ATIVIDADE 03- MÉTODOS NUMÉRICOS PARA DERIVAÇÃO E INTEGRAÇÃO

Coimbra, 17 de junho de 2021

Disciplina: Análise Matemática II
Docente: Professor Arménio Correia
Discentes: João Almeida 2020144466
Nuno Santos 2019110035
Pedro Nogueira 2020136533

isec Engenharia

Índice

1. Introdução

Este trabalho surge do âmbito da unidade curricular de Análise Matemática 2 do curso de Engenharia Informática do Instituto Superior de Engenharia de Coimbra e consiste na implementação em MATLAB de métodos de Derivação e Integração Numérica.

O principal objetivo é a implementação de funções, através do desenvolvimento de uma GUI em linguagem de programação MATLAB, para algumas fórmulas de Derivação e Integração Numérica, nomeadamente: Diferenças finitas em 2 e 3 pontos (Progressivas, Regressivas e Centradas), 2ª derivada e também regra dos Trapézios e regra de Simpson.

Além disso é pretendido apresentar uma breve pesquisa sobre os Métodos Numéricos para Derivação e Integração, assim como a Derivação e Integração Simbólica no MATLAB e mostrar alguns exemplos de aplicação e testes dos métodos.

isec Engenharia

Métodos Numéricos para Derivação

(Fórmulas das Diferenças Finitas)

Para entendermos a Derivação Numérica temos primeiro de perceber o significado de derivada. A derivada de uma função f em um ponto pode ser descrita graficamente como a inclinação da reta tangente à função naquele ponto.

A Derivação Numérica é utilizada para calcular a derivada em situações onde não está disponível a função, e sim apenas um conjunto de pontos pertencentes a esta, ou para funções que não são deriváveis em todo o seu domínio ou de derivação não trivial.

À medida que ***h*** diminui, o valor da derivada numérica aproxima-se do valor real. No entanto, por mais pequeno que seja ***h***, este método irá sempre apresentar um erro de arredondamento grande. Uma maneira de reduzir este erro é utilizar vários pontos.

O objetivo é: partir de um conjunto de pontos, que definem um intervalo [a,b], e determinar a função f que representa tais pontos, ou seja, interpolar este conjunto de pontos. Em seguida, pode-se calcular a derivada da função f e aplicá-la a qualquer ponto pertencente ao intervalo [a,b]. Quanto maior for o número de pontos melhor será o resultado. Normalmente utiliza-se fórmulas de 3 a 5 pontos.

Aplicação das F.D.F. em MATLAB

INPUT:

- f - função
- [a, b] - intervalo de derivação
- h - passo da discretização
- y - vetor das ordenadas (opcional)

OUTPUT:

- [x, y] - malha de pontos

- dydx - derivada de f

CÓDIGO:

- Varia de método para método

2.1 Fórmulas de Diferenças Finitas em 2 pontos

O método mais simples para aproximar a derivada é usar o método de diferenças finitas. Uma diferença finita é uma expressão da forma $f(x+b)-f(x+a)$, que ao ser dividida por $(b-a)$ passa a ser chamada de quociente de diferenças. A técnica de **diferenças finitas** consiste em aproximar a derivada de uma função via fórmulas discretas que requerem apenas um conjunto finito de pares ordenados, onde geralmente denotamos $y_i = f(x_i)$.

2.1.1 Progressivas

Fórmula:

$$f'(x_k) := \frac{f(x_{k+1}) - f(x_k)}{h}$$

onde:

- $f'(x_k)$ → Aproximação do valor da derivada no ponto de abcissa x_k;
- $f(x_{k+1})$ → Valor da função na próxima abcissa;
- $f(x_k)$ → Valor da função no ponto de abcissa atual;
- h → Valor de cada sub-intervalo (passo).

Algoritmo:

1. Alocar memória para ***x***;
2. Definir o número de pontos (***n***);
3. Se forem recebidos 4 elementos, ***y*** recebe o valor de ***f(x)***;
4. Alocar memória para a derivada;
5. Para ***i*** de ***1*** a ***n-1***, calcular a derivada (aproximada) de ***f*** no ponto atual, para a iésima iteração;
6. Calcular a derivada (aproximada) de f no ponto atual, em ***n***.

Função (MATLAB):

```
function [x,y,dydx] = DiferencasFinitasProgressivas2P(f,a,b,h,y)

x = a:h:b;
n = length(x);
if nargin == 4
    y = f(x);
end

dydx = zeros(1,n);
for i = 1:n-1
    dydx(i) = (y(i+1)-y(i))/h;
end

dydx(n) = (y(n)-y(n-1))/h;

end
```

2.1.2 Regressivas

Fórmula:

$$f'(x_k) := \frac{f(x_k) - f(x_{k-1})}{h}$$

onde:

- $f'(x_k)$ → Aproximação do valor da derivada no ponto de abcissa x_k;
- $f(x_k)$ → Valor da função no ponto de abcissa atual;
- $f(x_{k-1})$ → Valor da função na abcissa anterior;
- h → Valor de cada sub-intervalo (passo).

Algoritmo:

1. Alocar memória para ***x***;
2. Definir o número de pontos (***n***);
3. Se forem inseridos 4 elementos, ***y*** recebe o valor de ***f(x)***;
4. Alocar memória para a derivada;
5. Calcular a derivada (aproximada) de f no ponto atual, em ***1***;
6. Para ***i*** de ***2*** a ***n***, calcular a derivada (aproximada) de f no ponto atual, para a iésima iteração.

Função (MATLAB):

```
function [x,y,dydx] = DiferencasFinitasRegressivas2P(f,a,b,h,y)

x = a:h:b;
n = length(x);
if nargin == 4
    y = f(x);
end

dydx = zeros(1,n);
dydx(1) = (y(2)-y(1))/h;
for i = 2:n
    dydx(i) = (y(i)-y(i-1))/h;
end
```

2.2 Fórmulas de Diferenças Finitas em 3 pontos

2.2.1 Progressivas

Fórmula:

$$f'(x_k) := \frac{-3f(x_k) + 4f(x_{k+1}) - f(x_{k+2})}{2h}$$

onde:

- $f'(x_k)$ → Aproximação do valor da derivada no ponto de abcissa x_k;
- $f(x_k)$ → Valor da função no ponto de abcissa atual;
- $f(x_{k+1})$ → Valor da função na próxima abcissa;
- $f(x_{k+2})$ →Valor da função 2 abcissas à frente;
- h → Valor de cada subintervalo (passo).

