

# Projeto II

**Aplicação de métodos de Integração, interpolação numérica e  
análise de dados**

Matheus Araujo Souza      RA:184145      MS211

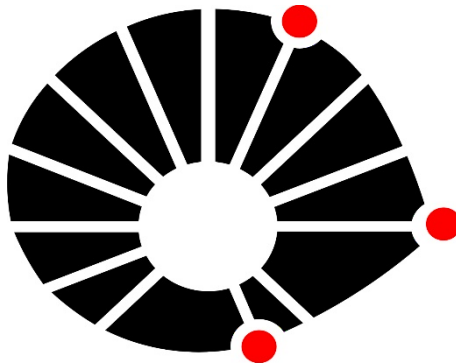
Supervisionado por

Giusepp Romanazzi

Assistant Professor

IMECC, instituto de Matemática Estatística e Computação Científica

Unicamp, Campinas, Brasil.



**UNICAMP**

Novembro de 2019

---

Universidade Estadual de Campinas (Unicamp)

# SUMÁRIO

Introdução-----	3
Objetivo -----	4
Análise de dados/ código I-----	5
Análise de dados/ código II-----	9
Análise de dados/ código III-----	12
Conclusão -----	15

# LISTA DE FIGURAS

1.1-----	5
1.2-----	7
1.3-----	8
1.4-----	8
1.5-----	8
1.6-----	12
1.7-----	12
1.8-----	13
1.9-----	15

## Introdução

Diferente do método dos mínimos quadrados, cujo o único objetivo é encontrar a melhor função que se aproxima à verdadeira, por uma combinação de funções conhecidas, o que não nos oferece nenhuma espécie de erro de aproximação. A interpolação polinomial é um dos métodos que até hoje pode se dizer, o mais utilizado, vamos então ligando ponto a ponto e aproximando a nossa função  $f(x)$ , aqui nosso único trabalho é encontrar o polinômio que passa por um conjunto de pontos. Existem dois casos em que a interpolação normalmente é requisitada, quando a nossa  $f(x)$  não é conhecida e sabemos apenas o valor de alguns pontos, isso ocorre frequentemente na prática, quando trabalhamos coletando dados experimentais a outra ocasião é quando estamos trabalhando com uma  $f(x)$  muito complicada, acabamos então escolhendo o sacrifício da precisão por uma facilitação do nosso trabalho.

Vamos imaginar agora a situação de se calcular uma integral avaliada em dois pontos, existem algumas funções em que o cálculo da primitiva acaba sendo muito complicado, se não impossível e isso ocorre com mais frequência do que se imagina, também precisamos ressaltar uma classe de funções que não tem primitivas definidas. Para esses dois casos apresentados serão necessários métodos numéricos, cujo alguns serão utilizados nos problemas que iremos solucionar daqui para frente.

## Objetivo

Dado os seguintes dados do tipo  $f(x, f(x))$

X	0	0.5	1	2	2.5	3	3.5
f(x)	2	1.6	0.8	0.04	0.004	$2 \cdot 10^{-4}$	$9 \cdot 10^{-6}$

Devemos então escrever um programa que automaticamente determine uma aproximação de  $f$  no ponto 1.5 usando polinômios de interpolação de ordem 1, 2 e 3, devemos também dar como opção ao usuário a escolha do polinômio para a interpolação. Para a análise do erro vamos determinar qual é o polinômio interpolador para que o erro  $|f(1.5) - p(1.5)| = \min |f(1.5) - p(1.5)|$ , isso entre todos os pontos interpoladores  $p$ . Exibindo então para o usuário os valores dos nós de interpolação, e o valor para  $p(1.5)$ , e o erro esperado.

No final escreveremos um código que aproxima a integral dos pontos interpolados, isso é a integral de  $f$  em  $[0, 3.5]$ , usando uma forma de integração repetida, também daremos como output o valor achado para a integral e o erro esperado.

