Projeto 3 - Métodos Básicos Integro-Diferenciais

Instituto de Física de São Carlos
Universidade de São Paulo

Gabriel Lima Alves (12558547)

Introdução à Física Computacional
Prof. Francisco Castilho Alcaraz

Novembro, 2022

Tarefa A

Na tarefa A é pedido para calcular numericamente as derivadas de primeira, segunda e terceira ordem no ponto $x = \frac{1}{2}$ com uma precisão de 10^{-11} (precisão dupla) da seguinte função:

$$f(x) = cosh(3x) \cdot sen(\frac{x}{2})$$

Para isso, é utilizado métodos computacionais, que serão demonstrados abaixo. Além disso, também é pedido para calcular o erro de cada interação dos métodos em relação ao valor obtido derivando analiticamente a função.

Os métodos numéricos utilizados são os seguintes:

Derivada simétrica (3 pontos):

$$f'(x_0) = \frac{f_1 - f_{-1}}{2h} + O(h^2)$$

Derivada para frente (2 pontos):

$$f'(x_0) = \frac{f_1 - f_0}{h} + O(h)$$

Derivada para trás (2 pontos):

$$f'(x_0) = \frac{f_0 - f_{-1}}{h} + O(h)$$

Derivada simétrica (5 pontos):

$$f'(x_0) = \frac{f_{-2} - 8f_{-1} + 8f_1 - f_2}{12h} + O(h^4)$$

Derivada segunda simétrica (5 pontos):

$$f''(x_0) = \frac{-f_{-2} + 16f_{-1} - 30f_0 + 16f_1 - f_2}{12h^2} + O(h^4)$$

Derivada terceira anti-simétrica (5 pontos):

$$f'''(x_0) = \frac{-f_{-2} + 2f_{-1} - 2f_1 + f_2}{2h^3} + O(h^2)$$

sabendo que:

$$x_n \equiv x_0 + n \cdot h, \ (n = 0, \pm 1, \pm 2, ...)$$
$$f(x_n) \equiv f(x_0 + n \cdot h), \ (n = 0, \pm 1, \pm 2, ...)$$

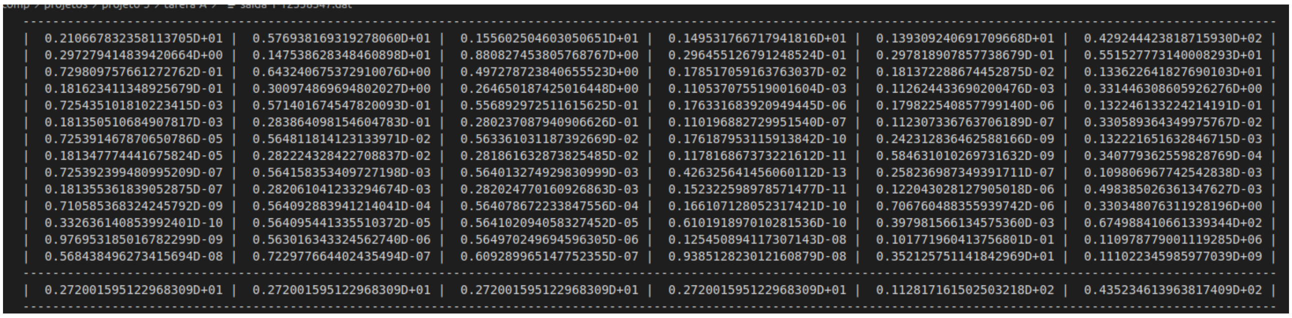
As funções $O(h^n)$ são os erros associados a cada método em função do valor de h utilizado, ou seja, representa a ordem do erro associado a cada cálculo, que é h^n.

