MÉTODOS NUMÉRICOS PARA INTEGRAÇÃO

Evandro Pedro Alves de Mendonçaa, Marcelino José de Lima Andradea.

a Núcleo de Tecnologia (NTI), Universidade Federal de Pernambuco (UFPE), Campus Acadêmico do Agreste (CAA), Rodovia BR-104, km 59, S/N, Nova Caruaru, CEP. 55.014-900, Caruaru-PE, Brasil, <http://www.ufpe.br/caa>

**Palavras Chave:** Integração numérica, fórmulas, erro, truncamento, aproximação, funções.

**Resumo**: Este trabalho visa analisar os diversos métodos de integração numérica e discernir qual o melhor método a ser empregado em diversas situações. Serão levados em conta vários parâmetros para a escolha do melhor método, fatores como tempo computacional e erro associado a cada método. No trabalho também é mostrada a implementação de cada método de integração pedido.

1. INTRODUção

A integração numérica tem importância inegável nas ciências exatas. Quando possível, a integração é feita analiticamente. Porém, existem algumas situações em que calcular uma integral analiticamente torna-se inviável. Pode ser que o integrando seja muito complicado, ou que não haja integrando realmente expresso, desta forma é quase que impossível resolver a integral analiticamente. Também existem situações em que é possível integra-se analiticamente, pois o integrando está devidamente expresso, porém o tempo para a análise do integrando é muito alto, e é mais viável optar-se pela integração numérica. Ainda pode-se ter o caso em que o integrando não está definido em todos os pontos do intervalo, tornando o cálculo analítico inviável novamente.

Alguns métodos numéricos levam em consideração o fato de que, a integral representa a área abaixo da curva na maioria dos casos. Por isso, os métodos fazem aproximações do integrando para funções mais simples, ou polígonos cuja área é facilmente calculável. Desta forma é possível ter-se uma boa aproximação da integral.

Dos métodos usados, dois bastante conhecidos são o de Newton-Cotes e da quadratura de Gauss. No método de Newton, o intervalo de integração é igualmente espaçado, e a função integrada é aproximada por polinômios conhecidos e facilmente integráveis. O método de Gauss é mais aprimorado e não necessariamente utiliza pontos igualmente espaçados, mas sim, pontos sabiamente escolhidos previamente, para que o erro seja mínimo.

O cálculo do erro deve ser feito sempre e é primordial, para que se tenha a noção da acurácia dos valores obtidos. A partir do erro, pode-se escolher e discernir qual o método deve ser utilizado em cada caso, e ajuda na quantidade de intervalos que devem ser usados.

1. Exercícios Propostos

Segue, abaixo, a solução dos exercícios propostos sobre o tema.

* 1. 1ª questão.

A resolução da questão está na figura abaixo.

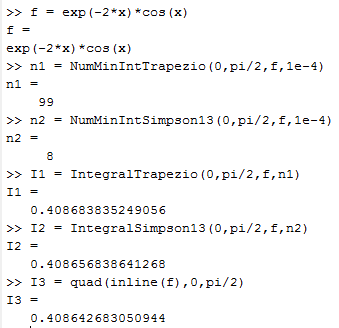


Figura 1: resolução da 1ª questão.

Primeiramente, declaramos f, que é uma função da variável simbólica x. Em seguida, chamamos a função *NumMinIntTrapezio*, que calcula o número mínimo de subintervalos para o método do Trapézio considerando a tolerância 10-4. O resultado indica que precisamos de no mínimo 99 subintervalos nesse método para garantir a tolerância exigida. Depois chamamos a função *NumMinIntSimpson13*, que faz a mesma coisa, só que considerando o método de 1/3 de Simpson. O resultado mostra que precisamos de apenas 8 subintervalos para garantir a tolerância. Para garantir que teremos quatro casas decimais iguais com os dois métodos, simplesmente calculamos a integral com ambos os métodos e também com uma terceira função que é nativa do MATLAB®. Como pode-se ver, nos três casos, as primeiras 4 casas decimais são idênticas. Os algoritmos desenvolvidos para essa questão estão nos Anexos X, Y, Z e W.

* 1. 2ª questão
     1. – 2.a)

Como não há uma maneira simples de encontrar o número de pontos que garante uma precisão determinada para a Quadratura de Gauss, implementamos um algoritmo que calcula a estimativa de erro. Então calculamos empiricamente a integral pedida e vemos se o erro atingiu a tolerância exigida.

Com o código do Anexo X, calculamos a integral da função dada.

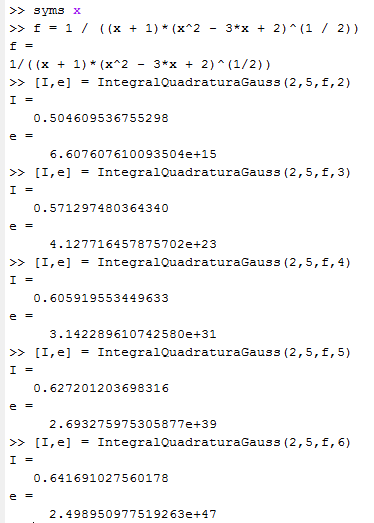


Figura 2: declaração da função.

