Projeto 03: Métodos básicos integro-diferenciais

7600017 - Introdução à Física Computacional - 2023/02
22/10/2023

Prof. Dr. José Abel Hoyos
Gabriel de Freitas de Azeredo (11810964)

Resumo

Cálculo numérico desempenha um papel fundamental na física contemporânea, sendo capaz de resolver problemas que não existem soluções analíticas diretas, bem como permitindo fazer previsões de comportamentos futuros. Neste projeto, nos foi apresentado métodos básicos para integração e diferenciação numérica. Como os projetos anteriores, a linguagem Fortran 77 foi utilizada.

---

1 Tarefa 1 - derivação numérica

1.1 Introdução teórica

O objetivo desta tarefa é calcular as derivadas numéricas por diferenças finitas, utilizando vários passos h, da função

$$f(x) = e^{\frac{\pi}{2}} \tan(2x). \tag{1}$$

A forma mais direta de se diferenciar por este método vem da definição, que em métodos números é chamada também de *derivada para frente de 2 pontos*

$$f'_{2f}(x) = \frac{f(x+h) - f(x)}{h} \tag{2}$$

de forma análoga, temos a *derivada para trás de 2 pontos*

$$f'_{2t}(x) = \frac{f(x) - f(x-h)}{h} \tag{3}$$

em ambos os casos cometemos um erro ao aproximar $f'(x)$ na ordem de h. Com o objetivo de minimizar esse erro, podemos combinar as duas soluções expostas e escrever a *derivada simétrica de 3 pontos*

$$f'_{3s}(x) = \frac{f(x+h) - f(x-h)}{2h} \tag{4}$$

e dessa forma ficamos com um erro na ordem de h^2. De forma semelhante, também é possível minimizar ainda mais o erro da aproximação com mais pontos, com a *derivada simétrica de 5 pontos* com erro na ordem de h^4

$$f'_{5s}(x) = \frac{f(x-2h) - 8f(x-h) + 8f(x+h) - f(x+2h)}{12h}. \tag{5}$$

Utilizando a expansão em Taylor de $f(x)$ podemos chegar em aproximações análogas para derivadas de ordem superiores. Como a *derivada segunda simétrica de três pontos*

$$f''_{3s}(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \tag{6}$$

a *derivada segunda simétrica de cinco pontos*

$$f''_{5s}(x) = \frac{-f(x-2h) + 16f(x-h) - 30f(x) + 16f(x+h) - f(x+2h)}{12h^2} \tag{7}$$

e a *derivada terceira anti-simétrica de 5 pontos*

$$f'''_{5a}(x) = \frac{-f(x-2h) + 2f(x-h) - 2f(x+h) + f(x+2h)}{2h^3}. \tag{8}$$

Para as tarefas, também será necessário o cálculo analítico das derivadas de $f(x)$. Nem sempre é possível, mas para 1 temos, para $x = 1/2$.

$f'(x = 1/2)$	$9,79678201384$
$f''(x = 1/2)$	$64,09832454947$
$f'''(x = 1/2)$	$671,51461345787$

Tabela 1: Derivadas para $f(x)$ em $x = 1/2$.

1.2 Estratégias de resolução

1.2.1 Entradas e saídas de dados

O programa não requer nenhum dado de entrada. Como saída, escreve no arquivo os dados que preenchem a tabela I do enunciado do projeto.

