UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS

Introdução à Física Computacional - Projeto 3

Silvio Lacerda de Almeida

Outubro, 2023
São Carlos, SP

Sumário

Lista de Figuras

1 Introdução

O segundo projeto de Introdução à Física Computacional contempla 3 tarefas, e delas explicarei detalhadamente o pensamento que originou o código, assim como discutirei o resultado dos mesmos.

Todos os códigos foram escritos em Fortran 77, aderindo à sintaxe implícita, ou seja, quaisquer variáveis declaradas tendo inicialmente letras no intervalo [i-n] são inteiras, e o resto são reais de precisão dupla. A declaração de variáveis entretanto se fez obrigatória quando foi necessário criar funções e instanciar as suas variáveis em seus respectivos escopos.

2 Tarefas

2.1 Tarefa 1

Figura 1: Tarefa 1 - parte 1

```
      program Derivada
      implicit real*8 (a-h, o-z)
      real*8 f
      real*8 frente2, tras2, simetrica3, simetrica5
      real*8 segundaSimetrica5, terceiraAntiSimetrica5
      dimension vec_h(14)
      vec_h = [5.0E-1, 2.0E-1, 1.0E-1, 5.0E-2, 1.0E-2, 5.0E-3, 1.0E-3
     &,5.0E-4, 1.0E-4, 5.0E-5, 1.0E-5, 1.0E-6, 1.0E-7, 1.0E-8]
      x0 = 1d0/2d0
      df1 = 9.79678201383810
      df2 = 64.0983245494729
      df3 = 671.514613457867

      open(unit=1, file="saida-1-13783203.txt")
      write(1,101) "h", "|",
     & "Simétrica 3 pontos", "|",
     & "P/frente 2 pontos", "|",
     & "P/trás 2 pontos", "|",
     & "Simétrica 5 pontos", "|",
     & "Segunda simétrica 5 pontos", "|",
     & "Terceira anti-simétrica 5 pontos"
101   format(A,9X,A,A,A,A,A,A,A,A,A,A,A,A)

      do i=1, 14
      write(1,100) vec_h(i), "|",
     & simetrica3(x0, vec_h(i), df1), "|",
     & frente2(x0, vec_h(i), df1), "|",
     & tras2(x0, vec_h(i), df1), "|",
     & simetrica5(x0, vec_h(i), df1), "|",
     & segundaSimetrica5(x0, vec_h(i), df2), "|",
     & terceiraAntiSImetrica5(x0, vec_h(i), df3)
100   format(F10.8, A, F24.16, A, F24.16, A, F24.16, A, F24.16, A,
     & F24.16, A, F30.16)
      end do

      write(1,*) "EXATOS", "|",
     &  df1, "|",
     &  df1, "|",
     &  df1, "|",
     &  df1, "|",
     &  df2, "|",
     &  df3

      close(unit=1)


      end program

      real*8 function f(x)
      real*8 x
      f = exp(x/2)*tan(2*x)
      return
      end function
```

Figura 2: Tarefa 1 - parte 2

```
real*8 function frente2(x, h, df)
real*8 f, x, h, df
frente2 = (f(x + h) - f(x))/h
frente2 = abs(frente2-df)
return
end function

real*8 function tras2(x, h, df)
real*8 f, x, h, df
tras2 = (f(x) - f(x - h))/(h)
tras2 = abs(tras2-df)
return
end function

real*8 function simetrica3(x, h, df)
real*8 f, x, h, df
simetrica3 = (f(x + h) - f(x - h))/(2*h)
simetrica3 = abs(simetrica3-df)
return
end function

real*8 function simetrica5(x, h, df)
real*8 f, x, h
real*8 A, B, C, D, df
A = f(x-2*h)
B = f(x-h)
C = f(x+h)
D = f(x+2*h)
simetrica5 = (A - 8*B + 8*C - D)/(12*h)
simetrica5 = abs(simetrica5-df)
return
end function

real*8 function segundaSimetrica5(x, h, df)
real*8 f, x, h
real*8  A, B, C, D, E, df
A = f(x-2*h)
B = f(x-h)
C = f(x)
D = f(x+h)
E = f(x+2*h)
segundaSimetrica5 = (-A + 16*B - 30*C + 16*D - E)/(12*(h**2))
segundaSimetrica5 = abs(segundaSimetrica5-df)
return
end function

real*8 function terceiraAntiSimetrica5(x, h, df)
real*8 f, x, h
real*8 A, B, C, D, df
A = f(x-2*h)
B = f(x-h)
C = f(x+h)
D = f(x+2*h)
terceiraAntiSimetrica5 = (-A + 2*B - 2*C + D)/(2*(h**3))
terceiraAntiSimetrica5 = abs(terceiraAntiSimetrica5-df)
return
end function
```