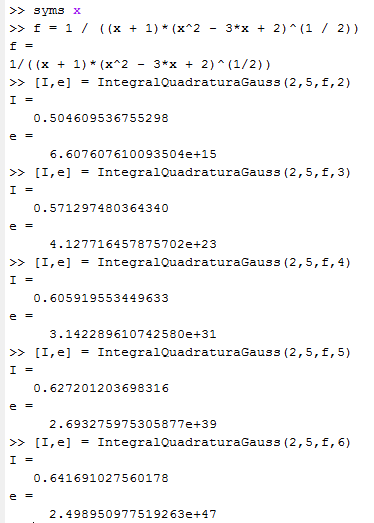


Figura 3: cálculo da integral utilizando a Quadratura de Gauss com 2, 3, 4, 5 e 6 pontos.

Como foi observado, o erro calculado foi muito grande em todos os casos e o valor da integral converge muito lentamente para o valor real com o aumento de pontos.

* + 1. – 2.b)

Utilizando os métodos de Newton-Cotes apresentados na disciplina, apenas dois tornam o cálculo da integral viável. Isso acontece porque a função é assintótica no ponto 2, ou seja, a imagem da função nesse ponto tende a infinito. Com os métodos fechados que calculam a imagem do ponto inicial, o valor da integral também diverge. Portanto, precisamos de um método que utilize as imagens em outros pontos que não sejam o ponto inicial. Temos o método do retângulo com o ponto final do intervalo e o método do ponto médio. Dentre esses, o que tem melhor exatidão é o método do ponto médio e este será o método utilizado nesta questão.

Para saber quantos subintervalos precisaremos para garantir a exatidão exigida, utilizamos o algoritmo do Anexo X. Utilizando o algoritmo do Anexo X, que implementa o método do ponto médio, calculamos a integral.

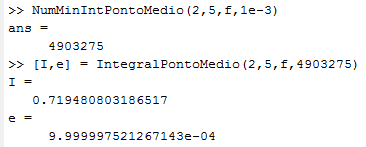


Figura 4: com a primeira função, descobre-se o número de intervalos e, com a segunda função, calcula-se a integral e a estimativa de erro.

Vemos que com o método do ponto médio, garantimos um erro menor que 10-3, porém o número de subintervalos é quase 5 milhões, isto é, o método é muito ineficiente.

* + 1. – 2.c)

Utilizando funções do MATLAB® para integrar a função, temos:

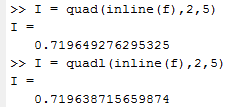


Figura 5: resultados utilizando as funções quad e quadl. Ambas realizam o cálculo com precisão de 10-6.

Temos outras funções para calcular as integrais, que são as funções *trapz* e *dblquad*, porém a primeira é usada para pontos discretos e a segunda para integrais duplas. Portanto, não se aplicam ao escopo dessa questão.

Comparando os resultados com os dos itens anteriores, vemos que a função *IntegralPontoMedio* conseguiu um valor exato nas três primeiras casas decimais enquanto que a função *IntegralQuadraturaGauss* não conseguiu. Já era esperado, visto que o cálculo da estimativa de erro para esta função retornou um valor muito alto.

* 1. 3ª questão

Primeiramente, calculamos os as integrais analiticamente.

O gráfico das duas funções é mostrado na figura a seguir:

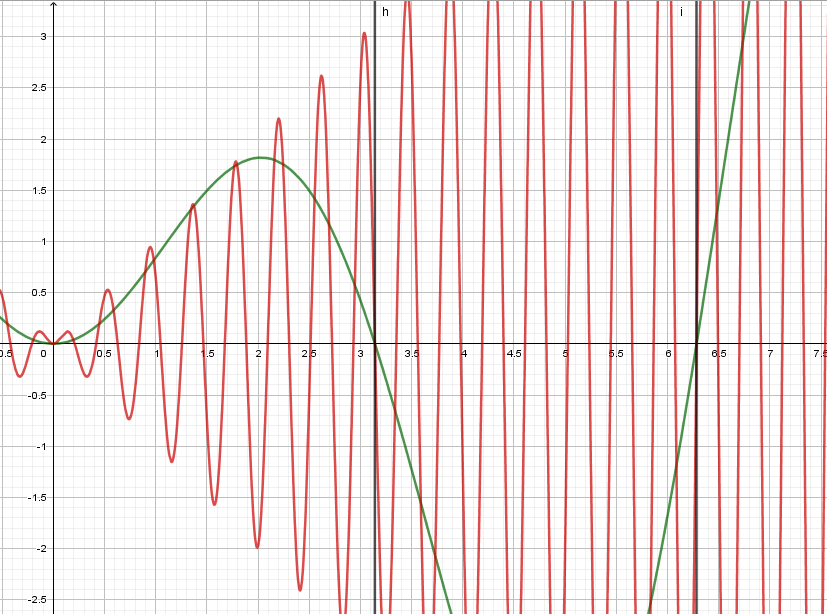


Figura 6: parte do gráfico das funções (i – em verde) e (ii – em vermelho). Também são mostradas na figura as retas x = e x = 2, que são os limites superiores de integração.