## Análise de dados/ código I

### Metodologia

Para a interpolação do nosso conjunto de pontos, vamos empregar o método de Newton, a forma de newton para polinômio  $p_n(x)$  que interpola  $f(x)$  em  $x_0, x_1, \dots, x_n, (n+1)$  pontos distintos é o seguinte,  $p_n(x) = d_0 + d_1(x-x_0) + d_2(x-x_0)(x-x_1) + \dots + d_n(x-x_0)(1-x_1) \dots (x-x_{n-1})$ , em que  $d_k$ , para  $k=0, 1, \dots, n$  são as diferenças divididas de ordem  $k$  entre os pontos  $(x_j, f(x_j))$ ,  $j=0, 1, \dots, k$ .

Quando trabalhamos com o nosso problema podemos perceber que o calculo de todas as diferenças divididas não são necessárias, o próprio usuário vai ditar qual é o intervalo desejado, mas estamos com uma restrição da posição de início, sabemos que os melhores pontos para iniciar a aplicação do método, são aqueles que ficam entre nosso ponto desejado que no caso é  $f(1.5)$ , segue abaixo o gráfico da distribuição de pontos.

Gráfico Y vs X

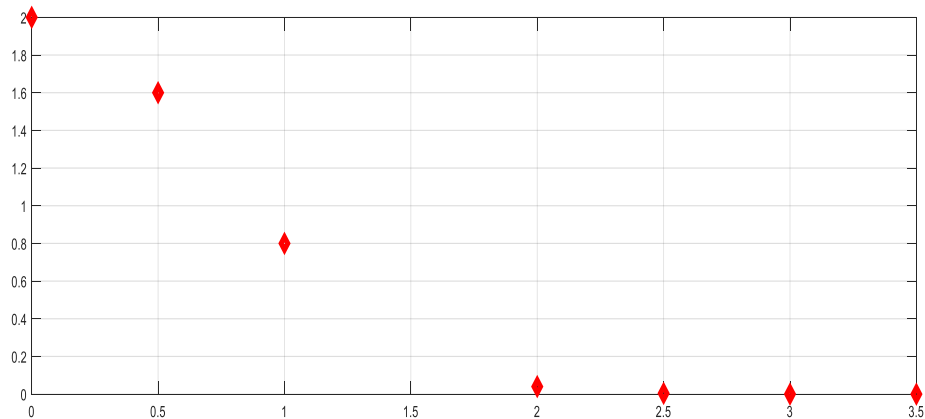


Figura 1.1

No nosso código implementado definimos como  $x_0=1$  e  $x_1=2$ , podemos observar que o ponto desejado em particular se encontra entre eles, esse método facilita os cálculos e beneficia na aproximação dos resultados.

## Código

Segue abaixo o código implementado para a resolução do problema, nele foram feitos alguns comentários e ideias de próximos procedimentos que estavam sendo executadas no momento do seu desenvolvimento.

O ambiente de desenvolvimento integrado de código aberto e multiplataforma que foi escolhido para a criação dos códigos foi o matlab2018.

```
%interpolação de newton

%carregando primeiro os dados
X=[0 0.5 1 2 2.5 3 3.5];
y=[2 1.6 0.8 0.04 0.004 0.0002 0.000009];

%nosso objetivo agora é escrever um programa que determine uma
aproximação
%de f no ponto 1.5

%calculando as diferenças divididas
%vamos reduzir o tamanho da nossa matriz para diminuição de uso da
memória
DifereDivi=zeros(7,5);
DifereDivi(:,1)=y;

%depois de criarmos nossa matriz das diferenças divididas vamos
efetuar os
%cálculos correspondentes a cada uma.
n=input('Dentre os graus 1,2 e 3 qual vamos usar na interpolação?,
n=');
for i=2:n+1
    for j=i+2:n+3
        DifereDivi(j,i)=[DifereDivi(j,i-1)-DifereDivi(j-1,i-1)]/[X(j)-
X(j-i+1)];
    end
end
%quando executamos esse laço sempre fazemos o cálculo até o n+1
diferenças
%divididas, sempre evitando cálculos desnecessários nas nossas
interações,
%não precisamos fazer os cálculos das diferenças divididas que tem em
baixo
%agora que já calculamos as diferenças divididas vamos calcular o
polinômio
%de newton
%como queremos calcular o melhor polinômio que passa pelo 1.5 usando
%polinômios de interpolação dada a nossa função de newton nosso f(X0)
vai
%ser o DifereDivi(3,1)
syms x
```

```
%com esse comando 'syms' é possível manipular variáveis simbólicas
polnewton=DifereDivi(3,1);
P=1;
for il=1:n
    P=P*(x-X(il+2));
    polnewton=polnewton+P*DifereDivi(il+3,il+1);
end
%para deixar o polinomio em uma melhor forma de visualização vamos
usar o
%comando expand
polnewton;
polnewton=expand(polnewton);
```