1.2.2 Desenvolvimento do código

A fórmulas foram implementadas diretamente no código, em um *loop* que utiliza todos os passos.

```
program tarefa1

real*8 :: f
real*8 :: x = 0.5d0, df_2f = 0.d0, df_2t = 0.d0, df_3s = 0.d0,
&   df_5s = 0.d0, d2f_3s = 0.d0, d2f_5s = 0.d0, d3f_5a = 0.d0

! definição do vetor com os passos
real*8 :: h_arr(14) = (/5.d-1, 1.d-1, 5.d-2, 1.d-2, 5.d-3, 1.d-3,
&   5.d-4, 1.d-4, 5.d-5, 1.d-5, 5.d-6, 1.d-6, 5.d-7, 1.d-8/)

! valor teórico das derivadas
real*8 :: df_dx = 9.79678201384d0, df2_dx = 64.09832454947d0,
&   df3_dx = 671.51461345787d0
```

```
 open(11, file = "tarefa-1-saida-1.dat")

 do i = 1, 14
    ! loop para cada passo

    ! calcula a derivada para frente de 2 pontos
    df_2f = (f(x + h(i)) - f(x)) / h(i)

    ! calcula a derivada para trás de 2 pontos
    df_2t = (f(x) - f(x - h(i))) / h(i)

    ! calcula a derivada simétrica de 3 pontos
    df_3s = (f(x + h(i)) - f(x - h(i))) / (2.d0 * h(i))

    ! calcula a derivada simétrica de 5 pontos
    df_5s = (f(x - 2 * h(i)) - 8 * f(x - h(i)) + 8 * f(x + h(i)) -
&   f(x + 2 * h(i))) / (12 * h(i))

    ! calcula a segunda derivada simétrica de 3 pontos
    d2f_3s = (f(x + h(i)) - 2 * f(x) + f(x - h(i))) / (h(i)**2)

    ! calcula a segunda derivada simétrica de 5 pontos
    d2f_5s = (-f(x - 2 * h(i)) + 16 * f(x - h(i)) -30 * f(x) + 16 *
&   f(x + h(i)) - f(x + 2 * h(i))) / (12 * h(i)**2)

    ! calcula a segunda derivada simétrica de 5 pontos
    d3f_5a = (-f(x - 2 * h(i)) + 2 * f(x - h(i)) - 2 * f(x + h(i))
&   + f(x + 2 * h(i))) / (2 * h(i)**3)

    ! escreve nos arquivos o desvio em módulo do valor analítico
    ! definido no início do programa e o calculado numericamente
    write(11,*) h(i), abs(df_2f - df_dx), abs(df_2t - df_dx),
&   abs(df_3s - df_dx), abs(df_5s - df_dx), abs(d2f_3s - df2_dx),
&   abs(d2f_5s - df2_dx), abs(d3f_5a - df3_dx)

 end do

 close(11)

 end program tarefa1

 function f(x)

 real*8 :: f
 real*8 :: x
```

```
    f = dexp(x / 2.d0) * dtan(2.d0 * x)

    return
    end function f
```

1.3 Resultados

Os resultados do arquivo de saída foram tratados e expostos na tabela 2. Todos os resultados, bem como os valores teóricos estão truncados para precisão de 10^{-11} . É possível ver que, descartando alguns valores anômalos, que são causados por perda de precisão em alguns cálculos pelo computador, o comportamento geral é muito consonante com o esperado. Métodos com erro na ordem de h^2, por exemplo, melhoram sua precisão com a diminuição de h mais rápido do que métodos com erro na ordem de h. Em alguns casos, existe um número de passos ideal com desvio 0.0, mas com a diminuição do número de passos, o computador se afasta do resultado esperado. Isso ocorre, pois as fórmulas utilizadas tem h no denominador, então para h pequeno demais, a precisão se perde.

h	$f'_{2f}(x)$	$f'_{2t}(x)$	$f'_{3s}(x)$	$f'_{5s}(x)$	$f''_{3s}(x)$	$f''_{5s}(x)$	$f'''_{5a}(x)$
0.5	21.00132761882	5.79727980863	13.39930371373	14.75200083393	94.50642016984	102.80439084503	639.04988257291
0.1	4.92612217561	2.37530472271	1.27540872645	1.22780691055	8.91594443374	8.60379921016	830.41476874181
0.05	1.94152308431	1.36425519847	0.28863394292	0.04029098492	2.01724110611	0.28232666977	117.90521336825
0.01	0.33208879951	0.30967753738	0.01120563107	5.515422×10^{-5}	0.07830913971	0.00038647116	4.13250361271
0.005	0.16309354382	0.15749587519	0.00279883432	3.43127×10^{-6}	0.01955925251	2.404322×10^{-5}	1.02912659934
0.001	0.03216147382	0.03193763287	0.00011192047	5.48×10^{-9}	0.00078213967	3.802×10^{-8}	0.04111423586
0.0005	0.01605260988	0.01599665016	2.797986×10^{-5}	3.5×10^{-10}	0.00019553069	4.9×10^{-9}	0.01028267633
0.0001	0.00320603581	0.00320379743	1.11919×10^{-6}	0.0	7.82573×10^{-6}	9.76×10^{-9}	0.00067436219
5×10^{-5}	0.00160273796	0.00160217837	2.798×10^{-7}	0.0	2.00816×10^{-6}	1.2818×10^{-7}	0.00199017307
1×10^{-5}	0.00032050277	0.00032048043	1.117×10^{-8}	4×10^{-11}	4.56436×10^{-6}	5.30451×10^{-6}	0.72542795266
5×10^{-5}	0.00016024871	0.00016024305	2.83×10^{-9}	4×10^{-11}	2.874233×10^{-5}	4.724604×10^{-5}	4.49262026308
1×10^{-6}	3.204926×10^{-5}	3.204924×10^{-5}	1×10^{-11}	5×10^{-11}	0.00017973266	0.00049429585	5.38079868278
5×10^{-7}	1.602386×10^{-5}	1.602473×10^{-5}	$4.4e - 10$	6.6×10^{-10}	0.00115253497	0.00144859444	2881.19906534263
1×10^{-8}	3.9369×10^{-7}	3.6126×10^{-7}	1.622×10^{-8}	1.992×10^{-8}	11.39684112504	15.467658882	444089881.36467606

Tabela 2: Tabela com resultados para a primeira tarefa.