* + 1. – 3.a) a 3.d)

Para os itens 3.a) até 3.d), foi feito um algoritmo (Anexo X) que resolve todas as integrais usando os algoritmos desenvolvidos para cada método (Anexos X, Y, Z, W), calcula os erros (Anexo X) e dispõe tudo em uma tabela, que é mostrada na figura abaixo.

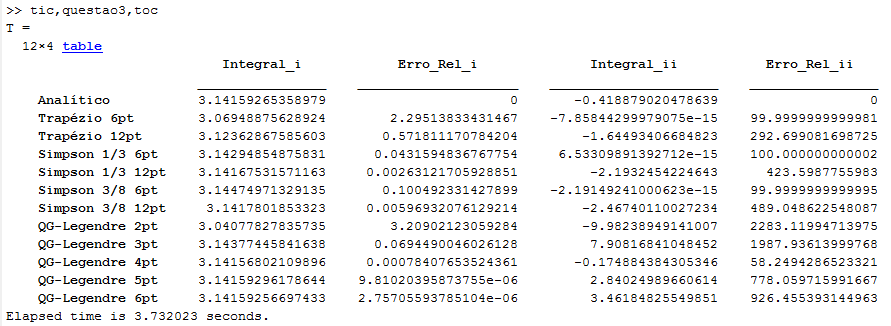


Figura 7

* + 1. – 3.e)

Os resultados da integral (i) foram bastante satisfatórios. Com um simples aumento no número de pontos usados, o erro diminuiu consideravelmente. Já para a integral (ii), não observa-se o mesmo comportamento. Não há como determinar se, com um aumento pequeno no número de pontos, o erro tende a diminuir. Para esse caso, talvez o erro só viesse a diminuir satisfatoriamente se o número de pontos fosse muito grande, o que fornece mais exatidão ao cálculo. O motivo do comportamento dessas integrais é que, como a primeira integral tem uma taxa de variação pequena no domínio de integração, o erro também tende a ser pequeno; isso não acontece na segunda integral, que tem uma taxa de variação muito grande com relação à primeira, logo o erro também tende a variar bastante. Para calcular com mais exatidão essas integrais, pode-se usar mais pontos em cada método ou usar um método diferente. Por exemplo, o método adaptativo de Simpson discretiza com mais pontos as regiões do domínio que têm grande taxa de variação, o que confere maior precisão nas aproximações daquela região; nas demais regiões, onde a taxa de variação é pequena, a discretização também é menor, pois o erro tende a ser menor. Caso seja necessário, por exemplo, calcular a integral de uma função que modela uma onda de alta frequência, é mais vantajoso utilizar um método adaptativo.

* 1. 4ª questão

Utilizando os algoritmos dos Anexos X e Y, encontramos os seguintes resultados.

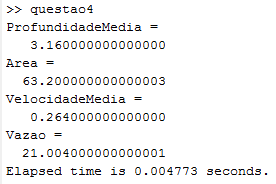


Figura 8: resolução da questão 4.

A seguir, discutiremos o procedimento feito para calcular cada uma das grandezas.

* + 1. – 4.a)

Para calcular a profundidade média, a partir dos dados de profundidade, usamos o Teorema do Valor médio, que define que existe um valor Hm (profundidade média) que representa um retângulo de base ∆y cuja área é igual à área sob a função H(y) (nesse caso, os dados de profundidade). Podemos então calcular a profundidade média como segue abaixo.

O valor da integral é obtido através do método do trapézio. A função *IntegralTrapezio* foi modificada para *IntTrapz*, em que ao invés de receber como parâmetros os extremos do domínio de integração, a função e o número de subintervalos, recebe apenas dois vetores de mesmo tamanho com dados discretos. Multiplicando o inverso de ∆y pelo resultado da integral acima, obtivemos o valor da profundidade média.

* + 1. – 4.b)

A área da seção foi simplesmente calculada a partir da integral:

Usando a função *IntTrapz*, encontramos o valor da integral e, consequentemente, da área.

* + 1. – 4.c)

O valor da velocidade média foi encontrado da mesma forma que a profundidade média, através da seguinte expressão:

Novamente, fez-se uso da função *IntTrapz* para calcular o valor da integral e, assim, encontrar a velocidade média.

* + 1. – 4.d)

Para calcular a vazão do escoamento, a partir da seguinte expressão:

Encontramos o resultado calculando a integral pela função *IntTrapz*, porém passamos como parâmetro a multiplicação elemento a elemento dos vetores H e U.

* 1. 5ª questão

1. conclusão

Conclui-se que os métodos para integração são bastante úteis no ramo da engenharia. Por diversas vezes não será possível realizar uma integração de maneira analítica. Por outras, será possível, porém, não será trivial. Por meio da utilização de métodos numéricos, é possível economizar tempo, e ter resultados com boa acurácia. Porém, é preciso escolher o método sabiamente, pois desta forma, tem-se o controle do erro obtido, e também, pode-se saber com a maior acurácia possível a dimensão do erro.

