanto, tal não acontece no segundo integral. Observando a última coluna da tabela, verifica-se que o quociente entre os erros, para os primeiros valores de *k*, é bastante superior a 4, sendo mesmo da ordem de 10^4 para apenas 32 subintervalos (com diferença entre aproximações sucessivas da ordem de 10^{-9}). Isto mostra que o método dos trapézios tem um comportamento de ordem superior a 2 para este integral.

Este comportamento pode dever-se ao facto de o gráfico da segunda função integranda ser propício à utilização do método dos trapézios, uma vez que os polinómios interpoladores de grau 1 se ajustam bastante bem à curva.

Apesar de em termos computacionais ser possível aplicar o método dos trapézios aos três integrais, só é viável calcular analiticamente o valor do segundo integral (aquele onde são necessários menos subintervalos para obter um resultado bastante próximo do real).

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

Código da função **subintervalos**

```
function [] = subintervalos(f, alpha, beta, epsilon)

% Função que calcula e imprime o número mínimo de subintervalos
% para obter um erro inferior a epsilon
%   Inputs:
```

```
%       f - função integranda
%       alpha - primeiro valor do intervalo de integração
%       beta - último valor do intervalo de integração
%       epsilon - erro pretendido

syms g(x)
g(x) = f;
Dg = diff(g, 2);
vector = linspace(alpha, beta, 1000);
derivada = feval(Dg, vector);
derivada = abs(derivada);
M = max(derivada);
n = sqrt((((beta-alpha)^3) * M)/(12 * epsilon));
ceil(n)
end
```

Command Window

```
>> subintervalos(@(x) exp(4-x).*sin(50.*(x-4)), 4, 10, 10^-6)
>> subintervalos(@(x) 1./(2+sin(x-4)), 4, 2*pi+4, 10^-6)
>> subintervalos(@(x) exp(-x.^2 + 8.*x - 16), 4, 6, 10^-6)
```

	Teórico	Experimental	Teórico / Experimental
1º caso	208968	32768	6.4
2º caso	4547	32	142.1
3º caso	1155	512	2.3

Comentário detalhado: Através da observação da tabela é possível verificar que o valor experimental é inferior ao valor teórico para os três integrais. Esta diferença pode dever-se ao facto de estarmos a majorar o erro pelo máximo da segunda derivada da função integranda no intervalo $[\alpha, \beta]$ e não a determinar a valor da segunda derivada no ponto ξ. No entanto, este erro resultante da majoração não justifica a elevada diferença entre os valores teóricos e experimentais.

O que pode provocar esta situação é o facto da fórmula de cálculo do erro, $E_n^T(f) = -\frac{(b-a)h^2}{12} f''(\xi)$, apenas representar com precisão ocorrências do método dos trapézios com ordem 2. De facto, no 3.º caso, aquele em que o quociente entre o número de subintervalos é menor, é possível observar que o quociente entre os erros é sempre muito próximo de 4, o que indicia uma convergência quadrática.

Por sua vez, no 2.º caso, onde o quociente entre o número de subintervalos é maior, como já foi explicado na alínea anterior, o método converge mais rapidamente. Desta forma, o quociente entre os valores teóricos e os valores experimentais é muito elevado, já que era expectável que o método tivesse ordem 2, ou seja, que fossem necessários mais subintervalos.

Bibliografia

Atkinson, K. 1989. *An Introduction to Numerical Analysis*. Wiley Sons.

Pina, H. 1995. *Métodos Numéricos*. McGraw-Hill.

Acetatos disponibilizados pela professora doutora Isabel Santos na página da Unidade Curricular de Matemática Computacional no ano letivo de 2020/2021. Disponível em: http://web.tecnico.ulisboa.pt/ isabel.santos/meec20/slidesMEEC4to1.pdf