2 Tarefa 2 - integração numérica

2.1 Introdução teórica

Para integração numérica, os métodos recorrem a definição de integral onde

$$I = \int_a^b f(x)dx = \lim_{N\to\infty} \sum_{i=1}^{N} f(x_i)h \tag{9}$$

para $x_i = a + (i-1)h$ e $h = \frac{b-a}{N}$. Como o computador é capaz de fazer um número grande de contas de forma relativamente rápida, a ideia é dividir o intervalo de integração em N partes e somar as parcelas. Novamente, para estas parcelas, vamos recorrer a expansão de Taylor de $f(x)$. Supondo que $f(x) = f(x_i) + f'(x_i)(x - x_i)$ em um dos sub-intervalos considerado ($[x_i, x_{i+1}]$), chegamos na *regra do trapézio*

$$\int_{x_i}^{x_i+h} f(x)dx = \frac{h}{2}(f(x_{i+1}) + f(x_i)). \tag{10}$$

Ao expandir mais um termo da série de Taylor, $f(x) = f(x_i)+f'(x_i)(x-x_i)+\frac{f''(x_i)}{2}(x-x_i)^2$, chegamos na *regra de Simpson*

$$\int_{x_i-h}^{x_i+h} f(x)dx = \frac{h}{3}(f(x_i + h) + 4f(x_i) + f(x_i - h)) \tag{11}$$

e utilizando outras formas para as derivadas, obtemos, por fim, a *regra de Boole*

$$\int_{x_i-2h}^{x_i+2h} f(x)dx = \frac{2h}{45}(7f(x_i+2h)+32f(x_i+h)+12f(x_i)+7f(x_i-2h)+32f(x_i-h)). \tag{12}$$

O objetivo central desta tarefa é utilizar esses métodos para calcular a integral

$$I = \int_0^1 dx \exp(-x)\cos(2\pi x) \tag{13}$$

2.2 Estratégias de resolução

2.2.1 Entradas e saídas de dados

O programa não espera entrada de dados e como saída escreve no arquivo *tarefa-2x-saida-1.dat* o resultado para todos os passos pedidos.

2.2.2 Desenvolvimento do código

Para facilitar a visualização, o programa foi divido nos itens (a), (b) e (c). Onde os itens seguem a ordem que o método foi exposto na seção de introdução: trapézio, Simpson e Boole, respectivamente. A base do programa é a mesma para todos os casos, um *loop* para cada passo previamente preenchido em um *array* e a aplicação do método percorrendo cada intervalo. Para o item (a) temos:

```
program tarefa2a

real*8 :: f

real*8 :: h(11) = 0.d0
real*8 :: dint_f = 0.d0, x_i = 0.d0

integer*8 :: ia = 0, ib = 1

open(11, file = "tarefa-2a-saida-1.dat")

! inicializa o vetor com os passos (h)

do i = 2, 12
   h(i - 1) = 1.d0 / (3.d0 * 2.d0 ** real(i, 8))
end do

do j = 1, 11

   ! loop com os passos

   N = (ib - ia) / h(j)
   dint_f = 0.d0

   ! percorre o intervalo, aplicando a regra do trapézio

   do i = 1, N + 1

      x_i = real(ia, 8) +  real(i - 1, 8) * h(j)

      dint_f = (f(x_i + h(j)) + f(x_i)) + dint_f
   end do

   dint_f = dint_f * 0.5d0 * h(j)

   write(11,*) dint_f

end do

close(11)

end program tarefa2a

function f(x)

real*8 :: f
```

```
      real*8 :: x
      real*8 :: pi = dacos(-1.d0)

      f = dexp(-x) * dcos(2.d0 * pi * x)

      return
      end function f
```