REFERÊNCiaS

Chapra, S. C., e Canale, R. P. *Métodos Numéricos para Engenharia*. 5ª edição. Porto Alegre: AMGH, 2011.

Gilat, A., e Subramaniam, V. *Métodos Numéricos para Engenheiros e Cientistas: uma introdução com aplicações usando o MATLAB*. Porto Alegre: Bookman, 2008.

anexo 1

function [ I,erro ] = IntegralPontoMedio(a,b,f,n)  
%Calcula integral numericamente pelo método do Ponto Médio  
%Parâmetros: [ I,erro ] = IntegralTrapezio(a,b,f,n)  
%a: ponto inicial do domínio de integração  
%b: ponto final do domínio de integração  
%f: função do integrando  
%n: número de intervalos  
%I: valor da integral (saída)  
%erro: estimativa de erro de truncamento (saída)  
  
 %%%VALIDAÇÃO%%%  
 if isa(f,'sym')==false  
 f = sym(f); %transforma string em uma função simbólica  
 end  
 if isnumeric(a)==false||isnumeric(b)==false||isnumeric(n)==false  
 disp('Erro. Parâmetros a, b e n devem ser numéricos.');  
 I = 'erro'; erro = I;  
 return;  
 end  
 if a>b % a deve ser menor que b  
 disp('Erro. Parâmetro a deve ser menor que b.');  
 I = 'erro'; erro = I;  
 return;  
 elseif a==b % a deve ser diferente de b  
 disp('Erro. Parâmetros a e b devem ser diferentes.');  
 I = 'erro'; erro = I;  
 return;  
 end  
 if mod(fix(n),n)~=0||n<=0  
 disp('Erro. Número de intervalos deve ser um inteiro positivo.');  
 I = 'erro'; erro = I;  
 return;  
 end  
  
 %%%PROCESSAMENTO%%%  
 h = (b-a)/n; %define tamanho do intervalo  
 x = a:h:b; %discretização do domínio  
 g = inline(f); %cria função inline a partir de f  
 soma = 0; %declaração da variável do somatório  
 for i = 1:n  
 %calcula imagem de cada ponto médio e acrescenta à soma  
 soma = soma + g((x(i)+x(i+1))/2);  
 end  
  
 %%%SAÍDA%%%  
 I = h\*soma; %cálculo da integral  
  
 %cálculo do erro  
 erro = ((b-a)/24)\*maximo(diff(f,2),a,b)\*h^2;  
end

anexo 2

function [ I ] = IntegralTrapezio(a,b,f,n)  
%Calcula integral numericamente pelo método do trapézio.  
%Parâmetros: [ I ] = IntegralTrapezio(a,b,f,n)  
%a: ponto inicial do domínio de integração  
%b: ponto final do domínio de integração  
%f: função do integrando  
%n: número de intervalos  
%I: valor da integral (saída)  
  
 %%%VALIDAÇÃO%%%  
 if isa(f,'inline')==false  
 f = inline(f); %transforma string em uma função inline  
 end  
 if isnumeric(a)==false||isnumeric(b)==false||isnumeric(n)==false  
 disp('Erro. Parâmetros a, b e n devem ser numéricos.');  
 I = 'erro';  
 return;  
 end  
 if a>b % a deve ser menor que b  
 disp('Erro. Parâmetro a deve ser menor que b.');  
 I = 'erro';  
 return;  
 elseif a==b % a deve ser diferente de b  
 disp('Erro. Parâmetros a e b devem ser diferentes.');  
 I = 'erro';  
 return;  
 end  
 if mod(fix(n),n)~=0||n<=0  
 disp('Erro. Número de intervalos deve ser um inteiro positivo.');  
 I = 'erro';  
 return;  
 end  
  
 %%%PROCESSAMENTO%%%  
 h = (b-a)/n; %define tamanho do intervalo  
 x = a:h:b; %discretização do domínio  
 y = f(x); %imagens dos pontos do domínio discretizado  
 soma = 0; %declaração da variável do somatório  
 for i = 1:n  
 soma = soma + (y(i)+y(i+1));  
 end  
  
 %%%SAÍDA%%%  
 I = (h/2)\*soma;  
end

anexo 3

function [ I ] = IntTrapz(X,Y)  
%Calcula integral numericamente pelo método do Trapézio  
%Parâmetros: [ I ] = IntTrapz(X,Y)  
%X: vetor do domínio  
%Y: vetor de imagens  
%I: valor da integral (saída)  
  
 %%%VALIDAÇÃO%%%  
 if isnumeric(X)==false||isnumeric(Y)==false  
 disp('Erro. Parâmetros devem ser numéricos.');  
 I = 'erro'; return;  
 end  
 if(((size(X,1)>1)&&(size(X,2)>1))||((size(Y,1)>1)&&(size(Y,2)>1)))  
 disp('Erro! X ou Y não é um vetor.');  
 I = 'erro'; return;  
 elseif(length(X)~=length(Y))%x e y devem ter mesmo número de elementos  
 disp('Erro! Dimensões de X e Y não são equivalentes.');  
 I = 'erro'; return;  
 end  
 if length(X)<=1  
 disp('Erro. Vetores devem ter ao menos 2 elementos.');  
 I = 'erro'; return;  
 end  
  