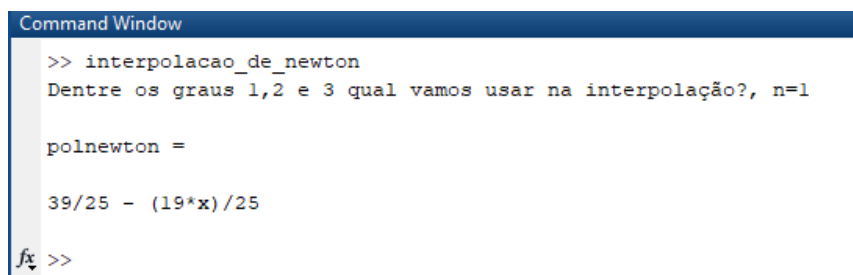
## Análise do Código

No desenvolvimento do nosso código foi necessária a utilização de intervalos para a redução de operações no cálculo das diferenças divididas, reduzindo então o tamanho do nosso vetor de armazenamento dos dados e ainda limitando para poucas operações no for, o usuário pode escolher qual o grau do polinômio que deseja receber no nosso intervalo estabelecido, também podemos notar que os pontos para interpolação, isso é os nós, são aqueles que contêm o intervalo do número especificado e pontos adiante, essa escolha não foi feita premeditadamente, podem acontecer casos que pontos anteriores e posteriores podem ajudar ou não na aproximação dos resultados.

## Análise de Dados

O comando `expand` foi utilizado de uma forma que conseguíssemos arrumar melhor os dados recebidos, mostrando então para o usuário não apenas multiplicações, mas um polinômio da forma simplificada, ainda assim existem limitações no nosso recurso de simplificação, não conseguimos reduzir os valores que acompanham as nossas variáveis. Podemos verificar que existe uma reta formada no intervalo do valor calculado, isso irá facilitar no cálculo do erro mais a frente. Segue abaixo os dados recebidos.

### Polinômio de Grau 1



```
Command Window
>> interpolacao_de_newton
Dentre os graus 1,2 e 3 qual vamos usar na interpolação?, n=1

polnewton =

39/25 - (19*x)/25

fx >>
```

Figura 1.2



## Polinômio de Grau 2

```
Command Window
>> interpolacao_de_newton
Dentre os graus 1,2 e 3 qual vamos usar na interpolação?, n=2

polnewton =

(172*x^2)/375 - (267*x)/125 + 929/375
fx >> |
```

Figura 1.3

## Polinômio de Grau 3

```
Command Window
>> interpolacao_de_newton
Dentre os graus 1,2 e 3 qual vamos usar na interpolação?, n=3

polnewton =

- (2957*x^3)/15000 + (15429*x^2)/10000 - (120263*x)/30000 + 3463/1000
fx >>
```

Figura 1.4

Também podemos fazer a análise gráfica dos polinômios adquiridos, confirmando os comentários já colocados e dando uma visão da aproximação de cada  $p_n(x)$  com o nosso conjunto de dados.

