Introdução à Física Computacional: Projeto 2

Instituto de Física de São Carlos, Universidade de São Paulo

Solano E. S. Felício, n° USP 10288907

02 de abril de 2018

Introdução

Este é um relatório entregue para avaliação da disciplina 7600017 – Introdução à Física Computacional. Cada seção é uma execução direta das tarefas do projeto 2, que introduz alguns conceitos básicos do cálculo numérico.

Os problemas foram resolvidos durante as aulas 3 e 4 do curso. Aqui apresento os programas (em Fortran 90) e resultados (em forma de tabelas ou gráficos). Os gráficos foram feitos em Python com as bibliotecas NumPy e matplotlib.

1 Derivada numérica

O código é auto-explicativo. As funções `f`, `f1` e `f2` calculam $f(x) = e^{2x}\cos(x/4)$, $f'(x)$ e $f''(x)$, respectivamente, em valor exato. A cada iteração, o programa lê uma precisão `h` e retorna `h` e os erros associados a cada estimativa de derivada.

```
program derivadas
implicit none

real*8 x,h,f,f1,f2,ff1,ft1,f3s1,f3s2
real*8 fx, fx1, fx2
real*8 eff1, eft1, ef3s1, ef3s2

x = 1.d0

do
    read(*,*,end=10) h

    ! função e derivadas exatas
    fx = f(x)
    fx1 = f1(x)
    fx2 = f2(x)

    ! calcular derivada para frente de 2 pontos
    ff1 = (f(x+h)-fx)/h
    eff1 = abs(ff1-fx1)

    ! derivada para trás de 2 pontos
    ft1 = (fx-f(x-h))/h
    eft1 = abs(ft1-fx1)

    ! derivada simétrica de 3 pontos
    f3s1 = (f(x+h)-f(x-h))/(2.d0*h)
    ef3s1 = abs(f3s1-fx1)

    ! derivada segunda simétrica de 3 pontos
    f3s2 = (f(x+h) - 2.d0*fx + f(x-h))/(h**2)
    ef3s2 = abs(f3s2 - fx2)
```

```
        write(*,*) h, eff1, eft1, ef3s1, ef3s2
    end do

10  end program derivadas

    real*8 function f(x)
        real*8 x
        f = exp(2.d0*x)*cos(x/4.d0)
        return
    end

    real*8 function f1(x)
        real*8 x
        f1 = 2.d0*exp(2.d0*x)*cos(x/4.d0) &
         - (exp(2.d0*x)*sin(x/4.0d0))/4.d0
        return
    end

    real*8 function f2(x)
        real*8 x
        f2 = 4.d0*exp(2.d0*x)*cos(x/4.d0) &
         - exp(2.d0*x)*sin(x/4.d0)/2.d0   &
         - exp(2.d0*x)*sin(x/4.0d0)/2.d0 &
         - exp(2.d0*x)*cos(x/4.d0)/16.d0
        return
    end
```

Entrando com os valores de `h` especificados no texto da tarefa, o programa retorna a tabela de valores seguinte (dígitos truncados).

h	f'_f	f'_t	f'_{3s}	f''_{3s}
5.000000e-01	9.199118e+00	4.937125e+00	2.130996e+00	1.910634e+00
1.000000e-01	1.403846e+00	1.239801e+00	8.202247e-02	7.461014e-02
5.000000e-02	6.799931e-01	6.390314e-01	2.048088e-02	1.863850e-02
1.000000e-02	1.326319e-01	1.309941e-01	8.189190e-04	7.453606e-04
5.000000e-03	6.610982e-02	6.570037e-02	2.047273e-04	1.863387e-04
1.000000e-03	1.318912e-02	1.317274e-02	8.189058e-06	7.452267e-06
5.000000e-04	6.592511e-03	6.588416e-03	2.047263e-06	1.862961e-06
1.000000e-04	1.318174e-03	1.318011e-03	8.189361e-08	3.331296e-08
5.000000e-05	6.590668e-04	6.590258e-04	2.048940e-08	4.774022e-07
1.000000e-05	1.318101e-04	1.318085e-04	7.895959e-10	1.184609e-05
5.000000e-06	6.590505e-05	6.590418e-05	4.343246e-10	5.917482e-06
1.000000e-06	1.317958e-05	1.318067e-05	5.426699e-10	1.604639e-03
1.000000e-07	1.325952e-06	1.329702e-06	1.874938e-09	1.946828e-01
1.000000e-08	1.180291e-07	5.960653e-08	2.921131e-08	8.598284e+00