 %%%PROCESSAMENTO E SAÍDA%%%  
 I = 0;  
 for i = 1:length(X)-1  
 %calcula a interpolação do ponto médio e multiplica pelo tamanho do  
 %intervalo  
 I = I + ((Y(i+1)+Y(i))\*(X(i+1)-X(i)))/2;  
 end  
end

anexo 4

function [ I ] = IntegralSimpson13(a,b,f,n)  
%Calcula integral numericamente pelo método de 1/3 de Simpson.  
%Parâmetros: [ I ] = IntegralTrapezio(a,b,f,n)  
%a: ponto inicial do domínio de integração  
%b: ponto final do domínio de integração  
%f: função do integrando  
%n: número de intervalos  
%I: valor da integral (saída)  
  
 %%%VALIDAÇÃO%%%  
 if isa(f,'inline')==false  
 f = inline(f); %transforma string em uma função inline  
 end  
 if isnumeric(a)==false||isnumeric(b)==false||isnumeric(n)==false  
 disp('Erro. Parâmetros a, b e tol devem ser numéricos.');  
 I = 'erro';  
 return;  
 end  
 if a>b % a deve ser menor que b  
 disp('Erro. Parâmetro a deve ser menor que b.');  
 I = 'erro';  
 return;  
 elseif a==b % a deve ser diferente de b  
 disp('Erro. Parâmetros a e b devem ser diferentes.');  
 I = 'erro';  
 return;  
 end  
 if mod(fix(n),n)~=0||n<=0||mod(n,2)~=0  
 disp('Erro. Número de intervalos deve ser inteiro, positivo e par.');  
 I = 'erro';  
 return;  
 end  
  
 %%%PROCESSAMENTO%%%  
 h = (b-a)/n; %define tamanho do intervalo  
 x = a:h:b; %discretização do domínio  
 y = f(x); %imagens dos pontos do domínio discretizado  
 soma = 0; %declaração da variável do somatório  
 for i = 2:n %pontos internos  
 if mod(i,2)==0  
 soma = soma + 4\*y(i);  
 else  
 soma = soma + 2\*y(i);  
 end  
 end  
 soma = (h/3)\*soma; %fator externo ao somatório  
  
 %%%SAÍDA%%%  
 I = soma + (h/3)\*(y(1)+y(n+1)); %adiciona pontos das extremidades  
end

anexo 5

function [ I ] = IntegralSimpson38(a,b,f,n)  
%Calcula integral numericamente pelo método de 3/8 de Simpson.  
%Parâmetros: [ I ] = IntegralTrapezio(a,b,f,n)  
%a: ponto inicial do domínio de integração  
%b: ponto final do domínio de integração  
%f: função do integrando  
%n: número de intervalos  
%I: valor da integral (saída)  
  
 %%%VALIDAÇÃO%%%  
 if isa(f,'inline')==false  
 f = inline(f); %transforma string em uma função inline  
 end  
 if isnumeric(a)==false||isnumeric(b)==false||isnumeric(n)==false  
 disp('Erro. Parâmetros a, b e tol devem ser numéricos.');  
 I = 'erro'; return;  
 end  
 if a>b % a deve ser menor que b  
 disp('Erro. Parâmetro a deve ser menor que b.');  
 I = 'erro'; return;  
 elseif a==b % a deve ser diferente de b  
 disp('Erro. Parâmetros a e b devem ser diferentes.');  
 I = 'erro'; return;  
 end  
 if mod(fix(n),n)~=0||n<=0||mod(n,3)~=0  
 disp('Erro. Número de intervalos deve ser inteiro, positivo e divisível por 3.');  
 I = 'erro'; return;  
 end  
 %%%PROCESSAMENTO%%%  
 h = (b-a)/n; %define tamanho do intervalo  
 x = a:h:b; %discretização do domínio  
 y = f(x); %imagens dos pontos do domínio discretizado  
 soma = 0; %declaração da variável do somatório  
 j = 1; %variável auxiliar  
 for i = 2:n %pontos internos  
 if mod(j,3)==0 %entra nessa condição a cada 3 iterações  
 soma = soma + 2\*y(i);  
 j = 0; %restaura valor de j  
 else  
 soma = soma + 3\*y(i);  
 end  
 j = j + 1; %incrementa j  
 end  
 soma = (3\*h/8)\*soma; %fator externo ao somatório  
 %%%SAÍDA%%%  
 I = soma + (3\*h/8)\*(y(1)+y(n+1)); %adiciona pontos das extremidades  
end

anexo 6

function [ I,erro ] = IntegralQuadraturaGauss(a,b,f,n)  
%Calcula integral numericamente pelo método da Quadratura de Gauss  
%utilizando de 2 até 6 pontos no domínio.  
%Parâmetros: [ I,erro ] = IntegralTrapezio(a,b,f,n)  
%a: ponto inicial do domínio de integração  
%b: ponto final do domínio de integração  
%f: função do integrando  
%n: número de pontos  
%I: valor da integral (saída)  
%erro: estimativa de erro de truncamento (saída)  
  