Algoritmo:

1. Alocar memória para ***x***;
2. Definir o número de pontos (***n***);
3. Se forem inseridos 4 elementos, ***y*** recebe o valor de ***f(x)***;
4. Alocar memória para a derivada;
5. Para ***i*** de ***1*** a ***n-2***, calcular a derivada (aproximada) de f no ponto atual, para a iésima iteração;
6. Calcular a derivada (aproximada) de f no ponto atual, em ***n-1***;
7. Calcular a derivada (aproximada) de f no ponto atual, em ***n***.

isec Engenharia

Função (MATLAB):

```
function [x,y,dydx] = DiferencasFinitasProgressivas3P(f,a,b,h,y)
x = a:h:b;
n = length(x);
if nargin == 4
    y = f(x);
end

dydx = zeros(1,n);
for i = 1:n-2
    dydx(i) = (-3*y(i) + 4*y(i+1) - y(i+2))/(2*h);
end

dydx(n-1) = (y(n-3) - 4*y(n-2) + 3*y(n-1))/(2*h);
dydx(n) = (y(n-2) - 4*y(n-1) + 3*y(n))/(2*h);

end
```

2.2.2 Regressivas

Fórmula:

$$f'(x_k) := \frac{f(x_{k-2}) - 4f(x_{k-1}) + 3f(x_k)}{2h}$$

onde:

- $f'(x_k)$ → Aproximação do valor da derivada no ponto de abcissa x_k;
- $f(x_{k-2})$ → Valor da função 2 abcissas atrás;
- $f(x_{k-1})$ → Valor da função na abcissa anterior;
- $f(x_k)$ → Valor da função no ponto de abcissa atual;
- h → Valor de cada subintervalo (passo).

Algoritmo:

1. Alocar memória para ***x***;
2. Definir o número de pontos (***n***);
3. Se forem inseridos 4 elementos, ***y*** recebe o valor de ***f(x)***;
4. Alocar memória para a derivada;
5. Calcular a derivada (aproximada) de f no ponto atual, em ***1***;
6. Calcular a derivada (aproximada) de f no ponto atual, em ***2***;
7. Para ***i*** de ***3*** a ***n***, calcular a derivada (aproximada) de f no ponto atual, para a iésima iteração.

```
function [x,y,dydx] = DiferencasFinitasRegressivas3P(f,a,b,h,y)
x = a:h:b;
n = length(x);
if nargin == 4
    y = f(x);
end

dydx = zeros(1,n);
dydx(1) = (-3*y(1) + 4*y(2) - y(3))/(2*h);
dydx(2) = (-3*y(2) + 4*y(3) - y(4))/(2*h);
for i = 3:n
    dydx(i) = (y(i-2) - 4*y(i-1) + 3*y(i))/(2*h);
end

end
```

Função (MATLAB):

2.2.3 Centradas

Fórmula:

$$f'(x_k) := \frac{f(x_{k+1}) - f(x_{k-1})}{2h}$$

onde:

- $f'(x_k)$ → Aproximação do valor da derivada no ponto de abcissa x_k;
- $f(x_{k+1})$ → Valor da função na próxima abcissa;
- $f(x_{k-1})$ → Valor da função na abcissa anterior;
- h → Valor de cada subintervalo (passo).

Algoritmo:

1. Alocar memória para ***x***;
2. Definir o número de pontos (***n***);
3. Se forem inseridos 4 elementos, ***y*** recebe o valor de ***f(x)***;
4. Alocar memória para a derivada;
5. Calcular a derivada (aproximada) de f no ponto atual, em ***1***.
6. Para ***i*** de ***2*** a ***n-1***, calcular a derivada (aproximada) de f no ponto atual, para a iésima iteração;
7. Calcular a derivada (aproximada) de f no ponto atual, em ***n***.

Função (MATLAB):

```
function [x,y,dydx] = DiferencasFinitasCentradas3P(f,a,b,h,y)
x = a:h:b;
n = length(x);
if nargin == 4
    y = f(x);
end

dydx = zeros(1,n);
dydx(1) = (-3*y(1) + 4*y(2) - y(3))/(2*h);
for i = 2:n-1
    dydx(i) = (y(i+1)-y(i-1))/(2*h);
end

dydx(n) = (y(n-2) - 4*y(n-1) + 3*y(n))/(2*h);

end
```

2.3 2ª Derivada

Fórmula:

$$f''(x_k) := \frac{f(x_{k+1}) - 2f(x_k) + f(x_{k-1})}{h^2}$$

onde:

- $f''(x_k)$ → Aproximação do valor da 2ª derivada no ponto de abcissa x_k;
- $f(x_{k+1})$ → Valor da função na próxima abcissa;
- $f(x_k)$ → Valor da função no ponto de abcissa atual;
- $f(x_{k-1})$ → Valor da função na abcissa anterior;
- h → Valor de cada subintervalo (passo).

Algoritmo:

1. Alocar memória para x;
2. Definir o número de pontos (n);
3. Se forem inseridos 4 elementos, ***y*** recebe o valor de ***f(x)***;
4. Alocar memória para a derivada;
5. Calcular a 1ª derivada no ponto: x = a;
6. Calcuar a 1ª derivada no ponto: x = a + h;
7. Calcular a 1ª derivada no ponto: x = a + h * 2;
8. Calcular a 2ª derivada no ponto: x = a;
9. Para ***i*** de ***2*** a ***n-1***, calcular a derivada (aproximada) de f no ponto atual, para a iésima iteração;
10. Cacular a 1ª derivada no ponto: x = b – h * 2;
11. Calcular a 1ª derivada no ponto: x = b - h;
12. Calcular a 1ª derivada no ponto: x = b;
13. Calcular a 2ª derivada no ponto: x = b.