Tabela 1: erros cometidos nas aproximações para as derivadas, de acordo com o valor de h. f'_f – derivada para frente, f'_t – derivada para trás, f'_{3s} – derivada simétrica de três pontos, f''_{3s} – derivada segunda simétrica de 3 pontos.

Note, na tabela 1, que dos valores especificados de h, nem sempre o menor teve resultado ótimo: no caso da derivada simétrica de 3 pontos, o melhor resultado foi obtido com $h = 5 \times 10^{-6}$, enquanto no da derivada segunda simétrica, o valor ótimo foi $h = 1 \times 10^{-4}$. Isto ocorre porque, quando se diminui muito o valor de h em comparação com $f(x)$, os valores $f(x-h)$, $f(x)$ e $f(x+h)$ são próximos e grandes, mas armazenados no computador com precisão finita, de modo que as diferenças entre eles não podem ser calculadas com precisão.

Antes de chegar ao valor ótimo, é esperado que o erro das quatro aproximações diminua com $\mathcal{O}(h)$, $\mathcal{O}(h)$, $\mathcal{O}(h^2)$ e $\mathcal{O}(h^2)$, respectivamente. Fazendo uma regressão linear em cada gráfico dilog na figura 1,

apenas para h maior ou igual ao valor ótimo em cada caso, obtemos os coeficientes angulares dados na tabela 2, confirmando o que se esperava.