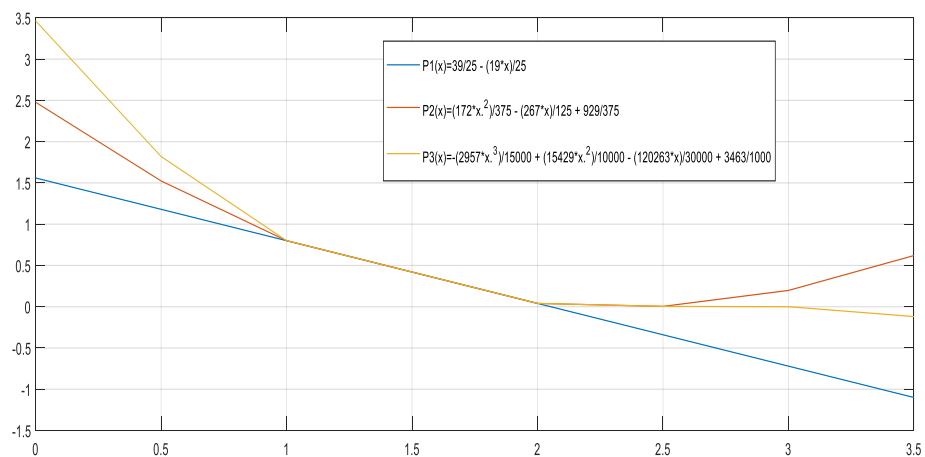


Figura 1.5

## Análise de dados/ código II

### Metodologia

Ao se aproximar uma função  $f(x)$  por um polinômio interpolador de grau  $\leq n$ , comete-se um erro, ou seja

$$Em(x) = F(x) - P_n(x) \text{ para todo } x \text{ no intervalo } [x_0, x_n]$$

Podemos perceber então a importância do estudo erro, estamos interessados o quão próximo  $f(x)$  está de  $p_n(x)$ . Como temos que a nossa  $f(x)$  é dada na forma de tabela, o valor absoluto do erro  $|E_n(x)|$  só pode ser estimado. Isto porque nesse caso, não é possível calcular  $M_{n+1}$ ; mas, se construirmos e analisarmos a tabela das diferenças divididas, podemos estimar nossa derivada usando o maior valor em módulo até a ordem  $n+1$ , nesse caso nosso erro fica como

$$|E_n(x)| \approx |(x - x_0)(x - x_1) \dots (x - x_n)|(\text{máx}|diferenças\ divididas\ de\ ordem\ n + 1|).$$

### Código

Segue abaixo o código implementado para a resolução do problema.

```
%vamos calcular a função que faz todas as diferenças divididas
%quando não temos a função que desejamos precisamos calcular uma
aproximação
%para nossa derivada então calculamos as diferenças divididas até a
ordem
%de n + 1
erro=0;
X=[0 0.5 1 2 2.5 3 3.5];
y=[2 1.6 0.8 0.04 0.004 0.0002 0.000009];
n=6;
%fixando nosso ponto de aproximação
pontodep=1.5;
DifereDivi=zeros(7,7);
DifereDivi(:,1)=y;
for i=2:n+1
    for j=i:n+1
        DifereDivi(j,i)=[DifereDivi(j,i-1)-DifereDivi(j-1,i-1)]/[X(j)-
X(j-i+1)];
    end
end
```

```

end
erro=zeros(1,9);
guardaN=zeros(9,7);
%depois de efetuar o calculo das diferenças divididas como ela
aproxima até
%o grau de n + 1 do grau do meu polinômio então só podemos pegar até 6
%pontos
%não precisamos recalcular a ordem do polinômio apenas fazer as
diferenças
%dos pontos já destacados, se queremos encontrar o melhor polinômio
%primeiro temos que verificar os melhores pontos, para a aproximação,
claro
%que nosso ponto que quer ser aproximado deve estar entre ele
    erro(1,1)=(pontodep-X(1,3))*(pontodep-
X(1,4))*max(abs(DifereDivi(:,3)));
    guardaN(1,:)=[3 4 0 0 0 0 0];
    erro(1,2)=(pontodep-X(1,3))*(pontodep-X(1,4))*(pontodep-
X(1,2))*max(abs(DifereDivi(:,4)));
    guardaN(2,:)=[2 3 4 0 0 0 0];
    erro(1,3)=(pontodep-X(1,3))*(pontodep-X(1,4))*(pontodep-
X(1,2))*(pontodep-X(1,1))*max(abs(DifereDivi(:,5)));
    guardaN(3,:)=[1 2 3 4 0 0 0];
    erro(1,4)=(pontodep-X(1,3))*(pontodep-X(1,4))*(pontodep-
X(1,2))*(pontodep-X(1,5))*max(abs(DifereDivi(:,5)));
    guardaN(4,:)=[2 3 4 5 0 0 0];
    erro(1,5)=(pontodep-X(1,3))*(pontodep-X(1,4))*(pontodep-
X(1,5))*max(abs(DifereDivi(:,4)));
    guardaN(5,:)=[3 4 5 0 0 0 0];
    erro(1,6)=(pontodep-X(1,3))*(pontodep-X(1,4))*(pontodep-
X(1,5))*(pontodep-X(1,6))*(pontodep-X(1,7))*max(abs(DifereDivi(:,6)));
    guardaN(6,:)=[3 4 5 6 7 0 0];
    erro(1,7)=(pontodep-X(1,3))*(pontodep-X(1,4))*(pontodep-
X(1,5))*(pontodep-X(1,6))*max(abs(DifereDivi(:,5)));
    guardaN(7,:)=[3 4 5 6 7 0 0];
    erro(1,8)=(pontodep-X(1,3))*(pontodep-X(1,4))*(pontodep-
X(1,5))*(pontodep-X(1,6))*(pontodep-X(1,2))*(pontodep-
X(1,7))*max(abs(DifereDivi(:,7)));
    guardaN(8,:)=[ 2 3 4 5 6 7 0];
    erro(1,9)=(pontodep-X(1,3))*(pontodep-X(1,4))*(pontodep-
X(1,5))*(pontodep-X(1,6))*(pontodep-X(1,2))*(pontodep-
X(1,1))*max(abs(DifereDivi(:,7)));
    guardaN(9,:)=[1 2 3 4 5 6 0];
    %agora vamos definir qual é o menor erro e assim vamos localizar os
    %pontos de interpolação referentes a ele então imprimir na tela do
    %usuário
    %assim pegamos o índice e o menor valor
    [m1,m2]=min(abs(erro));
    guardaN(m2,:);
    a=guardaN(m2,:);
    a(a==0)=[];
    %assim já temos o menor valor e os pontos de interpolação
    %montando nosso polinômio com esses pontos