Abaixo está o algoritmo utilizado para o Cálculo das derivadas e seus erro, bem como a saída do algoritmo que segue a mesma ordem em que os métodos foram apresentados. Em cada coluna esta o erro em relação ao valor calculado analiticamente e na ultima linha o valor calculado analiticamente.

```fortran
c      program tarefa-1
       implicit real*8 (a-h,o-z)
       Parameter(x=0.5d0)
       dimension vh(14)
       Parameter(ient = 10)
       vh = (/0.5d0,0.2d0,0.1d0,0.05d0,0.01d0,0.005d0,0.001d0,
      +0.0005d0, 0.0001d0, 0.00005d0, 0.00001d0,
      +0.000001d0, 0.0000001d0, 0.00000001d0/)

       open(unit=ient,file='saida-1-12558547.dat')
       write(ient,3)
3            format(169('-'))
       do i = 1, 14

             !derivada simétrica de 3 pontos
             d_sim3 = fx(x,1d0,vh(i)) - fx(x,-1d0,vh(i))
             d_sim3 = d_sim3/(vh(i)*2d0)

             !derivada para frente de 2 pontos
             d_frente2 = fx(x,1d0,vh(i)) - fx(x,0d0,vh(i))
             d_frente2 = d_frente2/(vh(i))

             !derivada para trás de 2 pontos
             d_tras2 = fx(x,0d0,vh(i)) - fx(x, -1d0, vh(i))
             d_tras2 = d_tras2/(vh(i))

             !derivada simétrica de 5 pontos
             d_sim5 = fx(x,-2d0,vh(i)) -8d0*fx(x,-1d0,vh(i))
             d_sim5 = d_sim5 +8d0*fx(x,1d0,vh(i)) -fx(x,2d0,vh(i))
             d_sim5 = d_sim5/(vh(i)*12d0)

             !derivada segunda simétrica de 5 pontos
             d_sec_s2 = -fx(x,-2d0,vh(i)) +16d0*fx(x,-1d0,vh(i))
             d_sec_s2 = d_sec_s2 -30d0*fx(x,0d0,vh(i))
             d_sec_s2 = d_sec_s2 +16d0*fx(x,1d0,vh(i))-fx(x,2d0,vh(i))
             d_sec_s2 = d_sec_s2/((vh(i)**2d0)*12d0)

             !derivada terceira anti-simétrica de 5 pontos
             d_ter_as5 = -fx(x,-2d0,vh(i)) +2d0*fx(x,-1d0,vh(i))
             d_ter_as5 = d_ter_as5 -2d0*fx(x,1d0,vh(i)) +fx(x,+2d0,vh(i))
             d_ter_as5 = d_ter_as5/((vh(i)**3d0)*2d0)

             write(ient,4) abs(d_sim3-d1fx(x)), abs(d_frente2-d1fx(x)),
      +abs(d_tras2-d1fx(x)), abs(d_sim5-d1fx(x)), abs(d_sec_s2-d2fx(x)),
      +abs(d_ter_as5-d3fx(x))
4            format('| ', 6(d25.18,' | '))
       end do

```

```
       write(ient,5)
 5            format(169('-'))

       write(ient,6) d1fx(x), d1fx(x),
      +d1fx(x), d1fx(x), d2fx(x), d3fx(x)
 6            format('| ', 6(d25.18,' | '))

       write(ient,7)
 7            format(169('-'))
       close(ient)

       end program

       function fx(x,dn,h)
              implicit real*8 (a-h,o-z)
              arg = x + dn*h
              fx = dcosh(3d0*arg)*dsin((arg/2d0))
       end function

       function d1fx(x)
              implicit real*8 (a-h,o-z)
              d1fx = 3d0*dsin(x/2d0)*dsinh(3d0*x)
      ++(1d0/2d0)*dcos(x/2d0)*dcosh(3d0*x)
       end function

       function d2fx(x)
              implicit real*8 (a-h,o-z)
              d2fx = (35d0/4d0)*dcosh(3d0*x)*dsin(x/2d0)
      ++3d0*dcos(x/2d0)*dsinh(3d0*x)
       end function

       function d3fx(x)
              implicit real*8 (a-h,o-z)
              d3fx = (107d0/8d0)*dcos(x/2d0)*dcosh(3d0*x)
      ++ (99d0/4d0)*dsin(x/2d0)*dsinh(3d0*x)
       end function
```

