Instituto Superior Técnico

Matemática Computacional

Mestrado Integrado em Engenharia Mecânica

Grupo 33 - Enunciado PA-52

Autores:

Diogo Silva - 93240 . . . .

Eduardo Monteiro - 93244 . . . .

Gil Simas - 93257 . . . .

Professor:

Pedro Areias

29 de Novembro 2019

Índice

1 Introdução e Fundamentos Teóricos

O enunciado do projeto (**Anexo** 6.1) pedia que fosse desenvolvido um programa que procedesse ao cálculo de 4 integrais de funções diferentes explícitas no enunciado recorrendo à regra de Simpson composta, numa integração adaptativa não-iterativa e se calculasse o erro absoluto e relativo em todos os casos.

A integração numérica é utilizada através da aproximação das funções a integrar a polinómios, quando:

- Não é conhecida a expressão da função – como quando é dada por uma tabela de valores ou por valores discretos na realização de uma atividade experimental.
- A expressão analítica da função é dada mas a sua integração é muito difícil ou demorada.
- A sua primitiva é conhecida mas devido à sua complexidade é preferível recorrer a aproximações da função para ser mais eficiente.

A regra de Simpson trata-se de uma Fórmula de Newton-Cotes fechada, mas, ao invés de considerarmos a aproximação em cada sub-intervalo através de um polinómio interpolador do 1º grau (reta), há uma aproximação melhor, considerando um polinómio interpolador do 2º grau (parábola). A Regra de Simpson tem por isso grau 3 (integra sem erros funções com grau 0, 1 e 2). Para o polinómio interpolador ser de 2º grau, ao considerarmos a regra de integração simples, precisamos de 3 pontos. Que serão ponto inicial, intermédio e final do intervalo (**Anexo** 6.3).

A principal diferença entre a regra de Simpson composta (**Anexo** 6.4) e a simples é que enquanto a regra simples usa um só intervalo para a integral, a regra de Simpson composta divide o intervalo de integração em sub-intervalos mais pequenos sendo a integral total igual à soma das integrais dos sub-intervalos. Assim há uma aproximação melhor do polinómio à função e o erro, que também vai ser igual ao somatório dos erros de cada sub-intervalo, será menor.

O enunciado também dizia que queria uma integração adaptativa não-iterativa com subdivisão do intervalo e controlo do erro local e global através de estimativas por diferenças finitas, mas por sugestão do professor, foi feita uma aproximação à quarta derivada, através da equação (32).

O algoritmo utilizado para o método adaptativo não iterativo será explicado no ponto seguinte, mas sucintamente, consiste em verificar se para todos os sub-intervalos, N. Caso não esteja, calcula-se o número de sub-divisões que é necessário aplicar a esse sub-intervalo e sub-divide-se.

A dedução destas fórmulas encontra-se no **Anexo** 6.5.

$$\text{Erro Total}, E = \sum_{i=1}^{N}\left[\sum_{i=1}^{M}\left|\frac{1}{2880}\cdot f^4(a_i, a_{ii})\cdot(a_{ii}-a_i)^5\right|\right] \tag{1}$$

$$I = \sum_{i=1}^{N}(a_{ii}-a_i)\cdot\frac{1}{6}\cdot\left(f(a_i)+f(a_{ii})+4f\left(\frac{a_{ii}-a_i}{2}\right)\right) \tag{2}$$

Caso não seja necessário dividir os sub-intervalos N, então M assume o valor 1 para efeitos de cálculo

$$\text{Quarta Derivada} = 3072\cdot\left(\frac{I_h - I_{\frac{h}{2}}}{h^5}\right) \tag{3}$$

Número de subdivisões, m, do sub-intervalo:

$$m \geq h\cdot\sqrt[4]{\frac{(b-a)}{2880\cdot\varepsilon}}\cdot f^{(4)} \tag{4}$$

2 Algoritmo do Método e Aspetos da sua Implementação

Apresenta-se aqui o algoritmo utilizado. O funcionamento da totalidade do programa construído é explicado no **Anexo** 6.6. (Para uma melhor ponte entre este algoritmo e o código criado no MATLAB, o nome dos métodos utilizados encontra-se entre parênteses retos)

Algoritmo utilizado [GetErrosEIntegrais]

Inicialização

a_0 = a

Definir Número de Sub-intervalos Iguais, N

h = (b-a)/N

Fixar Erro Máximo por Intervalo, ε

Fixar Acumulador para o Erro, E = 0

Fixar Acumulador do Integral Total, I = 0

Fixar Erro Máximo por Sub-intervalo, e = ε/N

Fixar Erro Máximo por Intervalo, ε

Para i = 1 até N fazer:

- $a_i = a_0 + h(i-1)$
- $a_{i+1} = a_i + h$
- Obter Estimativa do Erro do Sub-intervalo, $E_i[ErroInt]$
- Se $|E_i|$ ¡ e
    - Calcular Integral do Sub-intervalo $I_i[IntegrInterv]$
    - I = I + I_i
    - $E = E + |E_i|$
- Se não:
    - Calcular número de subdivisões, m [getmi]
    - $h_i = h/m$
    - Fixar Erro do Sub-intervalo, $E_i = 0$
    - Para j = 1 até m fazer:
        - $a_j = a_i + h(j-1)$
        - Calcular Integral dos Sub-intervalos, $I_j[IntegrInterv]$
        - Calcular Estimativa do Erro, $E_j[ErroInt]$
        - I=I+I_j
        - $E_i = E_i + |E_j|$
    - Fim do Ciclo J

E=E+E_i

Fim do Ciclo I

Na implementação deste programa recorreu-se a várias construções e métodos do Matlab, dos quais se consideram fundamentais: if, else, ciclos for, switch e case, e plot, loglog, stairs, zeros, abs e floor.

3 Aplicação do Método e Discussão dos Resultados

Recomenda-se como leitura complementar, a este tópico, o **Anexo** 6.8 e o **Anexo** 6.7 Os gráficos representados em 6.7 foram gerados para um erro pretendido=0.0001 e $N = 5$.

O primeiro gráfico das funções será apenas complementar, não será discutido nesta secção do relatório, porém é um bom auxilio à interpretação das subdivisões realizadas.