```

```

syms x
%com esse comando 'syms' é possível manipular variáveis simbólicas
[k,j]=min(a);
[g,h]=max(a);
polnewton=DifereDivi(j,j);
P=1;
%vamos fazer essas operações para controlar o nosso novo polinômio
for il=j:h-1
P=P*(x-X(il));
polnewton=polnewton+P*DifereDivi(il+1,il+1);
end

polnewton;
polnewton=expand(polnewton)
%vamos exibir os pontos de interpolação
Exib=zeros(1,h);
for i=j:h
    Exib(1,i)=X(1,i);
end
Exib
ml

```

## Análise do Código

O principal problema enfrentado nessa parte, foi o desenvolvimento e a manipulação de todas as possíveis combinações para a dupla do intervalo que contém o ponto 1.5, existem alguns estudos que falam sobre o efeito de pontos anteriores e pontos a frente, no erro da aproximação de  $P_n(x)$  a  $f(x)$ , encontrar um único laço que pudesse fazer todas as operações seria a maneira mais inteligente porém a mais complexa, a maneira implementada de pegar caso a caso pode funcionar para grupos pequenos, mas quando trabalhamos com conjuntos maiores dados isso se torna praticamente inviável ou até mesmo impossível.

## Análise de Dados

Encontramos então nossos dados, como já foi especificado anteriormente a função `expand` não realiza a simplificação completa, mas ainda é possível visualizar todos os dados e os nós de interpolação da nossa saída, dentre todas as combinações que podem ser realizadas não fica claro se podemos logo de início qual é o conjunto que pontos que devem ser interpolados para realizar o menor erro, mesmo que o processo de interpolação seja realizado manualmente, o resultado de número de pontos selecionados não surpreende, entretanto não é sempre que com uma quantidade elevada de nós de interpolação que vamos esperar um alto erro.

Segue abaixo os dados recebidos.

```
Command Window

polnewton =

(2921*x^5)/37500 - (14243*x^4)/18750 + (398569*x^3)/150000 - (540359*x^2)/150000 + (427*x)/1000 + 2

Exib =

    0    0.5000    1.0000    2.0000    2.5000    3.0000

ml =

    0.0060

fx >> |
```

Figura 1.6

## Análise de dados/ código III

### Metodologia

A integração numérica consiste de vários métodos, dentre eles escolhemos a Regra dos trapézios, imagine que queremos integrar uma função  $f$  em um intervalo  $[a, b]$  uma ideia simples para a realização do calculo dessa integral é a aproximação por uma área de um trapézio.