2.1.1 Explicando a tarefa 1

Começo instanciando as funções que usarei no código, e estabeleço que todas as variáveis que eu usar no escopo do programa principal, e logo depois defino um vetor *vec_h* que guardará todos os "h"que eu vou usar no código. Também defino o ponto que eu vou derivar, e também em três variáveis o valor analítico de cada derivada neste ponto, sendo assim *df1* para a primeira derivada, *df2* para a segunda derivada e *df3* para a terceira derivada.

Logo em seguida eu abro o arquivo *saida-1-13783203.txt* para guardar todos os dados resultantes. Começo então fazendo o cabeçalho da tabela, para me referenciar a cada método de cálculo numérico para derivação.

Logo depois faço um loop, onde coloco em cada coluna o valor da derivação em *x0*, sendo *x0* o valor do meu ponto a ser derivado, e este vale 1/2.

Logo depois eu faço uma coluna para guardar todos os valores exatos da primeira, segunda e terceira derivada.

Por fim, fecho o arquivo.

Logo abaixo estão todas as funções que foram necessárias no código, criando a *f(x)* e logo abaixo as funções para encontrar a primeira derivada pegando 2 pontos para frente, 2 pontos para trás, 3 pontos simétricos ou 5 pontos simétricos, em seguida eu crio uma função para calcular a segunda pegando 5 pontos simétricos e por último crio uma função que vai calcular a terceira derivada pegando 5 pontos simétricos. Em cada função eu pego três valores, o x que está sendo derivado, o valor de h e o valor analítico da derivada, que usarei para mostrar a diferença entre o analítico e o calculado através dos métodos.

2.1.2 Resultados

Figura 3: Resultado tarefa 1 - Parte 1

```
h         |Simétrica 3 pontos      |P/frente 2 pontos       |
0.50000000|     13.3993032398039578|     21.0013271448963579|
0.20000000|      8.7850566254811007|     21.3424071640949009|
0.10000000|      1.2754092436940443|      4.9261227623993964|
0.05000000|      0.2886344257177047|      1.9415235932877017|
0.01000000|      0.0112061044894354|      0.3320892657455499|
0.00500000|      0.0027993081151276|      0.1630940140350887|
0.00100000|      0.0001123944044412|      0.0321619492716376|
0.00050000|      0.0000284537848643|      0.0160530845667797|
0.00010000|      0.0000015931111008|      0.0032065096465104|
0.00005000|      0.0000007537180799|      0.0016032118403455|
0.00001000|      0.0000004851195960|      0.0003209767165160|
0.00000100|      0.0000004741305020|      0.0000325232717007|
0.00000010|      0.0000004722444444|      0.0000036774582792|
0.00000001|      0.0000004608630064|      0.0000007939299156|
 EXATOS   |   9.7967815399169922   |   9.7967815399169922   |
```