 %%%VALIDAÇÃO%%%  
 if isa(f,'sym')==false  
 f = sym(f); %transforma string em uma função simbólica  
 end  
 if isnumeric(a)==false||isnumeric(b)==false||isnumeric(n)==false  
 disp('Erro. Parâmetros a, b e tol devem ser numéricos.');  
 I = 'erro'; erro = I;  
 return;  
 end  
 if a>b % a deve ser menor que b  
 disp('Erro. Parâmetro a deve ser menor que b.');  
 I = 'erro'; erro = I;  
 return;  
 elseif a==b % a deve ser diferente de b  
 disp('Erro. Parâmetros a e b devem ser diferentes.');  
 I = 'erro'; erro = I;  
 return;  
 end  
 if mod(fix(n),n)~=0||n<2||n>6  
 disp('Erro. Número de pontos deve ser um inteiro entre 2 e 6.');  
 I = 'erro'; erro = I;  
 return;  
 end  
  
 %%%PROCESSAMENTO%%%  
 %definindo pontos e pesos  
 if n==2  
 %coeficientes (pesos)  
 c = [1 1];  
 %pontos do domínio  
 x = [-0.577350269 0.577350269];  
 df = diff(f,4); %calcula derivada de 4ª ordem de f  
 elseif n==3  
 %coeficientes (pesos)  
 c = [0.5555556 0.8888889 0.5555556];  
 %pontos do domínio  
 x = [-0.774596669 0 0.774596669];  
 df = diff(f,6); %calcula derivada de 6ª ordem de f  
 elseif n==4  
 %coeficientes (pesos)  
 c = [0.3478548 0.6521452 0.6521452 0.3478548];  
 %pontos do domínio  
 x = [-0.861136312 -0.339981044 0.339981044 0.861136312];  
 df = diff(f,8); %calcula derivada de 8ª ordem de f  
 elseif n==5  
 %coeficientes (pesos)  
 c = [0.2369269 0.4786287 0.5688889 0.4786287 0.2369269];  
 %pontos do domínio  
 x = [-0.906179846 -0.538469310 0 0.538469310 0.906179846];  
 df = diff(f,10); %calcula derivada de 10ª ordem de f  
 elseif n==6  
 %coeficientes (pesos)  
 c = [0.1713245 0.3607616 0.4679139 0.4679139 0.3607616 0.1713245];  
 %pontos do domínio  
 x = [-0.932469514 -0.661209386 -0.238619186 0.238619186];  
 x = [x 0.661209386 0.932469514];  
 df = diff(f,12); %calcula derivada de 12ª ordem de f  
 end  
  
 %mudança de variável  
 g = inline('(t\*(b-a)+a+b)/2'); %cria função de mudança de variável  
 dt = (b-a)/2; %termo jacobiano da mudança de variável  
 f = inline(f); %transforma f de simbólica para inline  
  
 %%%SAÍDA%%%  
 %cálculo da integral  
 I = 0; %declaração de variável de saída inicialmente nula  
 for i = 1:n  
 I = I + c(i)\*f(g(a,b,x(i)))\*dt;  
 end  
 %cálculo do erro  
 erro = ((2^(2\*n+3))\*(factorial(n+1))^4)/((2\*n+3)\*(factorial(2\*n+2))^3);  
 erro = erro\*(maximo(df,a,b));  
end

anexo 7

function [ n ] = NumMinIntTrapezio(a,b,f,tol)  
%Esta função calcula o número mínimo de intervalos para garantir uma  
%exatidão definida no processo de integração numérica pelo método do  
%trapézio.  
%Parâmetros: [ n ] = NumMinIntTrapezio(a,b,f,tol)  
%a: extremo esquerdo do domínio de integração  
%b: extremo direito do domínio de integração  
%f: função do integrando  
%tol: tolerância de exatidão.  
  
 %%%VALIDAÇÃO%%%  
 if isa(f,'sym')==false  
 f = sym(f); %transforma string em uma função simbólica  
 end  
 if isnumeric(a)==false||isnumeric(b)==false||isnumeric(tol)==false  
 disp('Erro. Parâmetros a, b e tol devem ser numéricos.');  
 n = 'erro';  
 return;  
 end  
 if a>b % a deve ser menor que b  
 disp('Erro. Parâmetro a deve ser menor que b.');  
 n = 'erro';  
 return;  
 elseif a==b % a deve ser diferente de b  
 disp('Erro. Parâmetros a e b devem ser diferentes.');  
 n = 'erro';  
 return;  
 end  
  
 %%%PROCESSAMENTO%%%  
 d2f = diff(f,2); %calcula segunda derivada de f  
 ymax = maximo(d2f,a,b); %calcula ponto máximo de d2f  
  
 %calcula tamanho do passo de acordo com fórmula de estimativa de erro  
 %do método do trapézio  
 h = sqrt((12\*abs(tol))/(abs(b-a)\*abs(ymax)));  
  
 %%%SAÍDA%%%  
 n = ceil((b-a)/h); %calcula o número de passos no intervalo e arredonda  
 %para cima  
end

anexo 8

function [ n ] = NumMinIntSimpson13(a,b,f,tol)  
%Esta função calcula o número mínimo de intervalos para garantir uma  
%exatidão definida no processo de integração numérica pelo método de  
%Simpson de 1/3.  
%Parâmetros: [ n ] = NumMinIntSimpson13(a,b,f,tol)  
%a: extremo esquerdo do domínio de integração  
%b: extremo direito do domínio de integração  
%f: função do integrando  
%tol: tolerância de exatidão.  
  