Algoritmo 1: Código para resolução da tarefa A



0.21066783235811370 5D+01	0.57693816931927806 0D+01	0.15560250460305065 1D+01	0.14953176671794181 6D+01	0.13930924069170966 8D+01	0.42924442381871593 0D+02
0.29727941483942066 4D+00	0.14753862834846089 8D+01	0.88082745380576876 7D+00	0.29645512679124852 4D-01	0.29781890785773867 9D-01	0.55152777314000829 3D+01
0.72980975766127276 2D-01	0.64324067537291007 6D+00	0.49727872384065552 3D+00	0.17851705916376303 7D-02	0.18137228867445287 5D-02	0.13362264182769010 3D+01
0.18162341134892567 9D-01	0.30097486969480202 7D+00	0.26465018742501644 8D+00	0.11053707551900160 4D-03	0.11262443369020047 6D-03	0.33144630860592627 6D+00
0.72543510181022341 5D-03	0.57140167454782009 3D-01	0.55689297251161562 5D-01	0.17633168392094944 5D-06	0.17982254085779914 0D-06	0.13224613322421419 1D-01
0.18135051068490781 7D-03	0.28386409815460478 3D-01	0.28023708794090662 6D-01	0.11019688272995154 0D-07	0.11230733676370618 9D-07	0.33058936434997576 7D-02
0.72539146787065078 6D-05	0.56481181412313397 1D-02	0.56336103118739266 9D-02	0.17618795311591384 2D-10	0.24231283646258816 6D-09	0.13222165163284671 5D-03
0.18134777444167582 4D-05	0.28222432842270883 7D-02	0.28186163287382548 5D-02	0.11781686737322161 2D-11	0.58463101026973163 2D-09	0.34077936255982876 9D-04
0.72539239948099520 9D-07	0.56415835340972719 8D-03	0.56401327492983099 9D-03	0.42632564145606011 2D-13	0.25823698734939171 1D-07	0.10980696774254283 8D-03
0.18135536183905287 5D-07	0.28206104123329467 4D-03	0.28202477016092686 3D-03	0.15232259897857147 7D-11	0.12204302812790501 8D-06	0.49838502636134762 7D-03
0.71058536832424579 2D-09	0.56409288394121404 1D-04	0.56407867223384755 6D-04	0.16610712805231742 1D-10	0.70676048835593974 2D-06	0.33034807631192819 6D+00
0.33263614085399240 1D-10	0.56409544133551037 2D-05	0.56410209405832745 2D-05	0.61019189701028153 6D-10	0.39798156613457536 0D-03	0.67498841066133934 4D+02
0.97695318501678229 9D-09	0.56301634332456274 0D-06	0.56497024969459630 5D-06	0.12545089411730714 3D-08	0.10177196041375680 1D-01	0.11097877900111928 5D+06
0.56843849627341569 4D-08	0.72297766440243549 4D-07	0.60928996514775235 5D-07	0.93851282301216087 9D-08	0.35212575114184296 9D+01	0.11102234598597703 9D+09
0.27200159512296830 9D+01	0.27200159512296830 9D+01	0.27200159512296830 9D+01	0.27200159512296830 9D+01	0.11281716150250321 8D+02	0.43523461396381740 9D+02

Figura 1: saída do algoritmo

Analisando os resultados é possível concluir que métodos que utilizam mais pontos convergem mais rápido para o resultado e são mais precisos visto que diluem o erro associado. Por este motivo, o método mais estável e preciso é o da derivada simétrica de 5 pontos. Além disso, podemos observar que todos os métodos tem um h ideal e quando passa desse ponto começam a perder pressão, isso acontece pois para h muito pequenos o computador perde a capacidade de calcular o erro atrelado ao cálculo. executado.

Tarefa B

A tarefa B nos pede para calcular numericamente a integral definida no intervalo $[0, 1]$ com uma precisão de 10^{-11} (precisão dupla) da seguinte função:

$$f(x) = e^{x/4} \cdot sen(\pi x).$$

Com isso em mente, é apresentado trés métodos computacionais para o Cálculo de uma integral definida, que estão demonstrado abaixo:

Regra do Trapézio

$$\int_{-h}^{h} f(x)dx = \frac{h}{2} \cdot (f_{-1} + 2f_0 + f_1) + O(h^3)$$

Regra de Simpson

$$\int_{-h}^{h} f(x)dx = \frac{h}{3} \cdot (f_1 + 4f_0 + f_{-1}) + O(h^5)$$

Regra de Boole

$$\int_{x0}^{x_4} f(x)dx = \frac{2h}{45} \cdot (7f_0 + 32f_1 + 12f_2 + 32f_3 + 7f_4) + O(h^7)$$