Figura 4: Resultado tarefa 1 - Parte 2

```
P/trás 2 pontos         |Simétrica 5 pontos      |Segunda simétrica 5 pontos|
      5.7972793347115559|     14.7520003600106726|    102.8043939322834035|
      3.7722939131326951|     17.8687106625418295|    125.2259161682299293|
      2.3753042750113069|      1.2278065502349751|      8.6038030935549017|
      1.3642547418522923|      0.0402905136077418|      0.2823297752869678|
      0.3096770567666809|      0.0000546802885761|      0.0003895583739109|
      0.1574953978048317|      0.0000029573429625|      0.0000271304355834|
      0.0319371604627552|      0.0000004684392643|      0.0000031243779688|
      0.0159961769970494|      0.0000004735782646|      0.0000030884067428|
      0.0032033234243105|      0.0000004739196662|      0.0000031400369096|
      0.0016017044041856|      0.0000004739204051|      0.0000029475982330|
      0.0003200064773239|      0.0000004739285480|      0.0000066335388595|
      0.0000315750106985|      0.0000004739824728|      0.0001399616417359|
      0.0000027329693904|      0.0000004722444444|      0.0318535101826143|
      0.0000001277960973|      0.0000004590126341|      4.5504635536921683|
      9.7967815399169922|   9.7967815399169922   |   64.098327636718750  |
```

Figura 5: Resultado tarefa 1 - Parte 3

```
Terceira anti-simétrica 5 pontos
             639.0498565173825227
            2034.0626523543558051
             830.4148441940824341
             117.9052434521836403
               4.1325294820693443
               1.0291526081023221
               0.0411398569598305
               0.0103054152459663
               0.0007513094249134
               0.0001368690620893
               0.0516512213088163
             116.4030708852000089
          221373.0825531244918238
       222043937.4588528871536255
               671.51458740234375
```

2.2 Tarefa 2

Figura 6: Tarefa 2: parte 1

```
1           program Integral
2           implicit real*8 (a-h, o-z)
3           real*8 f, trapezio, simpson, boole
4           dimension vec_n(10)
5           pi = 4d0*atan(1d0)
6           f_analitico = 1/(1+4*pi**2)-1/(dexp(1d0)*(1+4*pi**2))
7           open(unit=1, file="saida-2-13783203.txt")
8           write(1,*) "N", "|",
9         & "h", "|",
10        & "Regra do Trapezio", "|",
11        & "Regra do Simpson", "|",
12        & "Regra de Boole"
13
14          do i=1, 10
15          n = 12*2**i
16          write(1,101) n, "|",
17        & 1d0/n, "|",
18        & trapezio(0d0, n, f_analitico), "|",
19        & simpson(0d0, n, f_analitico), "|",
20        & boole(0d0, n, f_analitico)
21  101     format(I6,A,F18.16,A,F18.16A,F18.16,A,F18.16)
22          end do
23
24          write(1,102) "EXATOS", "|",
25        & f_analitico, "|",
26        & f_analitico, "|",
27        & f_analitico
28  102     format(A, A, F18.16, A, F18.16, A, F18.16, A, F18.16)
29
30          close(unit=1)
31          end program
32
33          real*8 function f(x)
34          real*8 x
35          real*8 PI
36          PI = 4d0*atan(1d0)
37          f = exp(-x)*cos(2*PI*x)
38          return
39          end function
40
41          real*8 function trapezio(x, n,an)
42          implicit real*8 (a-h, o-z)
43          real*8 f, x, an
44          integer n
45          trapezio = 0d0
46          h = 1d0/n
47          do j=0,n-1,2
48            trapezio=trapezio+(h/2)*(f(j*h)+2*f(h*(j+1))+f(h*(j+2)))
49          end do
50          trapezio = abs(trapezio-an)
51          return
52          end function
```

Figura 7: Tarefa 2: parte 2

```
          real*8 function simpson(x,n,an)
          implicit real*8 (a-h, o-z)
          real*8 f, x, an
          integer n
          simpson=0d0
          h = 1d0/n
          do j=0,n-1,2
            simpson=simpson+(h/3)*(f(h*(j+2))+4*f(h*(j+1))+f(h*j))
          end do
          simpson = abs(simpson-an)
          return
          end function

          real*8 function boole(x,n,an)
          implicit real*8 (a-h, o-z)
          real*8 f, x, an
          integer n
          boole=0d0
          h = 1d0/n
          do j=0,n-1,4
            boole=boole+(2*h/45)*(7*f(j*h)+32*f(h*(j+1))+12*f(h*(j+2))+
     & 32*f(h*(j+3))+7*f(h*(j+4)))
          end do
          boole = abs(boole-an)
          return
          end function
```

2.2.1 Explicando a tarefa

Começo o código instanciando as funções que eu vou usar, e puxando o valor analítico da integral. Logo em seguida abro o arquivo de saída. Logo depois eu aplico cada uma das funções para integração. As funções rodam N iterações conforme, e cada $h = 1/N$. As funções foram retiradas diretamente do arquivo do projeto.