O segundo gráfico representa o erro de cada intervalo N e, caso haja necessidade de dividir o intervalo em m sub-divisões, representa também o erro dessas sub-divisões.

O terceiro gráfico representa em escala logarítmica, para ambos os eixos, o erro absoluto obtido em função do erro pretendido.

Para os primeiros dois integrais (figuras 5 e 6), verificamos que o programa necessita de dividir os intervalos N tanto mais quanto maiores forem os seus valores de x, o que se deve a que mais perto da origem o gráfico tem a curvatura semelhante à de uma parábola, mas à medida que se afasta, devido ao crescimento exponencial, a curva da função irá ficar cada vez mais diferenciada da de uma parábola.

Verificamos no terceiro gráfico que no caso da 1ª integral para n=5 o erro máximo não chega a 1 e na 2ª não chega a 0.001 e que há alguma proporcionalidade entre o erro pretendido e o erro atual, sendo que o erro atual diminui sempre quanto menor for o erro pretendido visto que a aproximação dos polinómios interpoladores será cada vez melhor.

Para as integrais das figuras 7 e 8, verificamos que o programa não necessita de dividir os 5 intervalos para a função da figura 7, o que quer dizer que o erro já está a ser cumprido.

No caso da integral da figura 8 é necessário dividir m vezes os sub-intervalos N para se cumprir o erro inserido pelo utilizador.

Para os gráficos destas funções verificamos que o erro absoluto tende a diminuir com o erro pretendido, apresentando porém uns picos. Apesar destes picos, o valor do erro absoluto continua abaixo do erro pretendido.

Estes picos devem-se ao aumento do integral calculado pelo método adaptativo não-iterativo. Verificamos que à medida que o erro pretendido vai diminuindo o programa necessita de subdividir mais vezes os intervalos N para que este erro seja cumprido. Porém, para algumas gamas de erros pretendidos, estas subdivisões fazem com que a diferença entre a estimativa da integral calculada aumente em relação à real e à calculada para menos subdivisões. Visto que o erro é diretamente proporcional à quarta derivada, os dados recolhidos sugerem um aumento da quarta derivada. Uma vez que o número de intervalos é deduzido a partir da equação (34), no anexo 6.5, que depende da estimativa da quarta derivada, este aumento poderá também ser explicado com o erro da aproximação à quarta derivada no subintervalo analisado.

4 Conclusão

De acordo com os dados recolhidos, podemos concluir que com a diminuição do valor erro pretendido, utilizando o método adaptativo não-iterativo, maior será o número de subdivisões que o programa realizará em cada intervalo. N. Estas subdivisões fazem com que o erro absoluto diminua, ou seja, o valor da integral calculada seja mais próximo do valor da integral exata.

Apesar da estimativa da quarta derivada induzir algum erro no cálculo do erro verificamos que o erro absoluto continua abaixo do erro pretendido, o que comprova a eficácia do programa e, visto que mesmo com este erro associado à estimativa da quarta derivada, o valor do erro obtido é menor que o pretendido. A eficácia da Regra de Simpson Composta, utilizando um método adaptativo não-iterativo, fica demonstrada.

5 Referências Bibliográficas

[1] PINA, Heitor. Métodos Numéricos. 1995.

[2] PINA, Heitor. Slides das Aulas Teóricas 2009-10.

[3] FERNANDES, José Leonel. Acetatos das Aulas Teóricas.

[4] ANDRETTA, Marina, Interpolação polinomial: Polinómio de Lagrange. Disponível em: http://conteudo.icmc.usp.br/pessoas/andretta/ensino/aulas/sme0500-1-12/iplagrange.pdf

[5] ALVES, Carlos. *Regra dos Trapézios.* Disponível em: https://www.math.tecnico.ulisboa.pt / calves/courses/integra/capiii32.html. Acesso em 19 nov. 2018.

6 Anexos

6.1 Enunciado do Projeto

Desenvolva um programa para aplicar a regra de Simpson composta numa integração adaptativa não-iterativa com subdivisão do intervalo e controlo do erro local e global através de estimativas por diferenças finitas. Aplique o programa ao cálculo dos seguintes integrais e calcule o erro absoluto e relativo em todos os casos:

$$I = \int_0^3 (xe)^{2x} dx \tag{5}$$

$$I = \int_0^1 (10)^{2x} dx \tag{6}$$

$$\pi = \int_0^1 \frac{4}{x^2+1} dx \tag{7}$$

$$I = \int_0^3 \frac{e^x \sin 2x}{x^2+1} dx \tag{8}$$

Mostre graficamente quais as divisões que o programa realiza em relação ao intervalo inicial e o nível de erro local nesses intervalos.

Trace curvas de evolução do erro com o número de funções calculadas para cada um dos casos.

Verifique em todos os casos se o nível de erro especificado pelo utilizador está efectivamente a ser conseguido.

6.2 Interpolação Quadrática

Primeiramente, é sempre possível aproximar uma função contínua por um polinómio. Além disso, polinómios têm derivadas e integrais mais fáceis de serem calculadas.

A regra de Simpson composta é uma regra de integração de grau 3, sendo o polinómio usado na sua construção de grau 2.

Se a função estudada tiver grau igual ou inferior a 2, então o polinómio quadrático permite uma representação exata da função; caso tenha grau superior a 2, o polinómio será uma aproximação do polinómio que tem um erro associado.

A interpolação quadrática consiste em fazer uma parábola passar por 3 pontos distintos da função, aos quais chamamos nós de interpolação. Para cada intervalo estes 3 pontos serão os pontos iniciais, intermédios e finais: (x_0, y_0), (x_1, y_1), (x_2, y_2).