 %%%VALIDAÇÃO%%%  
 if isa(f,'sym')==false  
 f = sym(f); %transforma string em uma função simbólica  
 end  
 if isnumeric(a)==false||isnumeric(b)==false||isnumeric(tol)==false  
 disp('Erro. Parâmetros a, b e tol devem ser numéricos.');  
 n = 'erro';  
 return;  
 end  
 if a>b % a deve ser menor que b  
 disp('Erro. Parâmetro a deve ser menor que b.');  
 n = 'erro';  
 return;  
 elseif a==b % a deve ser diferente de b  
 disp('Erro. Parâmetros a e b devem ser diferentes.');  
 n = 'erro';  
 return;  
 end  
  
 %%%PROCESSAMENTO%%%  
 d4f = diff(f,4); %calcula segunda derivada de f  
 ymax = maximo(d4f,a,b); %calcula ponto máximo de d4f  
  
 %calcula tamanho do passo de acordo com fórmula de estimativa de erro  
 %do método do trapézio  
 h = ((180\*abs(tol))/(abs(b-a)\*abs(ymax)))^(1/4);  
  
 %%%SAÍDA%%%  
 n = ceil((b-a)/h); %calcula o número de passos no intervalo  
 if mod(n,2)~=0 %se n for impar, acrescenta mais uma unidade  
 n = n+1;  
 end  
end

anexo 9

function [ n ] = NumMinIntPontoMedio(a,b,f,tol)  
%Esta função calcula o número mínimo de intervalos para garantir uma  
%exatidão definida no processo de integração numérica pelo método do  
%ponto médio.  
%Parâmetros: [ n ] = NumMinIntTrapezio(a,b,f,tol)  
%a: extremo inferior do domínio de integração  
%b: extremo superior do domínio de integração  
%f: função do integrando  
%tol: tolerância de exatidão.  
  
 %%%VALIDAÇÃO%%%  
 if isa(f,'sym')==false  
 f = sym(f); %transforma string em uma função simbólica  
 end  
 if isnumeric(a)==false||isnumeric(b)==false||isnumeric(tol)==false  
 disp('Erro. Parâmetros a, b e tol devem ser numéricos.');  
 n = 'erro';  
 return;  
 end  
 if a>b % a deve ser menor que b  
 disp('Erro. Parâmetro a deve ser menor que b.');  
 n = 'erro';  
 return;  
 elseif a==b % a deve ser diferente de b  
 disp('Erro. Parâmetros a e b devem ser diferentes.');  
 n = 'erro';  
 return;  
 end  
  
 %%%PROCESSAMENTO%%%  
 d2f = diff(f,2); %calcula segunda derivada de f  
 ymax = maximo(d2f,a,b); %calcula ponto máximo de d2f  
  
 %calcula tamanho do passo de acordo com fórmula de estimativa de erro  
 %do método do trapézio  
 h = sqrt((24\*abs(tol))/(abs(b-a)\*abs(ymax)));  
  
 %%%SAÍDA%%%  
 n = ceil((b-a)/h); %calcula o número de passos no intervalo e arredonda  
 %para cima  
end

anexo 10

function [ Ymax ] = maximo(f,a,b)  
%Esta função retorna o ponto y máximo de uma função num intervalo [a,b]  
%Parâmetros: [ Xmax,Ymax ] = maximo(f,a,b)  
%f: função (simbólica)  
%a: extremo inferior do intervalo  
%b: extremo superior do intervalo  
%Ymax: coordenada y do ponto máximo (saída)  
  
 %%%VALIDAÇÃO%%%  
 if ischar(f)==true  
 f = sym(f); %transforma string em uma função simbólica  
 end  
 if isnumeric(a)==false||isnumeric(b)==false  
 disp('Erro. Parâmetros a e b devem ser numéricos.'); Ymax = 'erro'; return;  
 end  
 if a>b % a deve ser menor que b  
 disp('Erro. Parâmetro a deve ser menor que b.'); Ymax = 'erro'; return;  
 elseif a==b % a deve ser diferente de b  
 disp('Erro. Parâmetros a e b devem ser diferentes.'); Ymax = 'erro'; return;  
 end  
 %%%PROCESSAMENTO%%%  
 g = -f;%cria função que é a negativa de f  
 f = inline(f); %transforma f em inline  
 g = inline(g); %transforma g em inline  
 Xmax = fminbnd(g,a,b); %calcula o mínimo de g, que é o máximo de f  
 %%%SAÍDA%%%  
 Ymax = f(Xmax); %avalia f no ponto x máximo  
end

anexo 11

function [ erro ] = ErroRelativoPercentual(Vexato,Vaproximado)  
%Calcula o erro relativo percentual de um valor aproximado  
%Parâmetros: [ erro ] = ErroRelativoPercentual(Vexato,Vaproximado)  
%Vexato: valor exato  
%Vaproximado: valor aproximado  
%erro: erro relativo percentual (saída)  
  