2.2.2 Resultados

Figura 8: Resultado tarefa 2 - Parte 1

```
  N    |h                  |Regra do Trapezio |
     24|0.0416666666666667|0.0009917641952723|
     48|0.0208333333333333|0.0002288825917523|
     96|0.0104166666666667|0.0000557170032887|
    192|0.0052083333333333|0.0000014290231706|
    384|0.0026041666666667|0.0000003572415666|
    768|0.0013020833333333|0.0000000893095025|
   1536|0.0006510416666667|0.0000000223273198|
   3072|0.0003255208333333|0.0000000055818265|
   6144|0.0001627604166667|0.0000000013954562|
  12288|0.0000813802083333|0.0000000003488638|
 EXATOS                   |0.0156162369044908|
```

Figura 9: Resultado tarefa 2 - Parte 2

```
Regra do Simpson|Regra de Boole
0.0000012593659990|0.0000000542845778
0.0000000779427544|0.0000000008187953
0.0000000048595324|0.0000000000126824
0.0000000003035355|0.0000000000001977
0.0000000000189681|0.0000000000000031
0.0000000000011853|0.0000000000000000
0.0000000000000741|0.0000000000000000
0.0000000000000046|0.0000000000000001
0.0000000000000007|0.0000000000000001
0.0000000000000001|0.0000000000000003
0.0156162369044908|0.0156162369044908
```

2.3 Tarefa 3

Figura 10: Tarefa 3 - parte 1

```
1           program Raizes
2           implicit real*8 (a-h, o-z)
3           real*8 newtonRaphson, metodoSecante
4           dimension r(3)
5           r = [-7, 2, 9]
6
7           open(unit=1, file="saida-3-13783203.txt")
8           write(1,*) "Método da Procura Direta"
9           call procuraDireta()
10
11
12          write(1,*) "Método de Newton-Raphson"
13
14          x1 = r(1)+1
15          x2 = r(2)+1
16          x3 = r(3)+1
17          do i=1, 6
18            x1 = newtonRaphson(x1)
19            x2 = newtonRaphson(x2)
20            x3 = newtonRaphson(x3)
21            write(1,*) i, x1, x2, x3
22          end do
23
24          write(1,*) "Método da secante"
25
26          y1 = r(1) - 0.5
27          x1 = r(1) + 0.5
28          x2 = r(2) + 0.5
29          y2 = r(2) - 0.5
30          x3 = r(3) + 0.5
31          y3 = r(3) - 0.5
32          do i=1,6
33            x1 = metodoSecante(x1,y1)
34            x2 = metodoSecante(x2,y2)
35            x3 = metodoSecante(x3,y3)
36            write(1,*) i, x1, x2, x3
37          end do
38
39          close(unit=1)
40          end program
```

Figura 11: Tarefa 3 - parte 2

```
1           real*8 function f(x)
2           real*8 x
3           f = x**3 - 4*x**2 - 59*x + 126
4           return
5           end function
6
7           real*8 function df(x)
8           real*8 x
9           df = 3*x**2 - 8*x - 59
10          return
11          end function
12
13          subroutine procuraDireta()
14          implicit real*8 (a-h, o-z)
15          dimension r_pd(3)
16          dimension cont_pd(3)
17          x0 = -10
18          dx = 1e-1
19          j = 1
20          do while(j .lt. 4)
21          do while(f(x)*f(x+dx) .gt. 0)
22             x=x0+i*dx
23             i=i+1
24          end do
25          a=x
26          b=x+dx
27          c=b-a
28          cont=0
29          xm=(a+b)/2
30          do while((c .gt. 1e-6) .or. (abs(f(xm)) .gt. 10e-6))
31            if (f(xm)*f(a).LT.0) then
32              b=xm
33            else if(f(xm)*f(a) .gt. 0) then
34              a=xm
35            end if
36          c=b-a
37          xm=(a+b)/2
38          cont=cont+1
39          if(cont .gt. 100) then
40              ! limite de iterações excedido
41                stop
42          end if
43          end do
44          r_pd(j)=xm
45          cont_pd(j) = cont
46          x=b+dx
47          j=j+1
48                end do
49                write(1,*) maxv(cont_pd), r_pd(1), r_pd(2), r_pd(3)
50          return
51          end
```