Existe um só polinómio quadrático que passe pelos 3 pontos, sendo a sua expressão igual a:

$$P(x) = A \cdot x^2 + B \cdot x + C \tag{9}$$

E, visto que o polinómio é uma combinação linear dos valores quadráticos, então também se pode escrever assim:

$$P(x) = y_0 \cdot L_0(x) + y_1 \cdot L_1(x) + y_2 \cdot L_2(x) \tag{10}$$

Sendo L$_o(x), L_1(x)eL_2(x)asfunçõesdeinterpolaçãodeLagrange.$

Sabendo que:

$$L_{n,k}(x) = \frac{(X - X_0) \cdots (X - X_{k-1}) \cdot (X - X_{k+1}) \cdots (X - X_n)}{(X_k - X_0) \cdots (X_k - X_{k-1}) \cdot (X_k - X_{k+1}) \cdots (X_k - X_n)} = \prod_{i=0, i \neq k}^{n} \frac{(X - X_i)}{(X_k - X_i)} \tag{11}$$

Visto que $n = 2$, temos que:

$$L_0(x) = \frac{(X - X_1)(X - X_2)}{(X_0 - X_1)(X_0 - X_2)} \tag{12}$$

$$L_1(x) = \frac{(X - X_0)(X - X_2)}{(X_1 - X_0)(X_1 - X_2)} \tag{13}$$

$$L_2(x) = \frac{(X - X_0)(X - X_1)}{(X_2 - X_0)(X_2 - X_1)} \tag{14}$$

Cada uma das funções tem duas raízes que correspondem aos valores de x dos outros nós da interpolação, assim, por exemplo para x_o, $L_o(x_o) = 1 \ e \ L_1(x_0) = L_2(x_o) = 0$

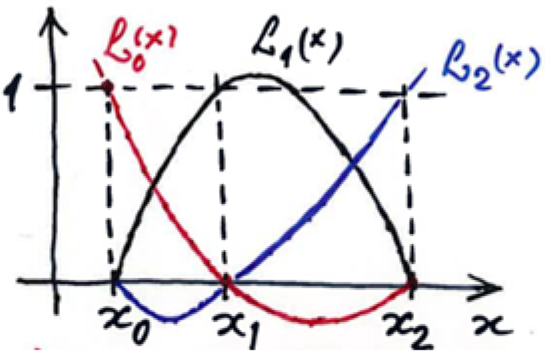

Figura 1: Visualização gráfica da interpolação quadrática

Assim, substituindo os valores na equação do polinómio como combinação linear, é óbvio que: $P(x_0) = y_0$, $P(x_1) = y_1$, $P(x_2) = y_2$

6.3 Regra de Simpson

Como dito na introdução, a regra de Simpson tem por isso grau 3 e o polinómio interpolador tem grau 2. Precisamos assim de 3 pontos, que serão os pontos inicial, intermédio e final do intervalo.

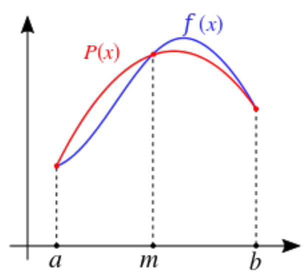

Figura 2: Integral de a a b de $p(x)$ igual$= y_0 \int I_o \ + y_1 \int I_1 \ + y_2 \int I_2$

$$\int_a^b l_0(x)\, dx = \frac{h}{3} \tag{15}$$

$$\int_a^b l_1(x)\, dx = \frac{4h}{3} \tag{16}$$

$$\int_a^b l_2(x)\,dx = \frac{h}{3} \tag{17}$$

Assim, integrando $p(x)$, o integral será, e tendo em conta que como visto no anexo 6.2 se tem $n = 2$ e $h = \frac{|b-a|}{n}$:

$$S(f) = I_2(f) = [f(a) + 4f(c) + f(b)]\frac{h}{3} = [f(a) + 4f(c) + f(b)] \cdot \left(\frac{|b-a|}{6}\right) \tag{18}$$

O erro é dado por:

$$E_s(f) = -\frac{h^5}{90}f^{(4)}(\xi) = -\frac{|b-a|^5}{2880}f^{(4}(\xi) \quad \text{onde} \quad \xi \in]a,b[\tag{19}$$

6.4 Regra de Simpson Composta

É vantajoso usar regras compostas caso a função tenha valores negativos para evitar cancelamento subtrativo, é preferível em relação a regras com grau maior pois as funções podem não ter derivada contínua (condição necessária) até à ordem necessária e é preferível em relação às regras simples pois, como foi dito na introdução, quanto menor for o comprimento dos sub-intervalos menor vai ser o seu erro.

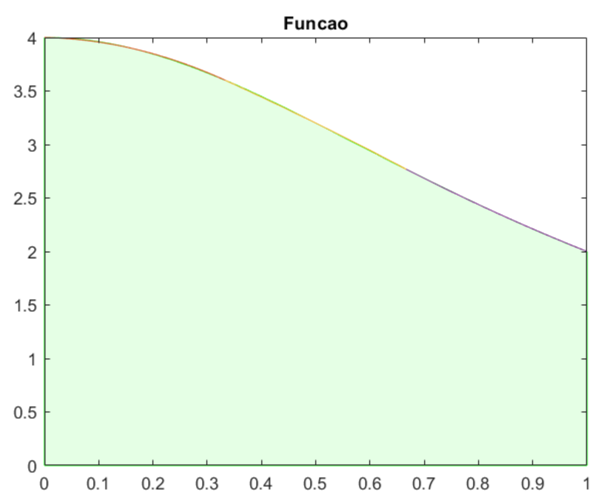

Figura 3: Exemplo da representação de Simpson composta com 3 intervalos para a função $\frac{4}{(x^2)+1}$ sendo as interpolações quadráticas representadas a azul, vermelho e amarelo.