Função (MATLAB):

```
function [x,y,dydx] = SegundaDerivada(f,a,b,h,y)
x=a:h:b;

n=length(x);

if nargin==4
    y=f(x);
end

dydx=zeros(1,n);

temp1=(-3*y(1) + 4*y(2) - y(3))/(2*h);
temp2=(-3*y(2) + 4*y(3) - y(4))/(2*h);
temp3=(-3*y(3) + 4*y(4) - y(5))/(2*h);

dydx(1)=(-3*temp1 + 4*temp2 - temp3)/(2*h);

for i=2:n-1
    dydx(i)=(y(i+1) - 2*y(i) + y(i-1))/(h*h);
end

tempn2=(y(n-4) - 4*y(n-3) + 3*y(n-2))/(2*h);
tempn1=(y(n-3) - 4*y(n-2) + 3*y(n-1))/(2*h);
tempn=(y(n-2) - 4*y(n-1) + 3*y(n))/(2*h);

dydx(n)=(tempn2 - 4*tempn1 + 3*tempn)/(2*h);

end
```

3. Derivação Simbólica no MATLAB

Para além de cálculo numérico, o MATLAB também permite efectuar cálculo simbólico. Para esse efeito, existe uma toolbox de matemática simbólica ("Symbolic Math Toolbox").

Esta toolbox permite efectuar cálculos de diferentes naturezas, designadamente:

- Cálculo propriamente dito (diferenciação, integração, determinação de limites, somatórios, série de Taylor, etc.);
- Álgebra linear (inversa de uma matriz, determinantes, valores próprios, etc.);
- Simplificação de expressões algébricas;
- Obtenção de soluções analíticas de equações algébricas e diferenciais;
- Transformadas de Laplace, Z e de Fourier, etc., etc.

3.1. Diff (MATLAB)

Como visto anteriormente, a *toolbox* de matemática simbólica disponibiliza um conjunto de funções de cálculo, entre as quais se destaca a derivação que pode ser efetuada utilizando o comando ***diff***.

Este comando permite calcular derivadas de várias ordens, em que em derivadas de ordem superior à primeira é necessária a utilização de mais um parâmetro de entrada.

Exemplo de utilização da função ***diff*** em MATLAB para o cálculo da Derivada Exata:

```
function f = DerivadaExata(strF)

syms x                                  % cria a variável simbólica x
h = @(x) eval(vectorize(strF));

f = matlabFunction(diff(h(x)));         % Calcula, através da função diff, a derivada exata
```

4. Métodos Numéricos para Integração

Na Matemática existe uma grande variedade de algoritmos cujo principal objetivo é aproximar o valor de uma dada integral definida de uma função sem o uso de uma expressão analítica para a sua primitiva.

Normalmente, estes métodos são constituídos pelas seguintes fases:

- Decomposição do domínio em subintervalos (um intervalo contido de subintervalos);
- Integração aproximada da função de cada subintervalo;
- Soma dos resultados numéricos obtidos.

Razões da necessidade de usar a integração numérica:

- Algumas funções não admitem uma primitiva de forma explícita;
- A primitiva da função é muito complicada para ser avaliada;
- Quando não se dispõe de uma expressão analítica para o integrando, mas conhece-se os seus valores em um conjunto de pontos do domínio.

A Integração Numérica de uma função ***f(x)*** num intervalo **[a, b]** consiste no cálculo da área delimitada por essa função, recorrendo à interpolação polinomial, como forma de obtenção de um polinómio $\mathbf{p_n(x)}$.

O método básico envolvido nesta aproximação é chamado de quadratura numérica e consiste na seguinte expressão:

$$\int_a^b f(x)dx \simeq \sum_{i=0}^{n} \alpha_i f(x_i)$$

4.1 Regra dos Trapézios

A Regra dos Trapézios é um método de Integração Numérica utilizado para aproximar o integral definido e pode também ser visto como o resultado da média da parte esquerda e direita da soma de Riemann, e por vezes pode mesmo ser definido desta forma.

Este método está dividido em 3 passos:

- Preencher a parte inferior da função (em cada subintervalo) com trapézios;
- Calcular as suas áreas;
- Somar os resultados do cálculo das várias áreas.

Nota: Quanto mais subintervalos existirem mais precisa se torna a aproximação.

Fórmula:

$$I_T(f) = \frac{h}{2}[f(x_0) + 2f(x_1) + \cdots + 2f(x_{n-1}) + f(x_n)]$$

onde:

- $I_T(f)$ → Cálculo da Regra dos Trapézios;
- $f(x_0)$ → Valor da função na abcissa x_0;
- $f(x_1)$ → Valor da função na abcissa x_1;
- $f(x_{n-1})$ → Valor da função na abcisa x_{n-1};
- $f(x_n)$ → Valor da função na abcissa x_n;
- h → Valor de cada subintervalo (passo).

Algoritmo:

1. Calcular o passo (***h***);
2. Atribuir o valor de ***a*** a ***x*** ;
3. Inicializar ***s*** com o valor ***0***;
4. Para ***i*** de ***1*** a ***n-1***:
   a. Somar o passo (***h***) a ***x***;
   b. Somar o valor da função em x (***f(x)***) a ***s***.
5. Cálculo da Regra dos Trapézios.

Função (MATLAB):

INPUT:

- f – função integranda
- [a, b] - intervalo de derivação
- n - número de subintervalos

OUTPUT:

- area - Valor da área calculada pela Regra dos Trapézios

```
function area = RTrapezios(f,a,b,n)
h=(b-a)/n;
x=a;
s=0;
for i=1:n-1
    x=x+h;
    s=s+f(x);
end
area = (h/2)*(f(a)+2*s+f(b));
end
```

4.2 Regra de Simpson

A Regra de Simpson é um método de Integração Numérica utilizado para a aproximação de integrais definidos e baseia-se em aproximar o integral definido pela área sob arcos de parábola que interpolam a função.