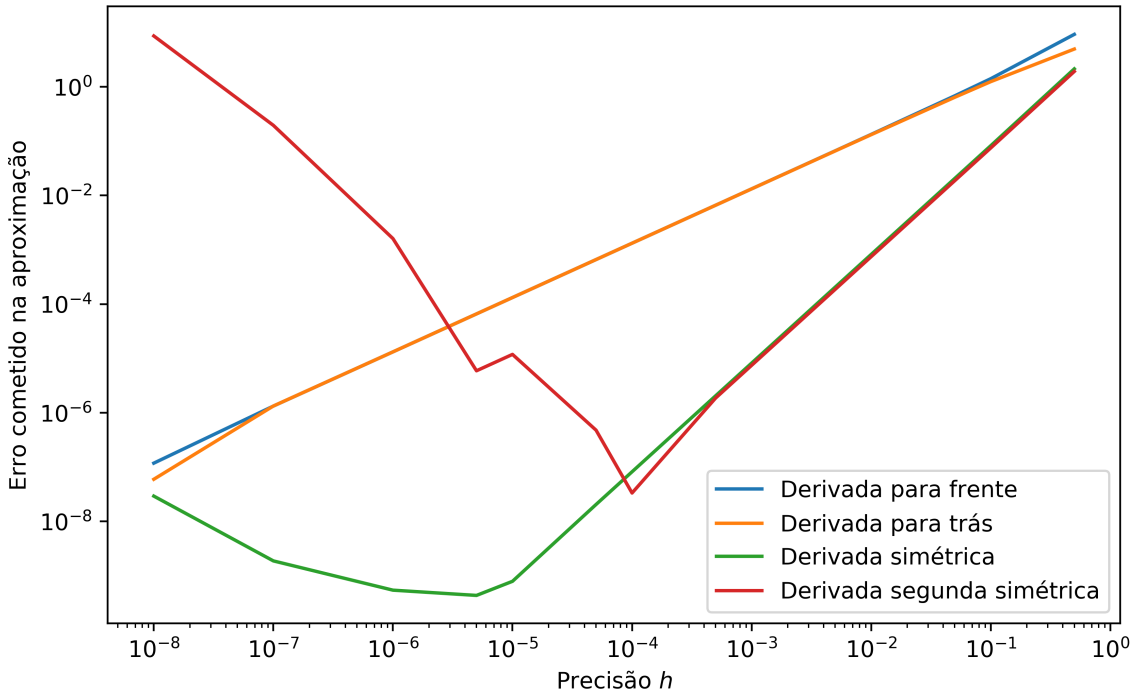

Figura 1: erros associados a cada aproximação, conforme o valor de h.

Aproximação	coeficiente na regressão	ordem esperada
Derivada para frente	1,011 ± 0,004	$\mathcal{O}(h^1)$
Derivada para trás	1,01 ± 0,01	$\mathcal{O}(h^1)$
Derivada simétrica	2,004 ± 0,001	$\mathcal{O}(h^2)$
Derivada segunda simétrica	2,003 ± 0,001	$\mathcal{O}(h^2)$

Tabela 2: Coeficientes angulares das regressões feitas nos gráficos dilog do erro *versus* precisão.

2 Integração numérica

Desta vez, as funções `f` e `primitiva` calculam $f(x) = e^{2x}\sin x$ e uma primitiva de f, respectivamente, em valores exatos. A variável `intexato` armazena a integral que nos interessa, calculada pelo Teorema Fundamental do Cálculo, enquanto as variáveis `intt` e `ints` armazenam as aproximações pelo método do trapézio e pelo método de Simpson, respectivamente.

A cada iteração, o programa lê um valor para `n`, e divide o intervalo $[0, 1]$ em `n` subintervalos, sendo `h` o tamanho de cada um. Como sugerido, tanto no método de Simpson como no do trapézio, calcula-se o valor da função nos extremos separadamente. O módulo da diferença entre `intt` (`ints`) e `intexato` dá o erro do método do trapézio (de Simpson) no valor de `n` usado. O programa retorna `h` e os dois erros em cada iteração.

```
program integrais
implicit none

real*8 a,b,h,x,f,primitiva
real*8 intt,errot, ints,erros, intexato
integer i,n
```

```
        parameter(a=0.d0)
        parameter(b=1.d0)

        intexato = primitiva(b)-primitiva(a)

        do
            read(*,*,end=10) n
            h = (b-a)/n

            ! Método do trapézio
            intt = (f(a)+f(b))/2.d0
            do i=1,n-1
                intt = intt+f(a+real(i,8)*h)
            end do
            intt = intt*h
            errot = abs(intt-intexato)

            ! Método de Simpson
            ints = f(a)+f(b)
            do i=1,n-1
                if (mod(i,2)==1) then
                    ints = ints + f(a+real(i,8)*h)*4.d0
                else
                    ints = ints + f(a+real(i,8)*h)*2.d0
                end if
            end do
            ints = ints*h/3.d0
            erros = abs(ints-intexato)

            write(*,*) h, errot, erros
        end do

10      end program integrais

        real*8 function f(x)
            real*8 x
            f = exp(2.d0*x)*sin(x)
            return
        end

        real*8 function primitiva(x)
            real*8 x
            primitiva = -exp(2.d0*x)*(cos(x)-2.d0*sin(x))/5.d0
            return
        end
```

Entrando com os valores de `n` sugeridos (primeira à 13ª potências de 2), o programa dá como saída a tabela 3, da qual se produzem os gráficos da figura 2.

O método de Simpson, como esperado, produziu resultados melhores, mas de novo a precisão não aumentou indefinidamente: $h = 1/4096 \approx 2{,}4\times10^{-4}$ foi o valor ótimo.