Sendo N o número de sub-intervalos o comprimento de cada sub-intervalo será $h = \frac{b-a}{N}$.
Considerando que os 3 nós de cada sub-intervalo são $[a_i, m_i, b_i]$,
a integral da função é, evidentemente, dada pela soma das integrais de cada sub-intervalo, assim:

$$\int_a^b f(x)\,dx = \sum_{k=1}^{\frac{N}{2}} \int_{a_i}^{b_i} f(x)\,dx \;=\; S_N(f) = \sum_{k=1}^{\frac{N}{2}} \Big(f(a_i) + 4f(m_i) + f(b_i) \Big) \tag{20}$$

$$= \frac{h}{3}\Big(f(x_0) + f(x_N) + 4\sum_{k=1}^{\frac{N}{2}} f(m_i) + 2\sum_{k=1}^{\frac{N}{2}-1} f(b_i) \Big) \tag{21}$$

O erro será obtido analogamente:

$$E_N(f) = -\sum_{i=1}^{\frac{N}{2}} \frac{h^5}{90} f^{(4)} \xi_i = -\frac{(b-a)h^4}{180} f^{(4)}(\xi) \quad \text{onde} \quad \xi \in [a, b] \tag{22}$$

6.5 Fórmulas utilizadas e deduções

Erro estimado total, E, é dado por:

$$E = \sum_{i=1}^{N} |E_i| \tag{23}$$

Seja ε a tolerância do erro global,e a tolerância do erro para cada sub-intervalo e:

$$e = \frac{\varepsilon}{N} \tag{24}$$

Seja h o comprimento do intervalo, erro no intervalo, a_{i-1} , a_i

$$E_h = -\frac{1}{2880} h^5 f^{(4)} \tag{25}$$

Erro no intervalo, a_{i-1} , a_i , subdividido em m sub-intervalos:

$$E_{\frac{h}{m}} = m\Big[-\frac{1}{2880}\Big(\frac{h}{m}\Big)^5 \Big] f^{(4)} \tag{26}$$

Dedução de $f^{(4)}$:

Tem-se que, das equações (25) e (26):

$$E_h = -\frac{1}{2880}h^5 f^{(4)} = I_{exato} - I_h \tag{27}$$

$$E_{\frac{h}{2}} = 2\left[-\frac{1}{2880}\left(\frac{h}{2}\right)^5\right] f^{(4)} = I_{exato} - I_{\frac{h}{2}} \tag{28}$$

Assim, as igualdades referidas podem-se reescrever:

$$I_{exato} = E_h + I_h = E_{\frac{h}{2}} + I_{\frac{h}{2}} \quad \Leftrightarrow \quad -\frac{1}{2880}h^5 \cdot f^{(4)} + I_h = -\frac{2}{2880}\left(\frac{h}{2}\right)^5 f^{(4)} + I_{\frac{h}{2}} \tag{29}$$

$$\Leftrightarrow \quad I_h - I_{\frac{h}{2}} = f^{(4)}\left(-\frac{2}{2880 \cdot 2^5} + \frac{1}{2880}\right)h^5 \tag{30}$$

$$\Leftrightarrow \quad I_h - I_{\frac{h}{2}} = f^{(4)}\left(\frac{1}{3072}\right)h^5 \tag{31}$$

Obtendo-se:

$$f^{(4)} = \frac{I_h - I_{\frac{h}{2}}}{h^5} \cdot 3072 \tag{32}$$

Número de Sub-divisões, m , do sub-intervalo

$$\left| m\left(\frac{h}{m}\right)^5 \cdot \frac{1}{2880} f^{(4)} \right| \leq e \Leftrightarrow m^4 \geq h^5 \cdot \frac{f^{(4)}}{2880 \cdot e} \tag{33}$$

$$m \geq h \cdot \sqrt[4]{\frac{f^{(4)} \cdot h}{2880 \cdot e}} \quad \text{ou} \quad m \geq h \cdot \sqrt[4]{\frac{(b-a)}{2880 \cdot \varepsilon}} \cdot f^{(4)} \tag{34}$$

6.6 Algoritmo

Algoritmo utilizado [GetErrosEIntegrais]

Inicialização

a_0 = a

Definir Número de Sub-intervalos Iguais, N

h = (b-a)/N

Fixar Erro Máximo por Intervalo, ε

Fixar Acumulador para o Erro, E = 0

Fixar Acumulador do Integral Total, I = 0

Fixar Erro Máximo por Sub-intervalo, e $= \varepsilon/N$

Fixar Erro Máximo por Intervalo, ε

Para i = 1 até N fazer:

a$_i = a_0 + h(i-1)$

a$_{i+1} = a_i + h$

Obter Estimativa do Erro do Sub-intervalo, E$_i$[$ErroInt$]

Se $|E_i|$ ¡ e

Calcular Integral do Sub-intervalo I$_i$[$IntegrInterv$]

I = I + I$_i$

$E = E + |E_i|$

Se não:

Calcular número de subdivisões, m [getmi]

h$_i = h/m$

Fixar Erro do Sub-intervalo, E$_i = 0$

Para j = 1 até m fazer:

a$_j = a_i + h(j-1)$

Calcular Integral dos Sub-intervalos, I$_j$[$IntegrInterv$]

Calcular Estimativa do Erro, E$_j$[$ErroInt$]

I=I+I$_j$

$E_i = E_i + |E_j|$

Fim do Ciclo J

E=E+E$_i$

Fim do Ciclo I

O método adaptativo não iterativo utilizado, consiste em analisar previamente o erro estimado para cada sub-intervalo, E_i: se este for menor que o erro admissível para cada sub-intervalo, e, procede-se a ao calculo da integral correspondente; se não, o sub-intervalo é subdividido e procede-se ao calculo da integral para cada sub-divisão, não havendo verificação de se o novo valor satisfaz o erro admissível sendo, por isso, um método não-iterativo

Na implementação deste programa recorreu-se a várias construções e métodos do Matlab, dos quais se consideram fundamentais: if, else, ciclos for, switch e case, e plot, loglog, stairs, zeros, abs e floor.

Assim que inicializado, o programa abre um menu no qual o utilizador escolhe que integral deve ser analisado. O utilizador pode escolher entre uma das quatro integrais indicadas pelo enunciado do projeto, tendo também a opção de inserir o seu integral. Para tal, o utilizador deve selecionar a opção 'Outra'. Uma vez escolhida, esta opção o utilizador insere a função a integrar (No formato exemplificado acima do espaço para o input da mesma) bem como o intervalo de integração. Notou-se que a opção 'Outra' para determinadas funções inseridas pode resultar num erro no MATLAB que pode, dependendo da situação, ser causado por falha ao utilizar a função integral do MATLAB ou por erro na inserção dos inputs. No entanto a equipa que desenvolveu este programa encoraja o utilizador a analisar algumas funções através desta opção.

Após este passo, o utilizador tem a opção de escolher o número de sub-intervalos inicial, bem como o erro máximo pretendido.

O programa tem como função utilizar a regra de Simpson, para calcular o valor aproximado do integral através de um integração adaptativa não-iterativa, calculando também o erro associado a este.