A seguir está o algoritmo utilizado para o cálculo das integrais e seus erro, bem como a saída do algoritmo que segue a mesma ordem em que os métodos foram apresentados. Nas duas primeiras colunas está o valor n e h de cada interação, nas demais colunas estão o erro calculado em relação ao valor obtido analiticamente (0.72245288409287811).

```
c       program tarefa-2
        implicit real*8 (a-h,o-z)
```

```
      implicit integer*8 (i-n)
      dimension ivn(10)
      Parameter(ient = 10)
      Parameter(a = 0d0)
      Parameter(b = 1d0)
      Parameter(in = 10d0)
      ivn = (/12,24,48,96,192,384,768,
     +1536,3072,6144/)

      open(unit=ient,file='saida-2-12558547.dat')
      do i = 1, in
        h = (b-a)/ivn(i)

        !regra do trapezio
        rtrap = 0d0
        do j = 1, (ivn(i)/2)
            xj = a + 2*j*h
            rtrap = +rtrap +fx(xj,-2d0,h) +2*fx(xj,-1d0,h) +fx(xj,0d0,h)
        end do
        rtrap = rtrap*(h/2)

        !regra de simpson
        rsim = 0d0
        do j = 1, (ivn(i)/2)
            xj = a + 2*j*h
            rsim = +rsim +fx(xj,-2d0,h) +4*fx(xj,-1d0,h) +fx(xj,0d0,h)

        end do
        rsim = rsim*(h/3)

        !regra de boole
        rboole = 0d0
        do j = 1, (ivn(i)/4)
                xj = 4*j*h
                rboole = + rboole + 7*fx(xj,-4d0,h) + 32*fx(xj,-3d0,h)!
     ++ 12*fx(xj,-2d0,h)+32*fx(xj,-1d0,h)+7*fx(xj,0d0,h)
        end do
        rboole = rboole*(2*h/45)

        rana = fana(a,b)
        write(ient,*) ivn(i), h ,abs(rtrap-rana),abs(rsim-rana),
     +  abs(rboole-rana)
      end do

      close(ient)

      end program

```

```
      function fx(x,dn,h)
       implicit real*8 (a-h,o-z)
       Parameter(rpi=dacos(-1.d0))
       arg = x + dn*h
       fx = dexp((arg/4d0))*dsin(rpi*arg)
      end function

      function fana(a,b)
        implicit real*8 (a-h,o-z)
        Parameter(rpi=dacos(-1.d0))
        fana = -((4d0*dexp(b/4d0)*(4d0*rpi*dcos(rpi*b)-dsin(rpi*b)))
     +   /(1d0+16d0*(rpi**2)))
        fana = fana + ((4d0*dexp(a/4d0)*(4d0*rpi*dcos(rpi*a)
     +   -dsin(rpi*a)))/(1d0+16d0*(rpi**2)))
       end function
```

Algoritmo 2: Código para resolução da tarefa B

12	8.3333333333333329E-002	4.1571359729467572E-003	1.8759302373161368E-005	4.8003127084417230E-007
24	4.1666666666666664E-002	1.0384097928567426E-003	1.1656005066695840E-006	7.3129510225200534E-009
48	2.0833333333333332E-002	2.5954789053006522E-004	7.2743578383160923E-008	1.1355028028958714E-010
96	1.0416666666666666E-002	6.4883564023787699E-005	4.5448125263192196E-009	1.7711387911845122E-012
192	5.2083333333333330E-003	1.6220677986455989E-005	2.8402535878768731E-010	2.7200464103316335E-014
384	2.6041666666666665E-003	4.0551561836243977E-006	1.7749801628497153E-011	2.2204460492503131E-015
768	1.3020833333333333E-003	1.0137882144878319E-006	1.1088907569956064E-012	7.7715611723760958E-016
1536	6.5104166666666663E-004	2.5344700327334380E-007	6.8056671409522096E-014	1.2212453270876722E-015
3072	3.2552083333333332E-004	6.3361746405199426E-008	6.7723604502134549E-015	1.3322676295501878E-015
6144	1.6276041666666666E-004	1.5840432854297148E-008	2.8865798640254070E-015	1.7763568394002505E-015

Figura 2: saída do algoritmo

É possível concluir, analisando os resultados, que o método de Boole é o mais preciso e o que converge mais rapidamente. Além disso, assim como anteriormente para valores de h muito pequenos perde se a precisão do erro, apesar de se ter uma variação menor no erro.