Para o item (b), a estrutura do código é a mesma, porém o método de Simpson é aplicado ao invés do método do trapézio

```
      do i = 2, N, 2

         x_i = real(ia, 8) + real(i - 1, 8) * h(j)

         dint_f = dint_f + (f(x_i + h(j)) + 4.d0 * f(x_i) +
     &   f(x_i - h(j)))

      end do

      dint_f = dint_f * h(j) / 3.d0
```

e da mesma forma para o item (c), então o método de Boole é aplicado.

```
      do i = 3, N - 1, 4

         x_i = real(ia, 8) + real(i - 1, 8) * h(j)

         dint_f = dint_f +
     &   (7.d0 * f(x_i + 2.d0 * h(j)) + 32.d0 * f(x_i + h(j))
     &   + 12.d0 * f(x_i)
     &   + 7.d0 * f(x_i - 2.d0 * h(j)) + 32.d0 * f(x_i - h(j)))

      end do
```

2.3 Resultados

Os resultados dos arquivos de saída foram organizados e tratados, então expostos na tabela 3. É possível ver que os métodos de Simpson e Boole convergem mais rápido para o valor esperado para um número menor de passos. Os resultados foram truncados para precisão de 10^{-11} e a partir de um número de passos, para essa precisão, os métodos não alteram seu resultado.

h^{-1}	Trapézio	Simpson	Boole
3×2^2	0.04352870085	0.01559527278	0.01562026045
3×2^3	0.03047304073	0.01561497754	0.01561629119
3×2^4	0.02319215742	0.01561615896	0.01561623772
3×2^5	0.0194301164	0.01561623204	0.01561623692
3×2^6	0.0175282177	0.0156162366	0.0156162369
3×2^7	0.01657330377	0.01561623689	0.0156162369
3×2^8	0.01609501625	0.0156162369	0.0156162369
3×2^9	0.01585568514	0.0156162369	0.0156162369
3×2^{10}	0.0157359753	0.0156162369	0.0156162369
3×2^{11}	0.01567610962	0.0156162369	0.0156162369
3×2^{12}	0.01564617414	0.0156162369	0.0156162369
Exato	0.01461024986		

Tabela 3: Resultados para os diferentes métodos de integração.

3 Tarefa 3 - raízes de equações

3.1 Introdução teórica

Caso o produto $f(a)f(b) < 0$, então existe pelo menos um número ímpar de raízes entre os pontos $x = a$ e $x = b$. O primeiro método numérico para encontrar raízes de uma equação consiste basicamente de utilizar deste conceito para k iterações. Encontrando um intervalo em que a desigualdade é satisfeita, calcula-se o ponto médio deste intervalo $x_m = \frac{a+b}{2}$ e então verifica-se novamente a condição de raízes nos dois sub-intervalos resultantes, o processo é repetido até a precisão desejada. Esse método é chamado de *método da bisseção*. Outro método iterativo de se achar uma raiz em um sub-intervalo em que a condição é satisfeita, é utilizar a derivada analítica de f de forma que

$$x_{n+1} = x_n - \frac{f(x_n)}{f(f'(x_n)} \tag{14}$$

nesse caso x_n converge até que um critério de convergência imposto seja respeitado como, por exemplo, $|x_{n+1} - x_n| <$ tolerância. Esse método é chamado de *Newton-Raphson*. Nem toda função tem uma derivada analítica, neste caso utilizamos o *método da secante*

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}. \tag{15}$$

Repare que par utilizar os dois últimos métodos temos que "chutar"um valor inicial (ou dois valores no caso do método da secante). A escolha de bons valores implicará diretamente em uma convergência rápida ou não.

3.2 Estratégias de resolução

3.2.1 Entradas e saídas de dados

O programa não espera entradas e como saída retorna o número da iteração e o valor atual de x_n para cada raiz.