O cálculo dos integrais e erros associados é feito com recurso ao método 'GetErrosEIntegrais', cujo algoritmo é descrito acima. Este método utiliza os valores do intervalo de integração, do erro máximo pretendido, do número de sub-intervalos e a função integrada, tendo como output uma cell array (um tipo de dados do MATLAB que permite simultaneamente o armazenamento de valores escalares e vetores) onde estão inseridos os valores da integral calculada e do erro estimado, bem como, sob a forma de vetores, os pontos dos sub-intervalos (e suas subdivisões) utilizados, os erros estimados para cada sub-intervalo e os erros estimados para cada sub-intervalo e(ou) subdivisões. Note-se que estes dois últimos vetores descritos poderão ser idênticos no caso de não ocorrerem subdivisões dos sub-intervalos.

Este método chama direta ou indiretamente os métodos com as seguintes denominações: 'IntegrInterv', 'IntegrItervDois', 'QuartaDeriv', 'getmi' e 'ErroInt', que utilizam algumas das equações indicadas ao longo deste documento . Note-se que para efeitos de calculo do erro, são considerados apenas valores absolutos, para que não ocorra cancelamento entre os erros estimados para cada sub-intervalo. A dinâmica entre 6 métodos é explicada através do seguinte fluxograma (figura 4).

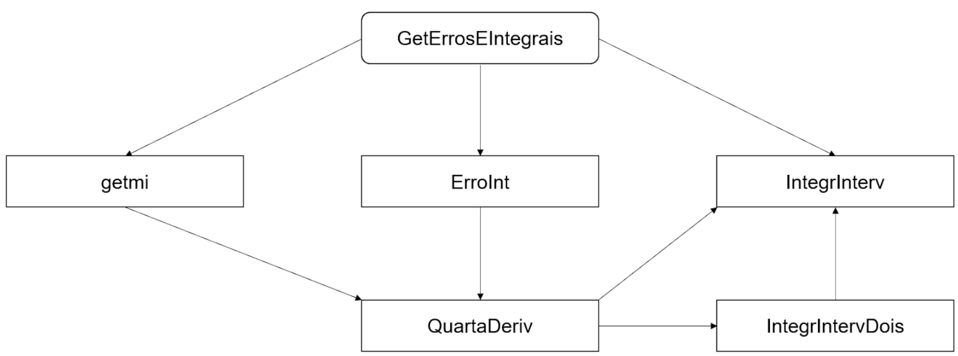

Figura 4: Fluxograma dos métodos criados

Uma vez chamado o método 'GetErrosEIntegrais', que retorna uma *cellarray*, os dados são retirados da cell array, para que possam ser tratados durante o resto do programa.

Os dados são então organizados em três gráficos.

O primeiro gráfico apresenta a curva da função, bem como a área por baixo desta, e também as curvas dos polinómios interpoladores correspondentes a cada sub-intervalo/subdivisão. A interpolação dos polinómios é feita por interpolação quadrática.

O segundo gráfico mostra o erro estimado relacionado a cada sub-intervalo e respectivas subdivisões.

O terceiro gráfico trata a evolução do erro real, isto é o módulo da diferença Integral Exato - Integral Estimado, em função do erro máximo pretendido (ou tolerância do erro). Os valores de erro máximo pretendido são gerados pela progressão geométrica de termo $a_n = 0.00001 * 1.01^{(n-1)}$, com $n = 1$ até 1500. O termo geral da progressão geométrica foi criado por tentativa e erro, até que os termos gerados por esta tivessem significado e relevância do ponto de vista da interpretação dos erros.

Consoante o erro pretendido seja satisfeito ou não, o programa irá exibir uma mensagem na *commandwindow* do MATLAB, e abaixo os valores relevantes para a análise da integral escolhida.

A informação sobre os gráficos e complementada abaixo.