Tarefa C

Na tarefa C é pedido para calcular numericamente as 3 raízes da função $f(x)$ escrita a seguir, com um erro de 10^{-6}.

$$f(x) = x^3 - \frac{3}{2}x^2 - \frac{3}{2}x + 1$$

Para isso, é apresentado três métodos computacionais diferentes o método da busca direta, o método de Newton-Raphson e o método da secante, que serão apresentados a seguir.

Método da busca direta

O método da busca direta é o modo que utiliza a força bruta para encontrar as raízes, a partir de um x_0 é calculado o resultado da função para todos os valores de x, em que $x = x_0 + passo$.

Para cada iteração é verificado se houve uma mudança de sinal, caso haja uma mudança significa que a raiz esta dentro desse intervalo limitado pelo x anterior e o atual. Em seguida, é calculado o ponto central do intervalo em que a raiz está e em seguida verificado se a raiz está entre o ponto central e o ponto a direita ou o ponto a esquerda e o central. Assim, um novo intervalo é definido e se repete o processo até que a diferença entre o ultimo e penúltimo ponto central seja menor que o erro arbitrário definido (10^{-6}).

Newton-Raphson

$$x^{i+1} = x^i - \frac{f(x^i)}{f'(x^i)}$$

nesse caso o algoritmo também continua até que a diferença entre o ultimo e penúltimo ponto seja menor que o erro arbitrário definido (10^{-6}).
Secante

$$x^{i+1} = x^i - f(x^i)\frac{x^i - x^{i-1}}{f(x^i) - f(x^{i-1})}$$

assim como anteriormente o algoritmo também continua até que a diferença entre o ultimo e penúltimo ponto seja menor que o erro arbitrário definido (10^{-6}).

A seguir está o algoritmo utilizado para o cálculo das raízes, bem como a saída do algoritmo que segue a mesma ordem em que os métodos foram apresentados. Na primeira coluna está o número de iterações necessário para chegar naquele valor e na segunda o valor encontrado. O resultado final é o valor da raiz encontrado que é menor que o erro definido. Os valores exatos das raízes que devem ser encontradas são: $-1, 0.5, 2$.

```
c     program tarefa-2s
      implicit real*8 (a-h,o-z)
      Parameter(erro = 10d-6)
      Parameter(a = -10d0)
      Parameter(passo = 0.1d0)
      Parameter(nraizes = 3)
      Parameter(ient = 10)
      Logical :: flag

      open(unit=ient,file='saida-3-12558547.dat')

      !Método de busca direta
      write(ient,*) "--------------------------------------------------"
      write(ient,*) "Método de busca direta"

      x1 = a

      do i = 1,nraizes
        iter = 0
        flag = .true.
        x2 = x1 + passo
        do while(flag .or. (abs(fx(x1)-fx(x2))>erro))
            if(fx(x1)*fx(x2)<0) then
                flag = .false.
```

```
                centro = (x1+x2)/2d0

                if(fx(x1)*fx(centro)<0) then
                  x2 = centro
                else
                  x1 = centro
                end if
                write(ient,*) iter, centro
            else
              x1 = x2
              x2 = x2 + passo
            end if
            iter = iter +1
        end do
        write(ient,*) "Resultado final: ",iter, centro
        x1 = x2
      end do

      !Método de Newton Raphson
      write(ient,*) "----------------------------------------------------"
      write(ient,*) "Método de Newton Raphson"

      x1 = a

      do i = 1,nraizes
        iter = 0
        flag = .true.
        x2 = x1 + passo
        do while(flag)
          if(fx(x1)*fx(x2)<0) then
            x_new = (x1+x2)/2d0
            do while(flag)
              iter = iter +1
              x_old = x_new
              x_new = x_old - (fx(x_old))/(dfx(x_old))

              write(ient,*) iter,x_new
              if(abs(x_new-x_old)<=erro .or. abs(fx(x_new))<=erro)then
                flag = .false.
              end if
            end do
          else
            x1 = x2
            x2 = x2 + passo
          end if
          iter = iter +1
        end do
        write(ient,*) "Resultado final: ",iter-1, x_new
```

```
    x1 = x2
  end do

  !Método de Newton Raphson
  write(ient,*) "---------------------------------------------------"
  write(ient,*) "Método de Newton Raphson"

  x1 = a

  do i = 1,nraizes
    iter = 0
    flag = .true.
    x2 = x1 + passo
    x_2 = 0d0
    do while(flag)
      if(fx(x1)*fx(x2)<0) then
        do while(flag)
          x_1 = (x1+x2)/2d0
          x_0 = x1
          iter = iter +1

          x_2 = x_1 -fx(x_1)*((x_1-x_0)/(fx(x_1)-fx(x_0)))

          write(ient,*) iter,x_2
          if(abs(x_2-x_1)<=erro .or. abs(fx(x_2))<=erro)then
            flag = .false.
          end if
          x_0 = x_1
          x_1 = x_2
        end do
      else
        x1 = x2
        x2 = x2 + passo
      end if
      iter = iter +1
    end do
    write(ient,*) "Resultado final: ",iter-1, x_2
    x1 = x2
  end do

  close(ient)
  end

  function fx(x)
    implicit real*8 (a-h,o-z)
    fx =(x**3d0) - (3d0/2d0)*(x**2) - (3d0/2d0)*(x) + 1
   end function
```

```fortran
       function dfx(x)
        implicit real*8 (a-h,o-z)
        dfx = 3d0*(x**2d0) - 3d0*(x) - (3d0/2d0)
       end function
```