Fazendo uma regressão linear nos gráficos dilog (apenas até o valor ótimo), obtemos os coeficientes angulares $1{,}9993 \pm 0{,}0003$, para o método do trapézio, e $4{,}005 \pm 0{,}001$, para o método de Simpson. Isto confirma o esperado: os erros globais são $\mathcal{O}(h^2)$ e $\mathcal{O}(h^4)$, respectivamente.

h	Método do trapézio	Método de Simpson
5.000000e-01	3.174202e-01	1.648282e-02
2.500000e-01	8.010554e-02	1.000649e-03
1.250000e-01	2.007273e-02	6.179023e-05
6.250000e-02	5.021069e-03	3.848893e-06
3.125000e-02	1.255447e-03	2.403477e-07
1.562500e-02	3.138731e-04	1.501846e-08
7.812500e-03	7.846899e-05	9.386023e-10
3.906250e-03	1.961729e-05	5.866130e-11
1.953125e-03	4.904325e-06	3.667067e-12
9.765625e-04	1.226082e-06	2.302603e-13
4.882812e-04	3.065204e-07	1.398881e-14
2.441406e-04	7.663010e-08	1.554312e-15
1.220703e-04	1.915752e-08	5.995204e-15

Tabela 3: Saída do programa. Dígitos truncados.

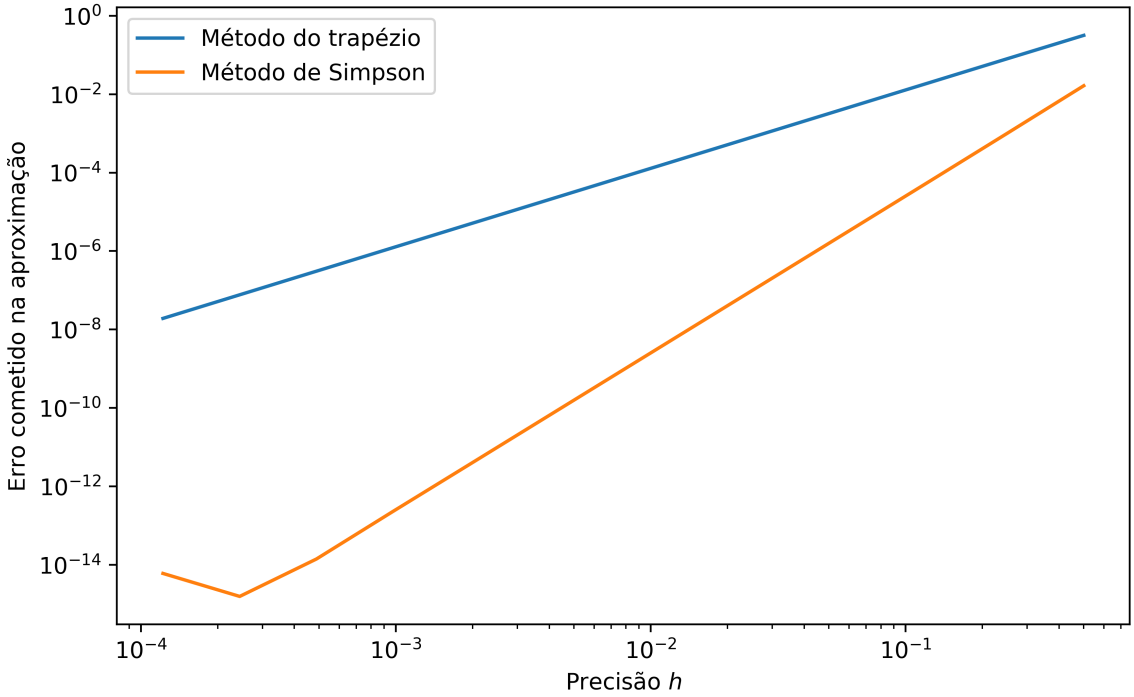

Figura 2: Erros cometidos nas aproximações das integrais, conforme o valor de h.