6.7 Discussão dos resultados- Gráficos

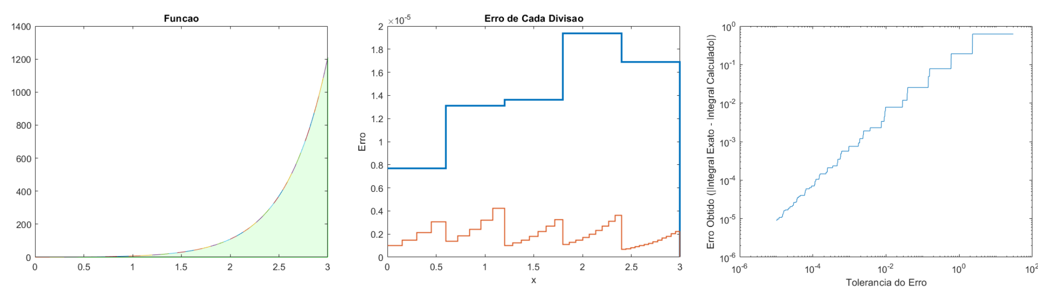

Figura 5: $I = \int_0^3 (xe)^{2x} dx$

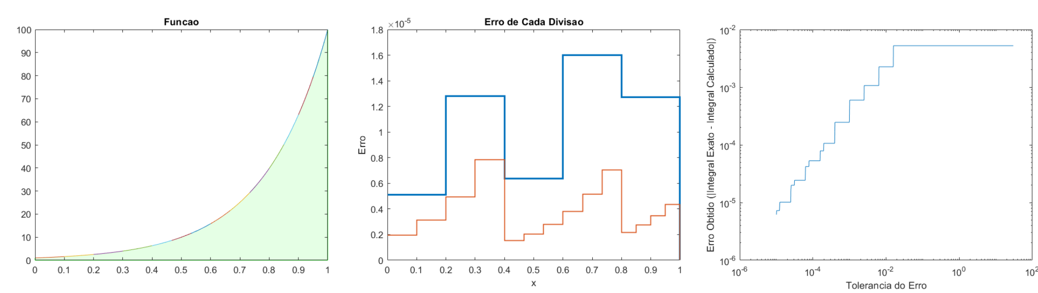

Figura 6: $I = \int_0^1 (10)^{2x} dx$

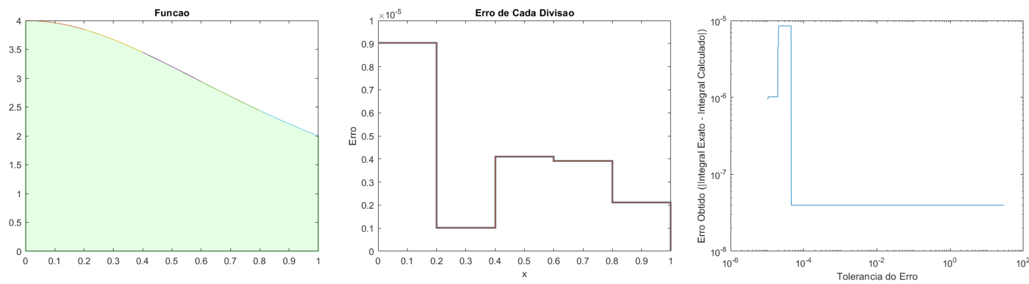

Figura 7: $\pi = \int_0^1 \frac{4}{x^2+1} dx$

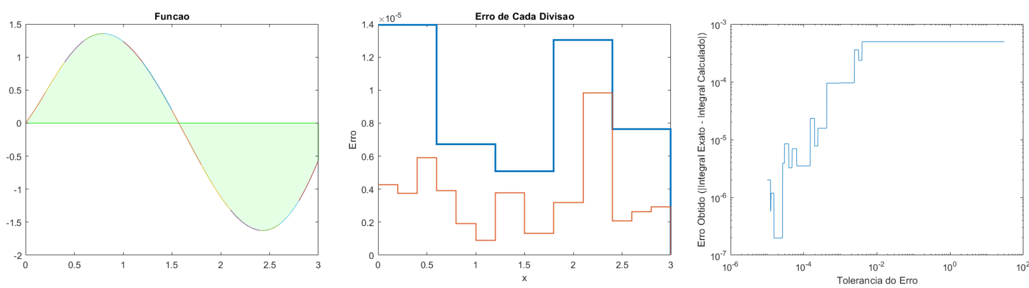

Figura 8: $I = \int_0^3 \frac{e^x \sin 2x}{x^2+1} dx$

6.8 Gráficos

O gráfico 1 (figura 9) tem como objetivo representar a área da função escolhida no intervalo de integração e ao mesmo tempo mostrar como é que está desenhado o polinómio interpolador para cada intervalo e, quando é necessário dividir em sub-intervalos, para sub-intervalos.

Os dois gráficos seguintes estão desenhados para $N = 2$ e para um erro muito grande para não haver sub-divisões e evidenciar os polinómios desenhados e cada polinómio tem uma cor diferente para se distinguir.

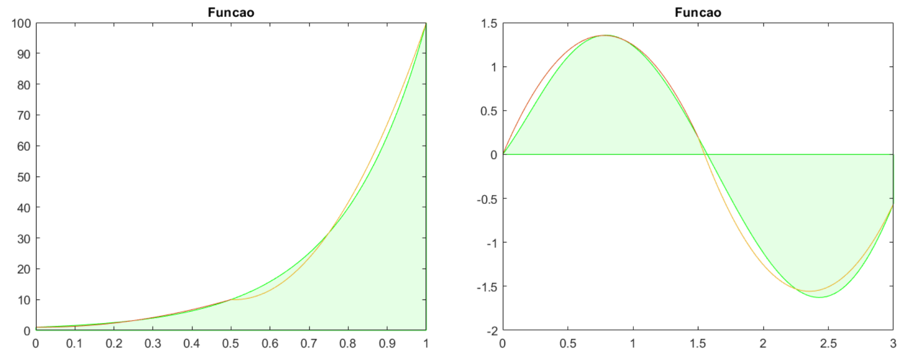

Figura 9: Visualização da função e dos polinómios interpoladores

Usando uma tolerância de erro total 0.0001 em que é necessário haver sub-divisões continuamos a distinguir os polinómios interpoladores de cada sub-intervalo.

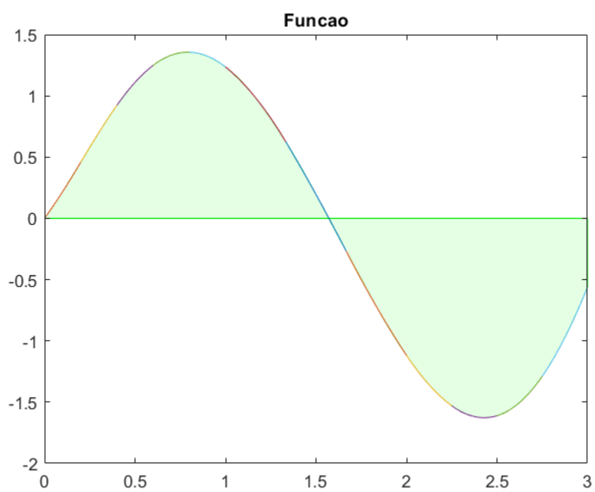

Figura 10: Gráfico 1 para tolerância do erro total de 0.0001 e N = 2

O gráfico 2 (figura 11) representa o erro de cada intervalo N e, caso haja necessidade de dividir o intervalo em m sub-intervalos representa também o erro desses sub-intervalos, sendo a sua soma igual ao erro do intervalo.

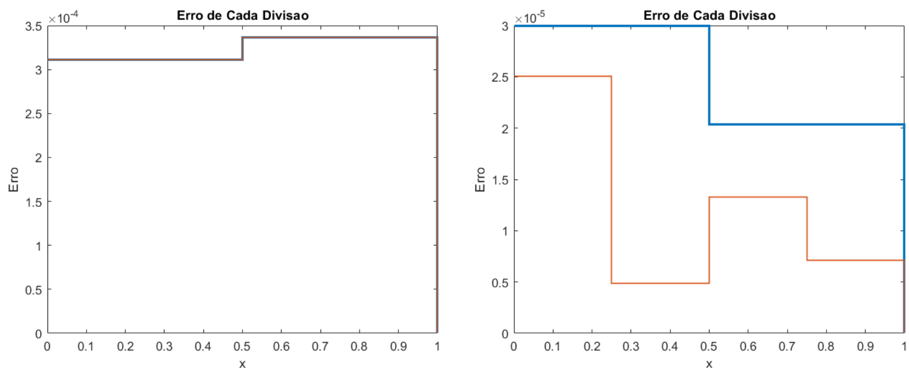

Figura 11: Representação do erro dos sub-intervalos: À esquerda sem subdivisão; à direita com subdivisão

O terceiro gráfico representa em escala logarítmica, para ambos os eixos, o erro absoluto obtido em função do erro pretendido.