Para conseguir uma melhor aproximação deve-se dividir o intervalo de integração em intervalos mais pequenos, aplicar a fórmula de Simpson para cada um deles e somar os resultados.

Fórmula:

$$I_S(f) = \frac{h}{3}[f(x_0) + 4f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)]$$

onde:

- $I_S(f)$ → Cálculo da Regra de Simpson;
- h → Valor de cada subintervalo (passo);
- $f(x_0)$ → Valor da função na abcissa x_0;
- $f(x_1)$ → Valor da função na abcissa x_1;
- $f(x_{n-1})$ → Valor da função na abcissa x_{n-1};
- $f(x_n)$ → Valor da função na abcissa x_n;
- n → Número de subintervalos.

Algoritmo:

1. Cálculo do passo (***h***);
2. Atribuir o valor de ***a*** a ***x***;
3. Inicializar ***s*** com o valor ***0***;
4. Para ***i*** de ***1*** a ***n-1***:
   a. Somar o passo (***h***) a ***x***;
   b. Se ***i*** for par, soma-se **2** vezes o valor de ***f(x)*** a ***s***;
   c. Se ***i*** for ímpar, soma-se 4 vezes o valor de ***f(x)*** a ***s***;
5. Calcular a Regra de Simpson.

Função (MATLAB):

INPUT:

- f – função integranda
- [a, b] - intervalo de derivação
- n - número de subintervalos

OUTPUT:

- area - Valor da área calculada pela Regra de Simpson

```
function S = RSimpson(f,a,b,n)
h = (b-a)/n;
x = a:h:b;
s = 0;
for i = 2:n-1
    if mod(i,2)==0
        s = s+2*f(x(i));
    else
        s = s+4*f(x(i));
    end
end
S = h/3*(f(a)+s+f(b));
end
```

5. Integração Simbólica no Matlab

Como já referido anteriormente, através da *toolbox* de Matemática simbólica é possível efetuar cálculos de diferentes naturezas. Uma das diversas possibilidades de cálculo é a de Integração.

5.1 Int (MATLAB)

A função ***int*** pode ser usada para calcular integrais definidos e indefinidos.

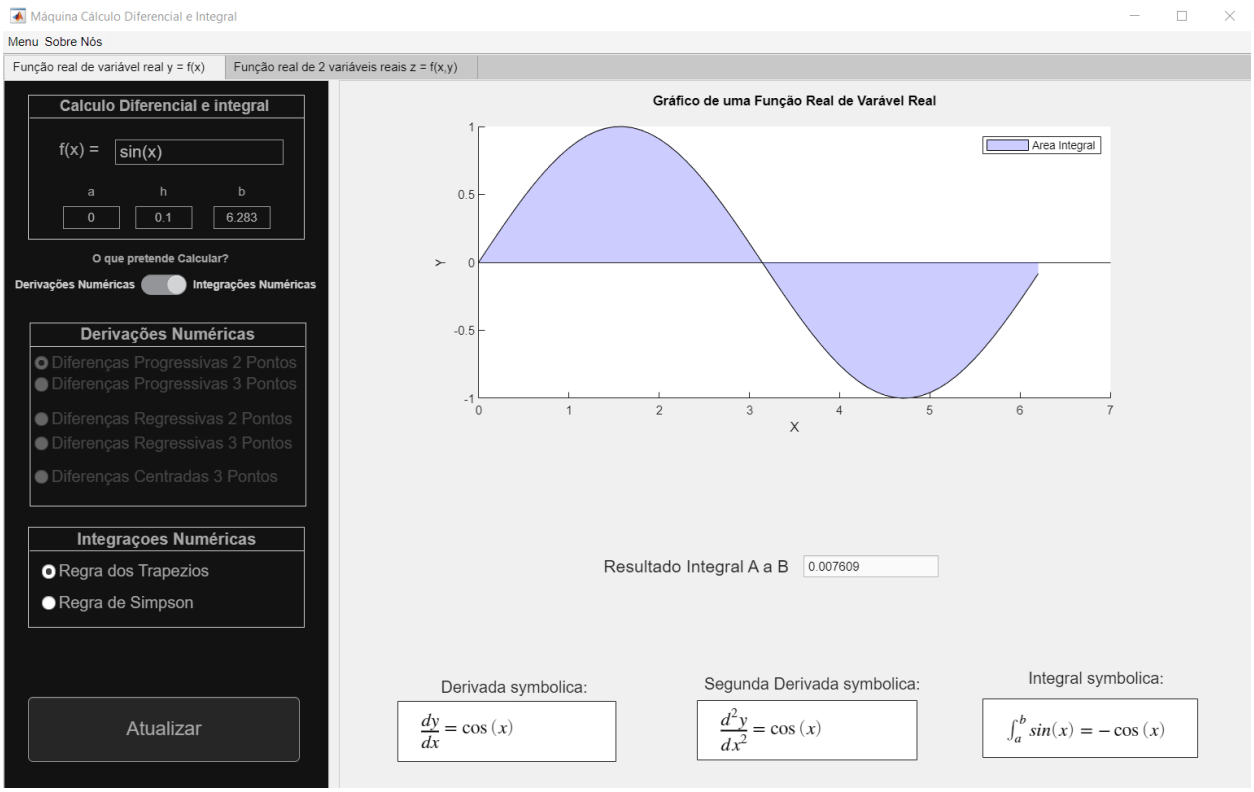
Exemplo de utilização da função ***int*** em MATLAB para o cálculo da solução exata do integral:

```
sExata=int(f(sym('x')),a,b)
```