3 Equações algébricas não lineares

Para a parte (a), dividimos o intervalo $[-1, 1]$ em 50 partes e calculamos os valores da função para o gráfico. O programa em Fortran calcula os dados, que são lidos e plotados em Python. Nas raízes da equação, o gráfico deve cruzar o eixo x.

```
program ising
implicit none

real*8 x
integer i

x = -1
do i=1,51
    write(*,*) x, tanh(x/0.8)-x, tanh(x/1.8)-x
```

```
    x = x+0.04
end do

end program ising
```

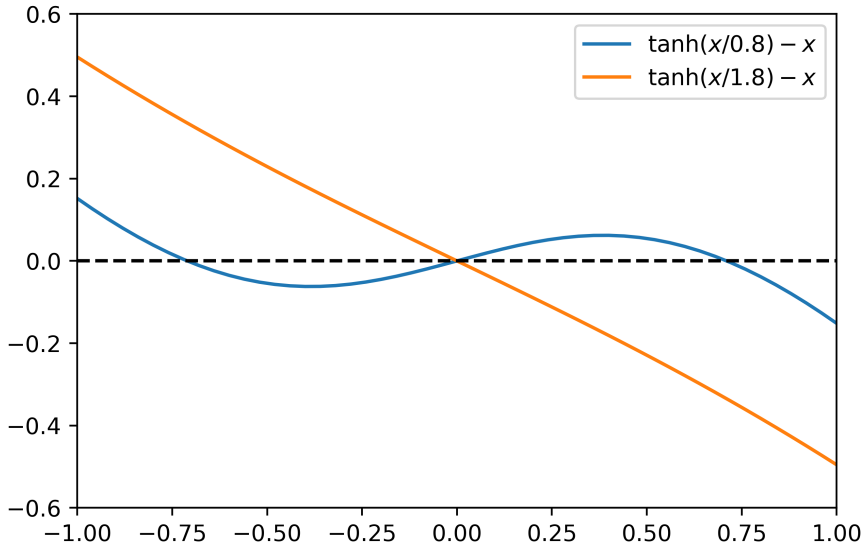

Figura 3: Raízes da equação do modelo de Ising. Para $T > 1$, há apenas uma solução ($x = 0$), enquanto para $0 < T < 1$, há três.

No programa seguinte, escrito para a parte (b), `a` e `b` são x_- e x_+, respectivamente. O algoritmo é o método da bisseção: em cada iteração, tomamos `m` como a média de `a` e `b` e calculamos a função $f(x) = x - \tanh(x/T)$ em $x =$ `m`. Se o valor for positivo, fazemos `b = m`; se for negativo, `a=m`; se for nulo, chegamos na raiz exata e então fazemos `a = m` e `b = m`. Calculamos o erro, igual a metade da diferença entre `a` e `b`, e se este for menor que ε e não tivermos ultrapassado o limite de iterações `imax`, repetimos o processo. `a` e `b` são inicializados com $\varepsilon > 0$ e $1 - \varepsilon$, o que garante que de início a função calculada em `a` e em `b` tem sinais opostos. A cada iteração, o programa retorna o número da iteração atual, a estimativa atual da raiz, e o erro associado à estimativa. O programa recebe como entrada a temperatura $0 < T < 1$ e a precisão desejada ε.

```
program bissecao
implicit none

real*8 a,b,m,erro,f
real*8 eps, T
integer i, imax

parameter(imax = 200)

read(*,*) T, eps

i=0
a = eps
b = 1-eps
erro = 10
do while (erro >= eps .and. i <= imax)
    m = (a+b)/2.d0
    f = m - tanh(m/T)
    if (f>0) then
        b = m
```

```
        else if (f < 0) then
            a = m
        else
            a = m
            b = m
        end if

        erro = (b-a)/2.d0
        i = i+1

        write(*,*) i, m, erro
    end do

    end program bissecao
```

Nota: se a precisão estipulada for menor que 10^{-16}, é comum que o programa atinja a raiz "exata" (até onde vai a precisão de 8 bytes) em torno da 50ª iteração. Nesse caso, o erro calculado é 0, mas a incerteza deve ser considerada da ordem da última casa decimal. Assim, uma escolha razoável para ε é 10^{-15}, embora isto não garanta que o erro retornado não seja 0.