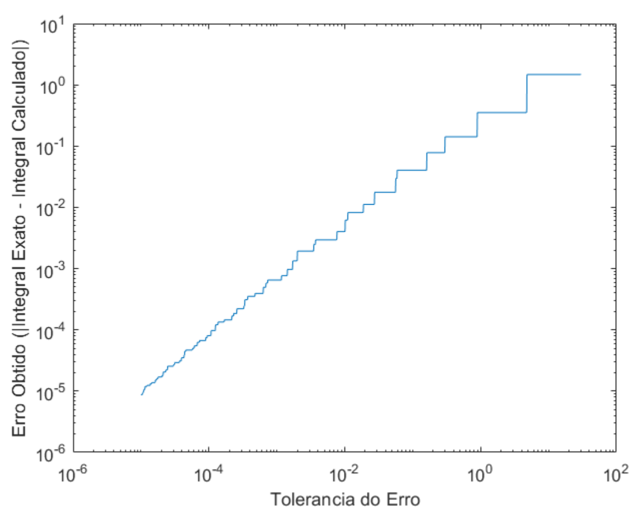

Figura 12: Evolução do erro real com a tolerância do erro

6.9 Código

clear

choice = menu('Escolha a função a integrar', 'I - x.*exp(2*x)' ,'II - 10.$^{(}2 * x)', \ldots' III - 4./(x.^2 + 1)',' IV - exp(x). * sin(2 * x)./(x.^2 + 1)',' Outra'$);

switch choice

case 1

fun = @(x) x.*exp(2*x);

a = 0; % a = limite inferior de integração

b = 3; % b = limite superior de integração

case 2

fun = @(x) 10.$^{(}2 * x)$;

a = 0;

b = 1;

case 3

fun = @(x) 4./((x.^{2}) + 1);

a = 0;

b = 1;

case 4

fun = @(x) exp(x).*sin(2*x)./((x.ˆ2) + 1);

a = 0;

b = 3;

case 5

prompt = 'Funcao **(ex. @(x) 4./((x.^{2}) + 1) $**',' Integrarde',' a'$;

dlg$_t itle =' Input'$;

num$_l ines = 1$;

def = ' ',' ', ' ';

answer = inputdlg(prompt, dlg$_t itle, num_l ines, def$);

fun = str2num(answer1);

a = str2double(answer2);

b = str2double(answer3);

flag = false;

end

prompt = 'Erro absoluto pretendido', 'Numero (inteiro maior que 1) de intervalos inciais';

num$_l ines = 1$;

dlg$_t itle =' Input'$;

def = ' ',' ';

answer = inputdlg(prompt, dlg$_t itle, num_l ines, def$);

ErroPretendidoAbsoluto = abs(str2double(answer1));

N = floor(abs(str2double(answer2)));

FunAvaliadaDados = GetErrosEIntegrais(a, b, ErroPretendidoAbsoluto, N, fun);

ErroTotalFun = FunAvaliadaDados1 ;

ArrayLimitsFun = FunAvaliadaDados2 ; %Recorrendo as funcoes sao calculadas

ArrayErrosPorInt = FunAvaliadaDados3 ; %as seguintes variaveis

IntegralTotalFun = FunAvaliadaDados4 ;

ArrayErroIntESubInt = [FunAvaliadaDados5, 0];

%%% Grafico 1: Representacao da integral escolhida (Area debaixo da fun [U+FFFD] [U+FFFD] o

%%% 1D) e representacao dos subintervalos pela regra de Simpson com

%%% curvatura quadratica

figure(1)

x= linspace(a,b) ;

k=fun(x) ;

ar=area (x,k);

ar.FaceAlpha = 0.1 ;

ar.EdgeColor = 'g';

ar.FaceColor= 'g';

ar.FaceAlpha = 0.1 ;

hold on

for ii=2:length(ArrayLimitsFun) % loop para desenhar os intervalos da regra de simpson com a curvatura (quadr[U+FFFD]tica)

xii = ArrayLimitsFun(ii-1); % so sera visivel se o numero de subintervalos for pequeno e a toler[U+FFFD]ncia for grande,

xfi= ArrayLimitsFun(ii) ; % caso contrario as linhas dos dois gr[U+FFFD]ficos v[U+FFFD]o se sobrepor com a grande proximidade

xmi=(xfi+xii)/2 ; % que as suas representacoes passariam a ter

x= linspace(xii,xfi) ;

lo=@(x) (x-xmi).*(x-xfi)/((xii-xmi)*(xii-xfi));

l1=@(x) (x-xii).*(x-xfi)/((xmi-xii)*(xmi-xfi));

l2=@(x) (x-xii).*(x-xmi)/((xfi-xii)*(xfi-xmi));

yo=fun(xii) ;

y1=fun(xmi) ;

y2=fun(xfi) ;

z= yo * lo(x) + y1 *l1(x)+ y2 *l2(x) ;

plot(x, z)

end

% clear figure

% figure(1)

title('Funcao')

hold off

%%%

%%% grafico 2 representa o erro de cada subdivisao do gafico num grafico em

%%% que o valor do eixo das abcissas representa as coordenas do x do

%%% intervalo e o eixo das ordenas representa o erro associado a cada

%%% intervalo

h = (b - a)/N ;

ArrayN = [] ;

for ii = a:h:b

  ArrayN = [ArrayN, ii] ;

end

figure(2)

ArrayErrosPorInt=[ArrayErrosPorInt,0] ;

stairs(ArrayN ,ArrayErrosPorInt,'LineWidth',2,'MarkerFaceColor','c')

hold on

stairs(ArrayLimitsFun ,ArrayErroIntESubInt,'LineWidth',1,'MarkerFaceColor','r')

hold off

title('Erro de Cada Divisao')

xlabel('x')

ylabel('Erro')