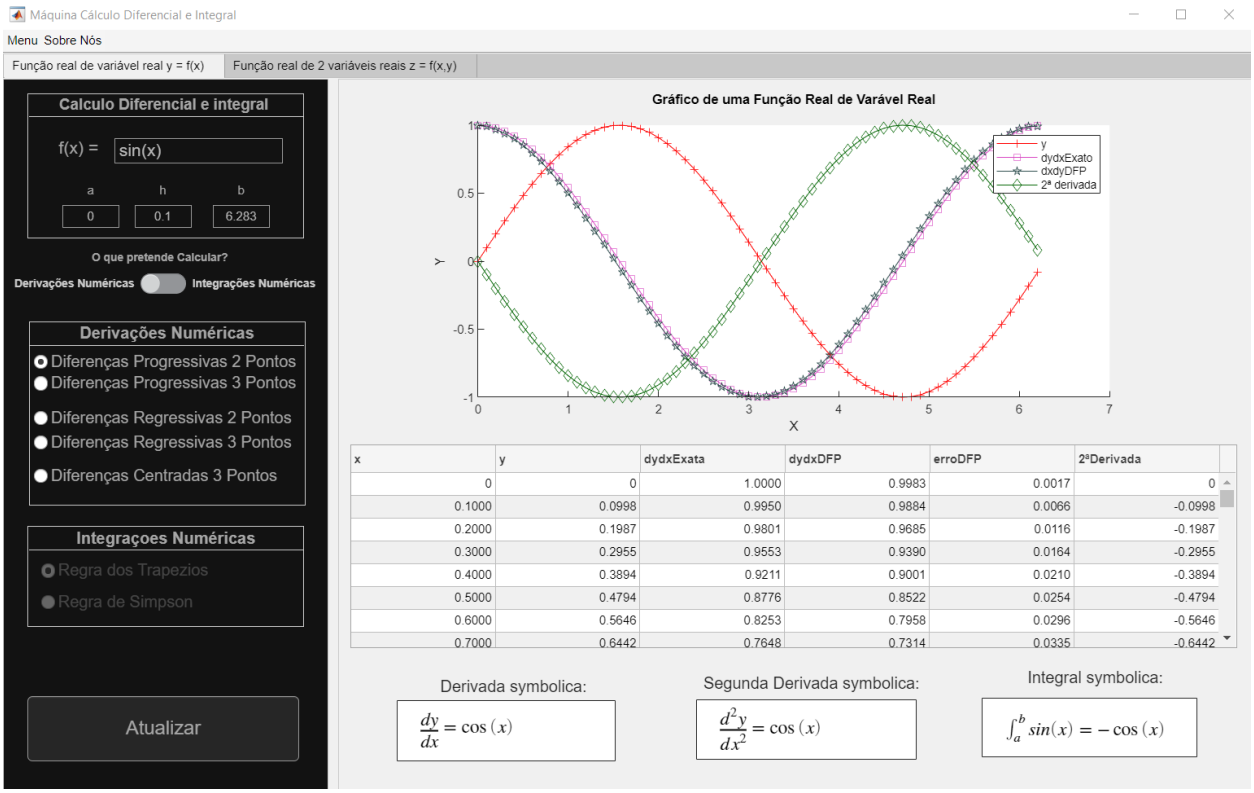
Onde *a* e *b* são os extremos do intervalo de integração [a, b] e *f* a função integranda.