O critério de convergência usado garante que, se o erro estimado pelo algoritmo (δ) não for exatamente 0, é uma quota superior para o erro real, de modo que, se m é a raiz estimada, o valor real da raiz está sempre no intervalo $(m - \delta, m + \delta)$.

Para a parte (c), basta uma pequena modificação do programa acima. Fixamos $\varepsilon = 10^{-15}$, dividimos o intervalo $[0, 1] \ni T$ em 100 partes e calculamos a raiz m para cada T. O resultado é apresentado na figura 4.

```
    program raizt
    implicit none

    real*8 a,b,m,erro,f
    real*8 eps, T
    integer i,imax,n

    parameter (eps=1.d-15)
    parameter (imax=100)
    parameter (n=100)

    do i=1,n
        T = dfloat(i)/dfloat(n)
        a = eps
        b = 1-eps
        erro = 10
        do while (erro >= eps .and. i <= imax)
            m = (a+b)/2.d0
            f = m - tanh(m/T)
            if (f>0) then
                b = m
            else if (f < 0) then
                a = m
            else
                a = m
                b = m
            end if

            erro = (b-a)/2.d0
        end do

        write(*,*) T, m
```

```
        end do

        end program raizt
```

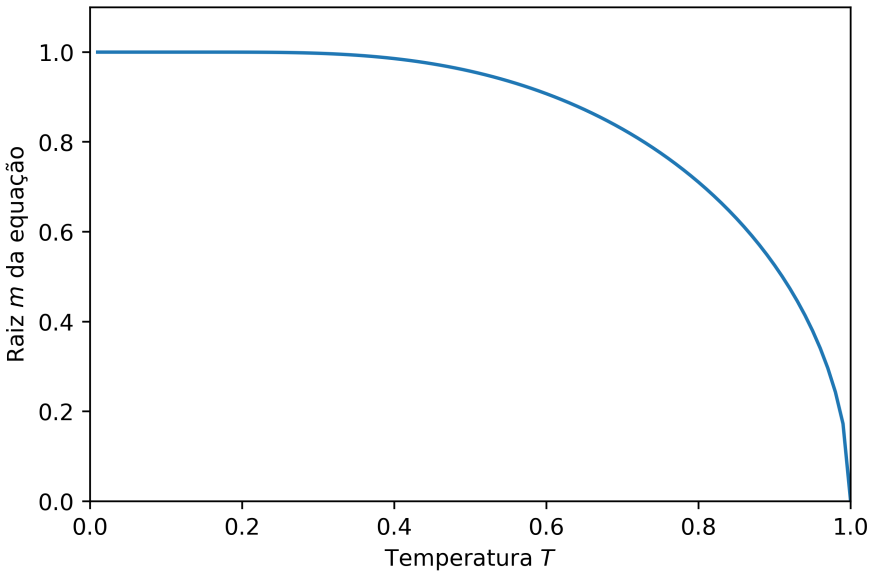

Figura 4: Raiz positiva da equação do modelo de Ising, como função do parâmetro T.

Note que, conforme T se aproxima de 1, a raiz $m > 0$ da função $f(x) = x - \tanh(x/T)$ se aproxima de 0. Isto está de acordo com o esperado, pois a aproximação cúbica de $\tanh(x/T)$ em torno de $x = 0$ é $(x/T) - 2(x/T)^3$, de modo que, para $|x| \ll 1$,

$$f(x) \approx \left(\frac{T-1}{T}\right) x + \left(\frac{2}{T^3}\right) x^3.$$

Este polinômio tem raiz positiva

$$\sqrt{\frac{(1-T)T^2}{2}}$$

que tende a 0 quando $T \to 1$.