%%%Grafico 3 demonstra a variacao do erro (Integral exata - integral

%%% calculada) em funcao do erro pretendido

figure(3)

IntegralExato = integral(fun,a, b, 'ArrayValue', true);

ErroInicial = 0.00001;

Razao = 1.01;

NRepeticoes = 1500;

Graf3ArrayErroPretendido = zeros (1,NRepeticoes); %Onde serao armazenados os erros maximos do integral

Graf3ArrayErroObtido = zeros(1,NRepeticoes); %One serao armazendos os erros reais

for ii=1:NRepeticoes

  ErroPretendidoAbcissas = ErroInicial*Razao.$^{(}ii-1); \%Progressaogeometricaparagerarconjuntodee$

```
        FunsDat = GetErrosEIntegrais(a, b, ErroPretendidoAbcissas, N, fun);
        IntegralObtido = FunsDat4;
        ErroReal = abs(IntegralExato - IntegralObtido) ;
        Graf3ArrayErroPretendido(ii) = ErroPretendidoAbcissas;
        Graf3ArrayErroObtido(ii) = ErroReal;
    end
    loglog(Graf3ArrayErroPretendido, Graf3ArrayErroObtido)
    ylabel('Erro Obtido (—Integral Exato - Integral Calculado—)')
    xlabel('Tolerancia do Erro')
    %%%Verificacao se o erro especificado esta ser consegido
    if ErroTotalFun ¡ ErroPretendidoAbsoluto
        disp('O erro especificado esta a ser conseguido')
    else
        disp ('Falha, erro estimado maior que o pretendido')
    end
    fprintf('de Erro Pretendida: %f', ErroPretendidoAbsoluto);
    fprintf('Calculado: %s', IntegralTotalFun);
    fprintf('Exato: %s', IntegralExato);
    fprintf('Estimado: %s', ErroTotalFun);
    ErroReal = abs(IntegralTotalFun-IntegralExato);
    fprintf('Real: %s', ErroReal);
    ErroRelativoReal = 100*ErroReal/IntegralExato;
    fprintf('Relativo Real (Em percentagem): %s', ErroRelativoReal);
    function output = IntegrInterv(ai, aii, fun) %Calcula o integral aproximado do intervalo/subintervalo
        output = (aii-ai)*(1/6).*(fun(ai) + fun(aii) + 4.*fun((aii+ai)/2));
    end
```

function output = IntegrItervDois(ai, aii, fun) %Calcula o integral de um intervalo subdividido em dois

xmi = (ai + aii)/2; % xmi = valor medio de aii e bii

output = IntegrInterv(ai, xmi, fun) + IntegrInterv(xmi, aii, fun);

end

function output = QuartaDeriv(ai, aii, fun) %Calcula aproximacao a quarta derivada

output = 3072*(IntegrInterv(ai, aii, fun) - IntegrItervDois(ai, aii, fun))/((aii-ai)^5);

end

function output = getmi(ErroPretendidoAbsoluto, ai, aii, a, b, fun) %Calcula numero, m, de subintervalos necessarios

fd = QuartaDeriv(ai, aii, fun) ;

output = floor((aii-ai)*((abs(fd).*(b-a)*((ErroPretendidoAbsoluto*2880)$^-$1))$^($1./4)))+ 1;

end

function output = ErroInt(ai, aii, fun) %Estima o erro, associado ao calculo do integral no intervalo/subintervalo

output = abs((1/2880)*QuartaDeriv(ai, aii, fun)*((aii-ai)^5));

end

function output = GetErrosEIntegrais(a, b, ErroPretendidoAbsoluto, N, fun)

h=(b-a)/N

ErroTotal = 0 ;

ArrayLimsInterv=[a]; %Onde se armazenam os pontos limites dos intervalos

ArrayDosErros=[]; %Onde se armazenam os erros dos intervalos

IntegralTotal = 0 ; %Soma do integral calculado em cada intervalo

ArrayDosErrosIntESubInt = [] ; %Onde se armazenam os erros dos intervalos e subintervalos

ErroMaximoPorInt = ErroPretendidoAbsoluto/N ;

for ii=1:N

ai= a + h*(ii-1) ; %Calcula os pontos delimitadores

aii= ai + h ; %dos intervalos

ErroIntervAiAii = ErroInt(ai, aii, fun);

if abs(ErroIntervAiAii) ¡ ErroMaximoPorInt

ArrayLimsInterv=[ArrayLimsInterv, aii];

ArrayDosErros = [ArrayDosErros, ErroIntervAiAii];

ArrayDosErrosIntESubInt = [ArrayDosErrosIntESubInt, ErroIntervAi-Aii] ;

IntegralTotal = IntegralTotal + IntegrInterv(ai, aii, fun) ;

ErroTotal = ErroTotal + ErroIntervAiAii ;

else

mi = getmi(ErroPretendidoAbsoluto, ai, aii, a, b, fun) ;

hi = h/mi; %Calculo do tamanho dos subintervalos

ErroAcomuladoSubInts = 0; %Acomulador dos erros nos subintervalos

for iii=1:mi

aSubInt = ai + hi*(iii-1); %aSubInt = inicio do subintervalo

bSubInt = aSubInt + hi; %bSubInt = fim do subintervalo

ErroSubInterv = ErroInt(aSubInt, bSubInt, fun); %Erro associado, no subintervalo

ErroAcomuladoSubInts = ErroAcomuladoSubInts + ErroSubInterv;

ArrayLimsInterv=[ArrayLimsInterv, bSubInt];

ArrayDosErrosIntESubInt = [ArrayDosErrosIntESubInt, ErroSubInterv];

IntegralTotal = IntegralTotal + IntegrInterv(aSubInt, bSubInt, fun) ;

end

```
            ArrayDosErros = [ArrayDosErros, ErroAcomuladoSubInts];
            ErroTotal = ErroTotal + ErroAcomuladoSubInts;
        end
    end
    output = ErroTotal, ArrayLimsInterv, ArrayDosErros, IntegralTotal, ArrayDosErrosIntESubInt;
end
```