6. Exemplos de aplicação

6.1 Integração

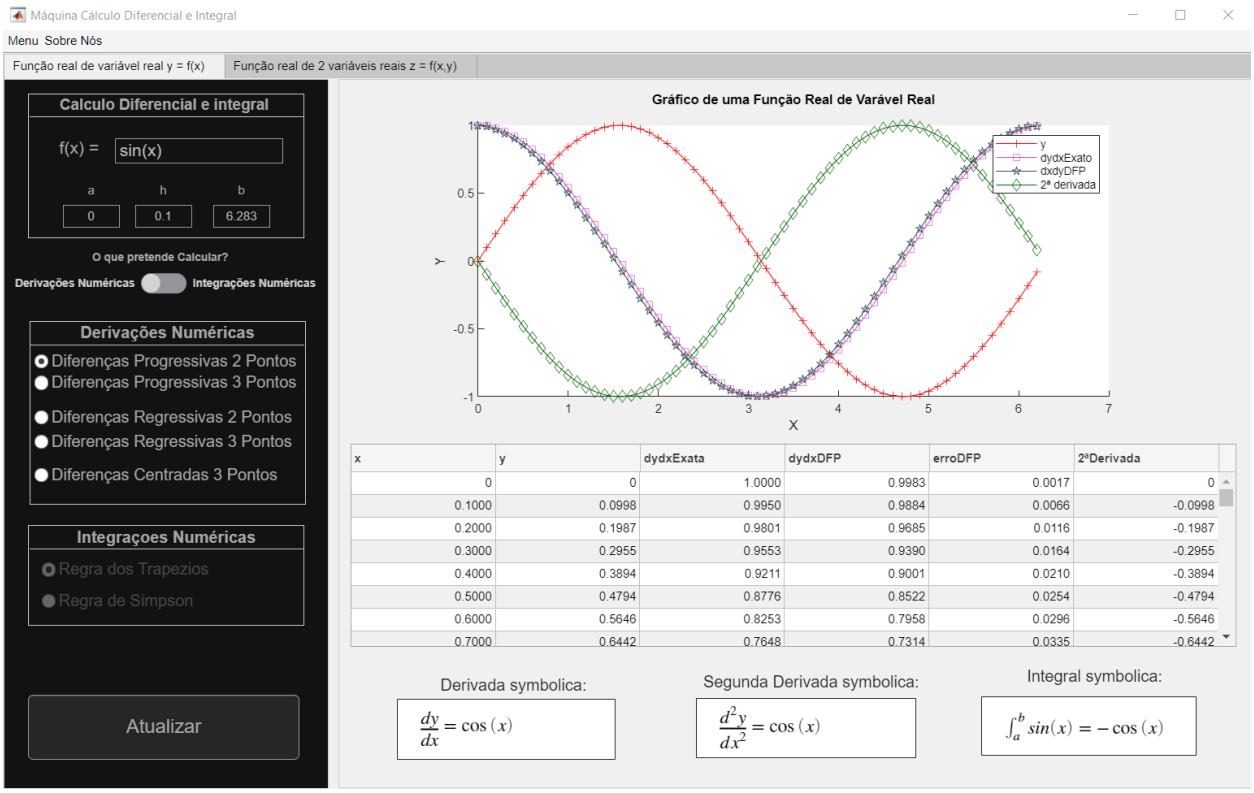


6.2 Derivação



x	y	dydxExata	dydxDFP	erroDFP	2ªDerivada
0	0	1.0000	0.9983	0.0017	0
0.1000	0.0998	0.9950	0.9884	0.0066	-0.0998
0.2000	0.1987	0.9801	0.9685	0.0116	-0.1987
0.3000	0.2955	0.9553	0.9390	0.0164	-0.2955
0.4000	0.3894	0.9211	0.9001	0.0210	-0.3894
0.5000	0.4794	0.8776	0.8522	0.0254	-0.4794
0.6000	0.5646	0.8253	0.7958	0.0296	-0.5646
0.7000	0.6442	0.7648	0.7314	0.0335	-0.6442

6.3 Função real de 2 Variáveis reais



x	y	dydxExata	dydxDFP	erroDFP	2ªDerivada
0	0	1.0000	0.9983	0.0017	0
0.1000	0.0998	0.9950	0.9884	0.0066	-0.0998
0.2000	0.1987	0.9801	0.9685	0.0116	-0.1987
0.3000	0.2955	0.9553	0.9390	0.0164	-0.2955
0.4000	0.3894	0.9211	0.9001	0.0210	-0.3894
0.5000	0.4794	0.8776	0.8522	0.0254	-0.4794
0.6000	0.5646	0.8253	0.7958	0.0296	-0.5646
0.7000	0.6442	0.7648	0.7314	0.0335	-0.6442

7. Conclusão

Com este trabalho podemos concluir que os métodos numéricos têm diversas aplicações, quer para derivadas, quer para integrais, o que facilita a resolução de problemas mais densos de variadas áreas da ciência (e não só), onde, por vezes, necessitamos de calcular integrais analiticamente, algo que pode não ser possível.

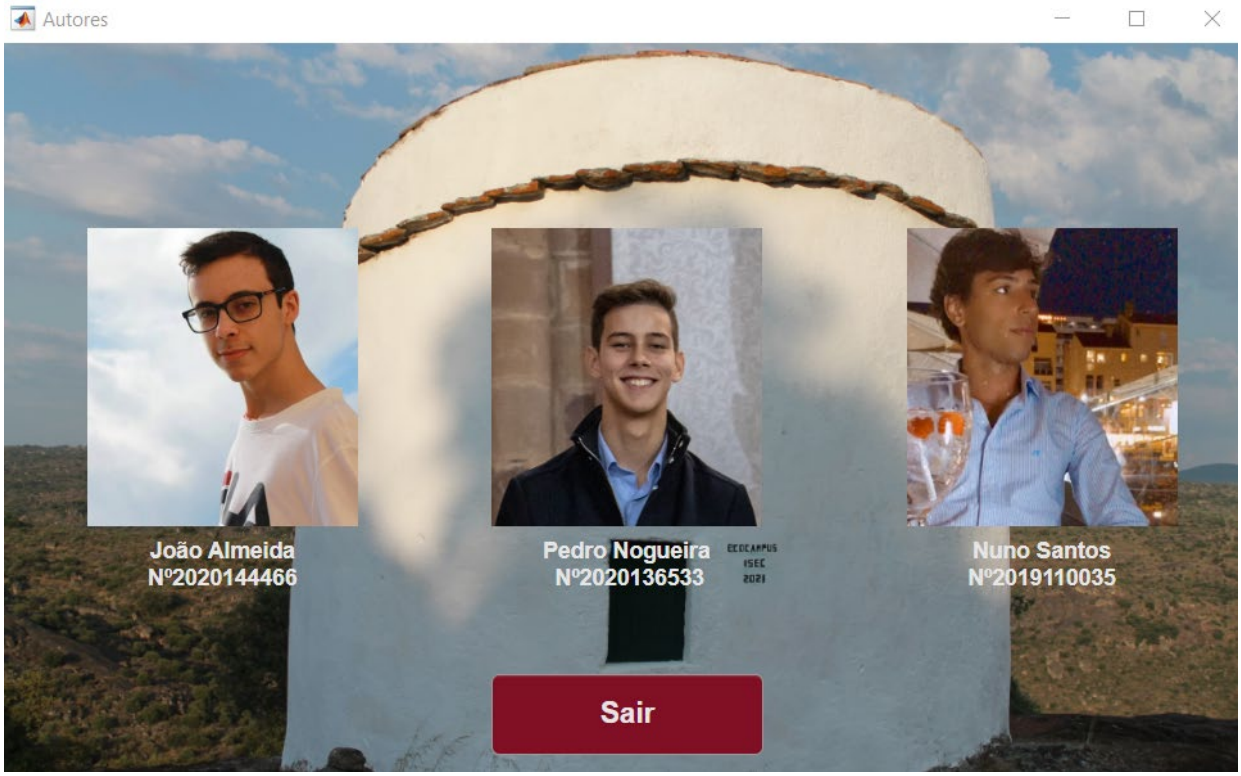
Como já visto anteriormente, verificamos que quanto maior for o número de subintervalos ***n***, menor é o erro dos métodos / fórmulas. A introdução do tamanho de cada subintervalo ***h*** tem o efeito contrário: quanto menor o tamanho introduzido, menor o erro dos métodos / fórmulas.

Relativamente à comparação entre os vários métodos da integração numérica, verificamos que o que apresenta menor erro é, consequentemente, melhor aproximação ao valor exato, é em regra geral, o Quad do MATLAB. O erro da Quad pode ser ajustado, mas na nossa GUI o seu valor é sempre aproximadamente $8.6 * 10^{-9}$.