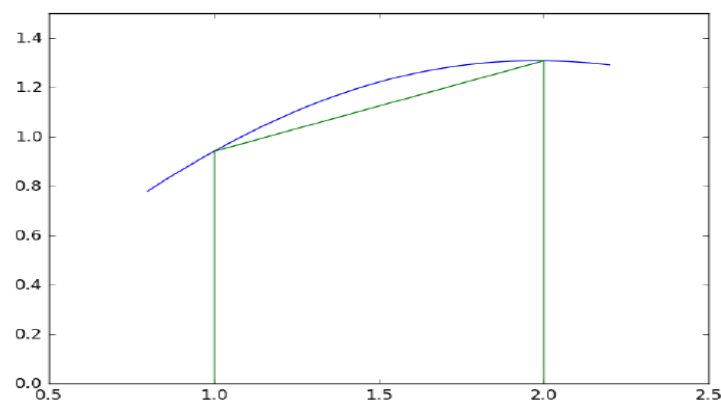


Figura 1.7 (ilustrativa)

No nosso problema não vamos apenas fazer uma simples implementação como na imagem ilustrada, vamos subdividir em pequenos subintervalos, isso é pegando o intervalo [a,b] e dividir em n intervalos menores, se o intervalo de integração é grande a formula dos trapézio nos fornece resultados que não se aproximam muito da integral exata.

$$\int_{x_0}^{x_m} f(x)dx = \frac{h}{2}[f(x_0) + 2f(x_1) + 2f(x_2) + \dots + 2f(x_{m-1}) + 2f(x_m)] - \frac{mh^3 f''(\xi)}{12}$$

Temos então que graficamente podemos observar

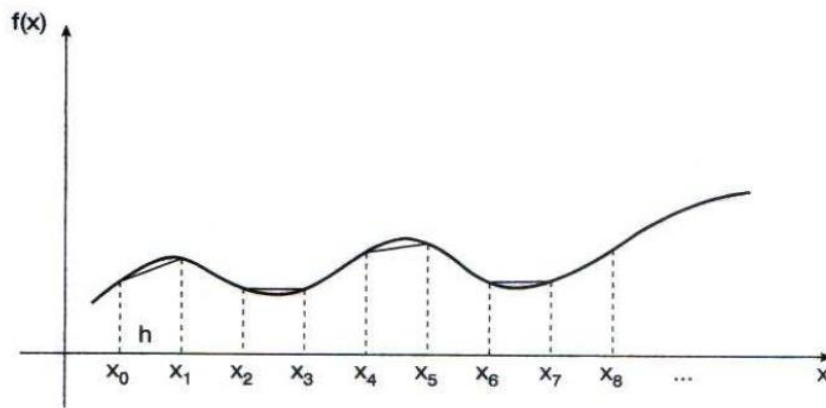


Figura 1.8 (ilustrativa)