3.2.2 Desenvolvimento do código

Novamente os métodos foram divididos em arquivos diferentes. Antes de aplicar o método, como pedido no enunciado, é realizada uma busca direta por intervalos (x_i, x_{i+1}) promissores. Porém, com passo $h = 0,1$ e de $x = -10$ até $x = 10$, a condição $f(x_i)f(x_{i+1}) < 0$ nunca é satisfeita, pois passamos exatamente pelas raízes, então neste caso o produto se torna 0. Uma forma de resolver este problema seria utilizando outro passo para a busca direta, mas utilizei de outro artifício: adicionei um $\epsilon = 10^{-3}$ em x_{i+1} e utilizo para a condição $f(x_i)f(x_{i+1} + \epsilon) < 0$. Dessa forma, nunca aplico f diretamente na raiz e contorno completamente o problema. Com relação à aplicação dos métodos, foram somente as equações em um *loop* até a condição da tolerância ser respeitada. Para o método da bisseção:

```
program tarefa3a

real*8 :: f
real*8 :: tol = 1.d-6, x_i = 0.d0, x_i_1 = 0.d0, x_m = 0.d0
real*8 :: eps = 1.d-3, h = 0.1d0
integer*8 :: ia = -10, ib = 10, icount = 0

open(11, file = "tarefa-3a-saida-1.dat")

N =  (ib - ia) / h

do i = 1, N

   ! loop para busca direta

   x_i = real(ia + (i - 1) * h, 8)
   x_i_1 = x_i + eps + h


   if (f(x_i) * f(x_i_1).lt.0) then

      icount = 0

      ! primeiro intervalo a = x_i, b = x_i_1

      do while(abs(x_i_1 - x_i).gt.tol)

         icount = 1 + icount

         x_m = (x_i_1 + x_i) / 2

         if(f(x_m) * f(x_i_1).gt.(0.d0)) then
            x_i_1 = x_m
         else
```

```
            x_i = x_m
        end if

        write(11,*) icount, x_i
    end do

  end if
end do

close(11)

end program tarefa3a

function f(x)

real*8 :: f
real*8 :: x

f = x**3 -4 * x**2 -59 * x + 126

return
end function f
```

Como nos códigos da tarefa 2, somente a aplicação do método em si que se diferencia do código. Para o método de Newton-Raphson e secante o algoritmo da busca direta continua exatamente o mesmo, as alterações podem ser vistas abaixo, respectivamente.

```
! aplica o método de newton-raphson

icount = 0

! utiliza o extremo x_i como "chute" inicial para o método

x_n = x_i
x_n_1 = x_n - f(x_n) / df_dx(x_n)

do while(abs(x_n_1 - x_n).gt.tol)

    icount = 1 + icount

    x_n = x_n_1
    x_n_1 = x_n - f(x_n) / df_dx(x_n)

    write(11,*) icount, x_n_1

end do
```

Para o método da secante:

```
        ! aplica o método da secante

        icount = 0

        x_n_minus_1 = x_i
        x_n = x_i_1

        x_n_plus_1 = x_n - f(x_n) * (x_n - x_n_minus_1) /
     &  (f(x_n) - f(x_n_minus_1))

        do while(abs(x_n_plus_1 - x_n).gt.tol)

           icount = 1 + icount

           x_n_minus_1 = x_n
           x_n = x_n_plus_1

           x_n_plus_1 = x_n - f(x_n) * (x_n - x_n_minus_1) /
     &     (f(x_n) - f(x_n_minus_1))

           write(11,*) icount, x_n_plus_1

        end do
```

3.3 Resultados

Os resultados dos arquivos de saída foram organizados e tratados, então expostos nas tabelas 4, 5 e 6. Todos os resultados foram truncados com 10^{-11}. Para o método da bisseção, como esperado, são necessárias várias iterações até o resultado convergir para a tolerância definida. Já os métodos de Newton-Raphson e secante convergem bem mais rapidamente, mas isso também se deve ao fato do "chute"inicial ser muito próximo da raiz, por consequência do passo da busca direto requerido no enunciado.

i	Método da bisseção		
1	-7.0495	1.9505	8.9505
2	-7.02425	1.97575	8.97575
3	-7.011625	1.988375	8.988375
4	-7.0053125	1.9946875	8.9946875
5	-7.00215625	1.99784375	8.99784375
6	-7.000578125	1.999421875	8.999421875
7	-7.000578125	1.999421875	8.999421875
8	-7.00018359375	1.99981640625	8.99981640625
9	-7.00018359375	1.99981640625	8.99981640625
10	-7.00008496094	1.99991503906	8.99991503906
11	-7.00003564453	1.99996435547	8.99996435547
12	-7.00001098633	1.99998901367	8.99998901367
13	-7.00001098633	1.99998901367	8.99998901367
14	-7.00000482178	1.99999517822	8.99999517822
15	-7.0000017395	1.9999982605	8.9999982605
16	-7.00000019836	1.99999980164	8.99999980164
17	-7.00000019836	1.99999980164	8.99999980164
Exato	-7	2	9

Tabela 4: Resultados para o método da bisseção.

i	Método de Newton-Raphson		
1	-7.00000049618	1.99999999744	9.00000092434
2	-7.0	2.0	9.0
Exato	-7	2	9

Tabela 5: Resultados para o método de Newton-Raphson.

i	Método da secante		
1	-7.00000000298	1.9999999999	8.99999999571
2	-7.0	2.0	9.0
Exato	-7	2	9

Tabela 6: Resultados para o método da secante.