Para valores elevados de ***n*** e grandes intervalos (ou seja, pequeno ***h***), a Regra de Simpson é claramente melhor do que a Regra dos Trapézios, mas, para valores pequenos de ***n*** e pequenos intervalos (ou seja, grande ***h***), não se nota tanto essa diferença, sendo que por vezes a Regra dos Trapézios obtém erros menores. Para valores suficientemente grandes de ***n***, como o erro da Quad é constante, temos que a Regra de Simpson obtém erros menores, na nossa GUI.

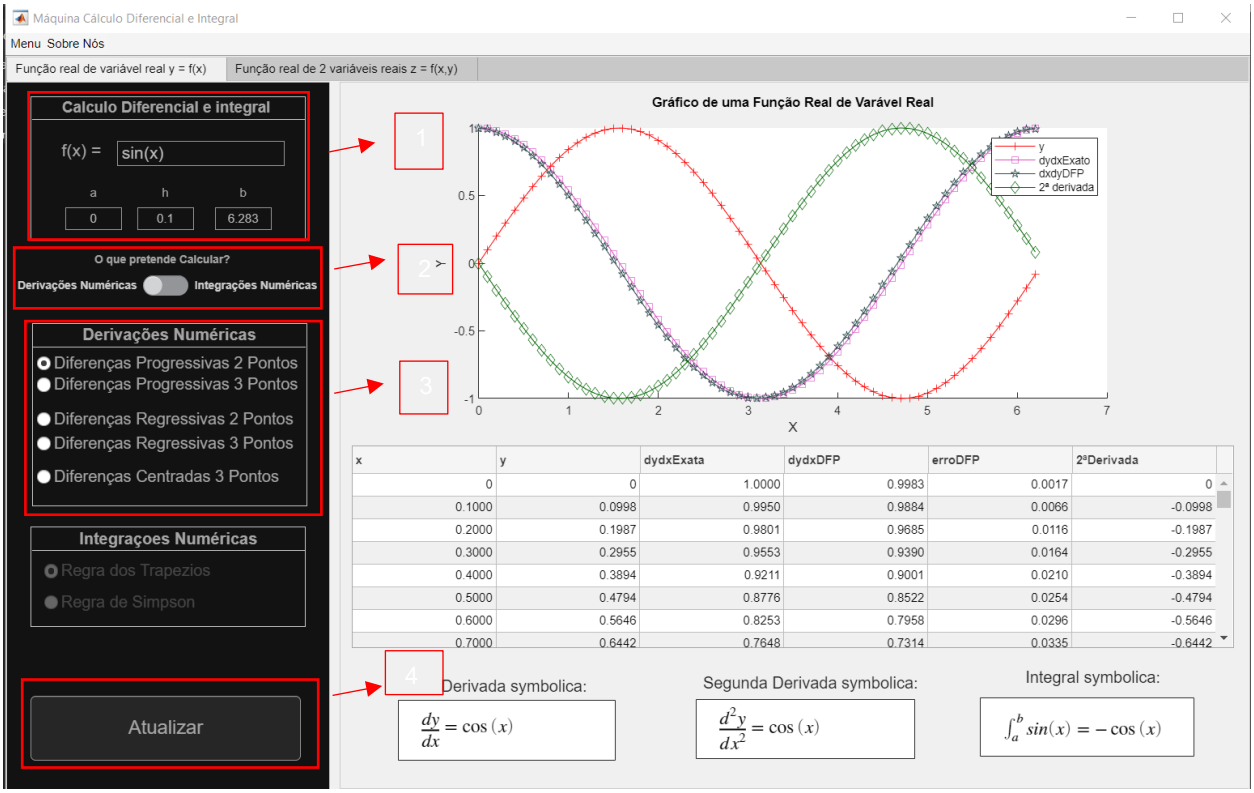
Em relação às fórmulas da derivação numérica, a comparação encontrada foi que a melhor aproximação ao valor real se origina a partir das fórmulas que utilizam 3 pontos, comparativamente às que apenas utilizam 2 pontos (que muitas vezes apresentam o dobro do erro, ou mais)

GUI **Autores**


Autores
João Almeida
Nº2020144466
Pedro Nogueira
Nº2020136533
Nuno Santos
Nº2019110035
Sair