## Código

Segue abaixo o código implementado para a resolução do problema.

```
%vamos usar a regra do trapézio
function T3=Trapezio
X=[0 0.5 1 2 2.5 3 3.5];
y=[2 1.6 0.8 0.04 0.004 0.0002 0.000009];
DifereDivi=zeros(7,7);
DifereDivi(:,1)=y;
n=6;
for i=2:n+1
    for j=i:n+1
        DifereDivi(j,i)=[DifereDivi(j,i-1)-DifereDivi(j-1,i-1)]/[X(j)-X(j-i+1)];
    end
end

%assim vamos calcular todas as diferenças divididas e montar nosso
%polinômio para a integração
syms x
polnewton=DifereDivi(1,1);
P=1;
%vamos fazer essas operações para controlar o nosso novo polinômio
```

```

for i1=1:6
    P=P*(x-X(i1));
    polnewton=polnewton+P*DifereDivi(i1+1,i1+1);

end

    polnewton=expand(polnewton);
    polnewton=simplify(polnewton);
    F=@(x) -0.0102*x^6+0.0779*x^5 - 0.2923*x^4+0.6800*x^3-0.800*x^2-
0.800*x + 2;
    a=0;
    b=3.5;
    integral=0.5*(F(a)+(b));
    n=10000;
    h=(b-a)/n;
    c=a+h;
    for i=1:n-1
        integral=integral+F(c);
        c = c + h;
    end
    integral=h*integral
    %agora vamos calcular o erro
    %temos que a derivada de segunda ordem da nossa função é dada
como
    Derivate=@(x) -(3060*x^4-15580*x^3+35076*x^2-
40800*x+16000)/10000;
    % cálculo do erro pode ser dado como uma aproximação, já que não
temos
    % a função exata
    xe=max(abs(X));
    m2=abs(Derivate(xe));
    erro_de_aproximacao=(n*h^3*m2)/12
    %podemos também comparar com um erro das diferenças divididas
    erro_difereDivi=(n*h^3*abs(DifereDivi(7,7)))/6
    %estariamos nesse caso fazendo uma aproximação da função
exatamente
    %falando porque não temos a cara dela então apenas pegamos a
diferença
    %dividida de ordem n+1 que no nosso caso foi 7

end

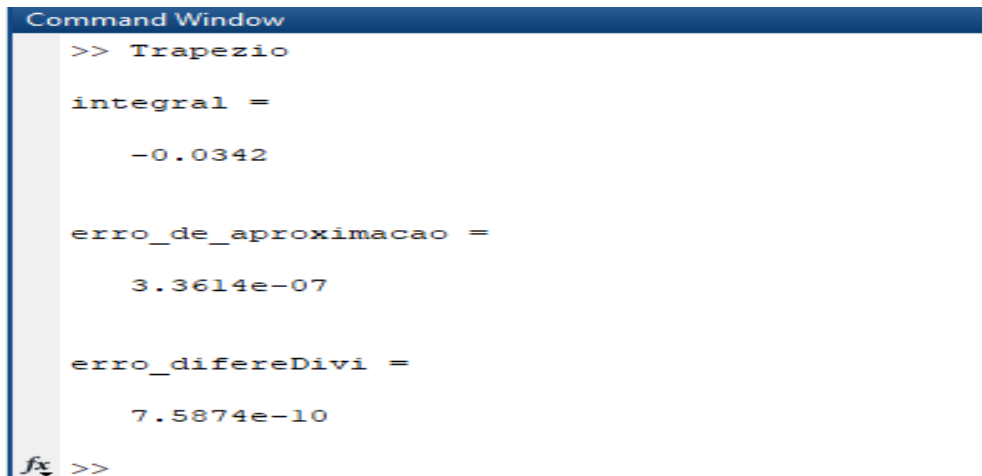
```

## Análise do Código

Quando estávamos implementando esse código tínhamos a opção de fazer o calculo ponto a ponto ou criarmos um polinómio interpolador, que caracterizasse a função, que foi a opção desejada, isso acaba alongando mais o código, mas também não é nada que não tenha sido utilizado em problemas passados.

## Análise de Dados

Podemos observar que o erro de aproximação foi diferente do obtido com as diferenças divididas, quando realizamos a interpolação de todos os pontos, para criar o polinômio não estamos recriando a função verdadeira e sim fazendo o único polinômio que interpola todos os pontos, então no cálculo do erro, como um foi realizado através das diferenças divididas e o outro pela derivada da função observamos uma diferença considerável.



```
Command Window
>> Trapezio

integral =

    -0.0342

erro_de_aproximacao =

    3.3614e-07

erro_difereDivi =

    7.5874e-10

fx >>
```

Figura 1.9

## Conclusão

Através de todos os procedimentos apresentados e executados, foram realizadas metodologias que obedeceram a todos os processos teóricos, buscando sempre que na medida do possível a máxima eficiência nos resultados e nas operações executadas, evitando sempre o custo computacional desnecessário, poderíamos sim ter melhorado, refinado ainda mais nossos códigos, mas com o que já foi apresentado conseguimos obter ótimos resultados que se aproximam e muito dos resultados teóricos.



## BIBLIOGRAFIA

Notas de aula do Professor Giusepp Romanazzi.

Cálculo Numérico aspectos teóricos e computacionais, Márcia A.gomes Ruggiero,  
Vera lúcia da Rocha Lopes.