Figura 12: Tarefa 3 - parte 3

```
1           integer function maxv(arrr)
2           implicit real*8 (a-h, o-z)
3           dimension arrr(3)
4           temp = arrr(1)
5           do l=1, 3
6                    if (arrr(l) .GT. temp) then
7                      temp = arrr(l)
8                     end if
9           end do
10          maxv = temp
11          end function
12
13          real*8 function newtonRaphson(x)
14          real*8 x, f, df
15          newtonRaphson = x - f(x)/df(x)
16          return
17          end function
18
19          real*8 function metodoSecante(x,y)
20          real*8 x, y, f
21          metodoSecante = x - f(x)*((x-y)/(f(x)-f(y)))
22          return
23          end function
```

2.3.1 Explicando a tarefa 3

Começo o código instanciando as funções que usarei, no caso, preciso fazer isto apenas para os métodos de Newton-Raphson e Secante, porque o método da Procura Direta eu coloquei numa subroutine. Logo em seguida define um vetor com as três raízes do polinômio, pois, como todos os métodos se baseiam em pegar as raízes contidas em intervalos, o código fica bem mais rápido se pegar um intervalo que garantidamente está uma das raízes, como pegar o intervalo $(r_1 - 1, r_1 + 1)$, sendo r_1 uma das raízes.

Depois eu abro o arquivo de saída, e coloco abaixo método da Procura Direta, que explicarei mais tarde.

Em seguida, crio um x para cada raíz e calculo utilizando o método de Newton-Raphson.

No bloco de código seguinte, crio um intervalo para cada raíz utilizando o eixo x-y para assim poder utilizar o método da secante. E por fim, fecho o programa.

Logo abaixo temos as funções que utilizei neste código, como a f(x) e a derivada de f(x). Abaixo também está a subroutine de *procuraDireta*, aonde instancio um vetor para guardar as raízes através deste método, assim como o número de iterações. Especificamente para este método utilizarei muitas iterações, para ficar dentro da faixa de erro desenhada que é $1e-6$. Defino um *x0* e também um um *dx*. As linhas seguintes são aplicação direta do método da Procura Direta só que sem se restringir às 6 interações para respeitar o critério de erro exigido. As outras funções também são aplicações diretas do que foi mostrado no arquivo de apresentação do projeto.

2.4 Tarefa 3

Figura 13: Tarefa 3 : Cada coluna é uma raiz, e a primeira coluna é o número de iterações.

```
    Método da Procura Direta
19  -7.0000000506639495        1.9999999880790682        8.9999999970197635
 Método de Newton-Raphson
1   -7.2371134020618557        1.9285714285714286        9.1552795031055894
2   -7.0091795388655305        1.9998502424919649        9.0047146488304328
3   -7.0000145933586699        1.9999999992881354        9.0000045577040666
4   -7.0000000000369722        2.0000000000000000        9.0000000000042650
5   -7.0000000000000000        2.0000000000000000        9.0000000000000000
6   -7.0000000000000000        2.0000000000000000        9.0000000000000000
 Método da secante
1   -6.9566724436741767        2.0079681274900398        8.9487750556792882
2   -6.9964568053514968        2.0000942584113650        9.0057727737341313
3   -6.9997116714389671        2.0000011089946299        8.9993560624701647
4   -6.9999765466027251        2.0000000130470057        9.0000719119777326
5   -6.9999980923018743        2.0000000001534941        8.9999919702298481
6   -6.9999998448283449        2.0000000000018057        9.0000008966257461
```