Algoritmo 3: Código para resolução da tarefa C

```
----------------------------------------------------
Método de busca direta
             90 -0.95000000000001861
             91 -0.97500000000001863
             92 -0.98750000000001870
             93 -0.99375000000001867
             94 -0.99687500000001861
             95 -0.99843750000001863
             96 -0.99921875000001870
             97 -0.99960937500001867
             98 -0.99980468750001861
             99 -0.99990234375001863
            100 -0.99995117187501870
            101 -0.99997558593751867
            102 -0.99998779296876861
            103 -0.99999389648439363
            104 -0.99999694824220620
            105 -0.99999847412111242
Resultado final:          106 -0.99999847412111242
             14  0.45000152587888748
             15  0.47500152587888744
             16  0.48750152587888745
             17  0.49375152587888749
             18  0.49687652587888748
             19  0.49843902587888744
             20  0.49922027587888745
             21  0.49961090087888749
             22  0.49980621337888748
             23  0.49990386962888744
             24  0.49995269775388745
             25  0.49997711181638749
             26  0.49998931884763748
             27  0.49999542236326244
             28  0.49999847412107495
Resultado final:           29  0.49999847412107495
```

Figura 3: saída do algoritmo 1ª parte

```
        14   1.9500015258788883
        15   1.9750015258788882
        16   1.9875015258788882
        17   1.9937515258788880
        18   1.9968765258788881
        19   1.9984390258788882
        20   1.9992202758788882
        21   1.9996109008788880
        22   1.9998062133788881
        23   1.9999038696288882
        24   1.9999526977538882
        25   1.9999771118163880
        26   1.9999893188476381
        27   1.9999954223632632
        28   1.9999984741210757
        29   1.9999999999999818
Resultado final:          30   1.9999999999999818
--------------------------------------------------
Método de Newton Raphson
        91  -1.0027110289587162
        92  -1.0000073188147569
        93  -1.0000000000535645
Resultado final:          93  -1.0000000000535645
        15  0.49988851727982181
        16  0.50000000000123168
Resultado final:          16  0.50000000000123168
        15   2.0023197175995953
        16   2.0000053617428746
        17   2.0000000000287481
Resultado final:          17   2.0000000000287481
--------------------------------------------------
Método de Newton Raphson
        91 -0.99999999999999900
Resultado final:          91 -0.99999999999999900
        15  0.50000000000000000
Resultado final:          15  0.50000000000000000
        15   1.9999999999999991
Resultado final:          15   1.9999999999999991
```

Figura 4: saída do algoritmo 2ª parte

Analisando os resultados é possível concluir que o método da secante converge muito mais rapidamente que o método da busca direta e um pouco mais rápido que o método de

Newton-Raphson. Isso se deve, pois como já explicado o método da busca direta funciona pela força bruta enquanto que o método da secante é muito similar ao de Newton-Raphson.