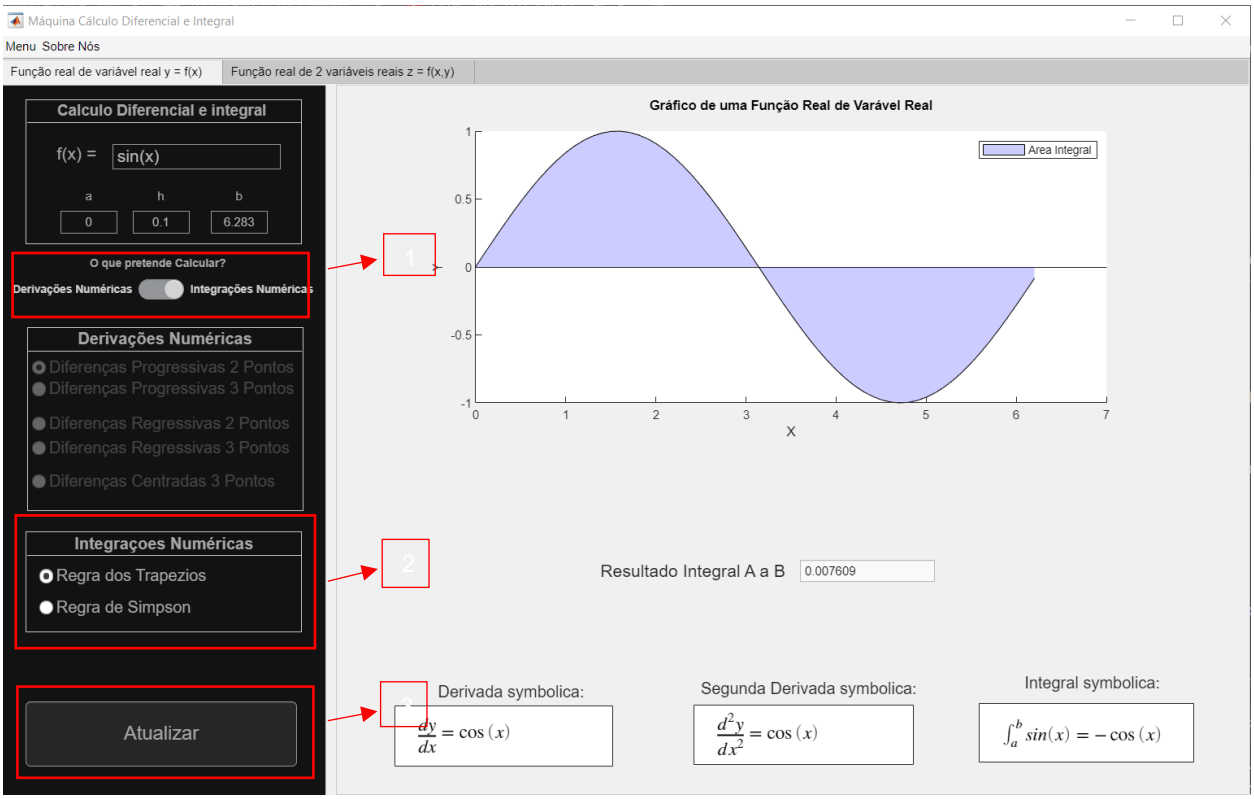
8 Breve Introdução da Aplicação

8.1 Função real de Variável real



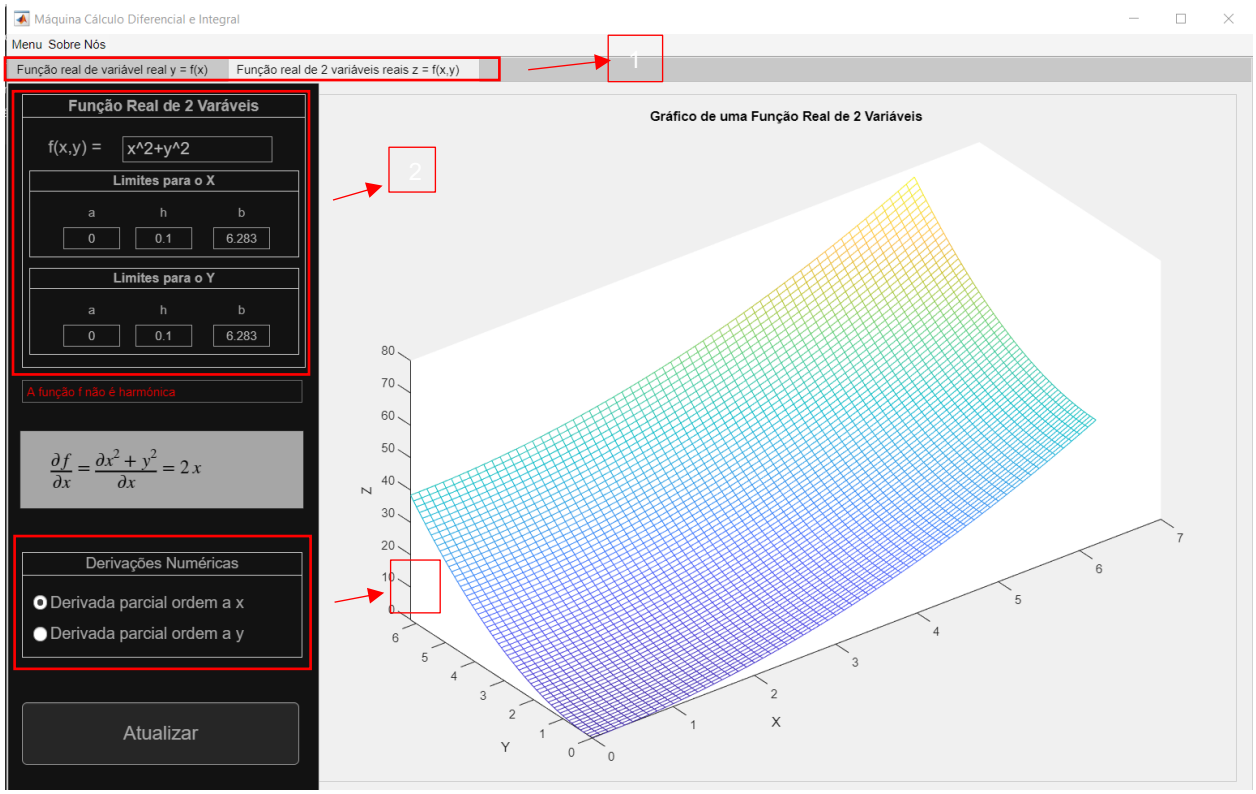
A aplicação obtém os dados fornecidos pelo utilizador (1) e é possível calcular as derivações numéricas ou as integrações numéricas alterando apenas a posição do cursor (2).

Escolhendo as derivações numéricas o programa libertará a opção de escolher qual a fórmula de cálculo (3), utilizando o botão de atualizar (4) é mostrada toda a informação sobre a função.



Optando pelo cálculo das Integrações numéricas (1), é possível optar por calcular a regra dos trapézios e a regra de simpson (2), ao efetuar o cálculo (3) a aplicação é adaptada para uma melhor perceção sobre as integrações.

8.2 Função real de 2 Variáveis reais



Utilizando a 2 página (1) da aplicação, é apresentada informação para calcular a função.

A aplicação obtém os dados fornecidos pelo utilizador (2) e é possível calcular as Derivações Numéricas, dependendo da escolha do utilizador (3), o programa indicará a sua derivada, o gráfico da função e se a função é harmónica ou não.

8 Bibliografia

- Moodle: Atividade03Trabalho – MáquinaCDI (10-06-2021)
  https://moodle.isec.pt/moodle/mod/assign/view.php?id=141724

- Mathworks Answers (15-06-2021)
  https://www.mathworks.com/matlabcentral/answers/index

- WolframAlpha (17-06-2021)
  https://www.wolframalpha.com/