 %%%VALIDAÇÃO%%%  
 if isnumeric(Vexato)==false||isnumeric(Vaproximado)==false  
 disp('Erro. Todos os parâmetros devem ser numéricos.'); erro = 'erro'; return;  
 end  
 %%%PROCESSAMENTO E SAÍDA%%%  
 erro = (abs(Vexato - Vaproximado)./abs(Vexato))\*100;  
end

anexo 12

%valores calculados analiticamente  
for i = 1:12  
 Vexato1(i,1) = pi;  
 Vexato2(i,1) = -(2\*pi)/15;  
end  
%declaração das funções  
syms x; f1 = x\*sin(x); f2 = x\*sin(15\*x);  
%vetor com nomes dos métodos  
metodos = string({'Analítico';'Trapézio 6pt';'Trapézio 12pt';...  
 'Simpson 1/3 6pt';'Simpson 1/3 12pt';'Simpson 3/8 6pt';...  
 'Simpson 3/8 12pt';'QG-Legendre 2pt';'QG-Legendre 3pt';...  
 'QG-Legendre 4pt';'QG-Legendre 5pt';'QG-Legendre 6pt'});  
metodos = cellstr(metodos);  
%cálculo das integrais  
%método do trapézio  
I1(1,1) = IntegralTrapezio(0,pi,f1,6);  
I1(2,1) = IntegralTrapezio(0,pi,f1,12);  
I2(1,1) = IntegralTrapezio(0,2\*pi,f2,6);  
I2(2,1) = IntegralTrapezio(0,2\*pi,f2,12);  
%método de 1/3 de Simpson  
I1(3,1) = IntegralSimpson13(0,pi,f1,6);  
I1(4,1) = IntegralSimpson13(0,pi,f1,12);  
I2(3,1) = IntegralSimpson13(0,2\*pi,f2,6);  
I2(4,1) = IntegralSimpson13(0,2\*pi,f2,12);  
%método de 3/8 de Simpson  
I1(5,1) = IntegralSimpson38(0,pi,f1,6);  
I1(6,1) = IntegralSimpson38(0,pi,f1,12);  
I2(5,1) = IntegralSimpson38(0,2\*pi,f2,6);  
I2(6,1) = IntegralSimpson38(0,2\*pi,f2,12);  
%método da Quadratura de Gauss-Legendre  
I1(7,1) = IntegralQuadraturaGauss(0,pi,f1,2);  
I1(8,1) = IntegralQuadraturaGauss(0,pi,f1,3);  
I1(9,1) = IntegralQuadraturaGauss(0,pi,f1,4);  
I1(10,1) = IntegralQuadraturaGauss(0,pi,f1,5);  
I1(11,1) = IntegralQuadraturaGauss(0,pi,f1,6);  
I2(7,1) = IntegralQuadraturaGauss(0,2\*pi,f2,2);  
I2(8,1) = IntegralQuadraturaGauss(0,2\*pi,f2,3);  
I2(9,1) = IntegralQuadraturaGauss(0,2\*pi,f2,4);  
I2(10,1) = IntegralQuadraturaGauss(0,2\*pi,f2,5);  
I2(11,1) = IntegralQuadraturaGauss(0,2\*pi,f2,6);  
%inserindo valores exatos nos vetores dos resultados  
I1 = [Vexato1(1);I1]; I2 = [Vexato2(1);I2];  
%calculando erro relativo percentual  
erro1 = ErroRelativoPercentual(Vexato1,I1);  
erro2 = ErroRelativoPercentual(Vexato2,I2);  
%criando tabela de saída  
nomes = string({'Integral\_i' 'Erro\_Rel\_i' 'Integral\_ii','Erro\_Rel\_ii'});  
nomes = cellstr(nomes);  
T = table(I1,erro1,I2,erro2,'RowNames',metodos,'VariableNames',nomes)

anexo 13

%Questão 4  
tic;  
H = [0;1.8;2;4;4;6;4;3.6;3.4;2.8;0];  
U = [0;0.1;0.14;0.36;0.38;0.47;0.43;0.33;0.25;0.18;0];  
y = 0:2:20;  
delta\_y = y(length(y))-y(1);  
  
%4.a) Calcular profundidade média  
ProfundidadeMedia = (1/(delta\_y))\*IntTrapz(y,H)  
  
%4.b) Calcular área da seção transversal  
Area = IntTrapz(y,H)  
  
%4.c) Calcular velocidade média  
VelocidadeMedia = (1/(delta\_y))\*IntTrapz(y,U)  
  
%4.d) Calcular vazão do escoamento  
Vazao = IntTrapz(y,H.\*U)  
toc;