Tarefa 2

Kaio Henrique de Sousa

24 de março de 2020

Questão 1

Pelo **Teorema de Bolzano** se $f(x)$ é contínua e $f(a) < 0$ e $f(b) > 0$, ou ainda, $f(a) > 0$ e $f(b) < 0$, então existe ao menos uma raiz no intervalo $[a, b]$. Como a função $f(x) = x^3 - 1.7x^2 - 12.78x - 10.08$ é contínua, podemos utilizar o teorema para encontrar raízes da função. Para isso, iremos usar alguns métodos numéricos para encontrar as raízes da função.

Primeiramente foi definido como critério de parada o seguinte:

$$|x_{k+1} - x_k| < e \tag{1}$$

Onde e é o erro de aproximação e x_{k+1} e x_k são pontos com sinais diferentes.

Bisseção

Neste método foi definido uma função que recebe dois pontos a, b e tol que será o erro de aproximação e os pontos têm sinais diferentes. Enquanto a (1) não for satisfeita, calculamos um ponto médio p no intervalo $[a, b]$, se a distância média entre a e b é menor que a tolerância então foi encontrado uma aproximação da raiz, senão repetimos o procedimento com $a = p$ caso $a * p > 0$ ou $b = p$ caso $b * p > 0$. O método é implementado no Apêndice A.

Ponto Fixo

Para este método é necessário definir uma função $g(x)$ tal que $f(x) = x - g(x)$ e para algum ponto $p, g(p) = p$. Para encontrar a raiz de $f(x)$ basta encontrar um ponto fixo em g. No Apêndice B é implementado uma função na linguagem de programação C que recebe um ponto inicial *initPT* e a tolerância *tol*. Enquanto a condição (1) não for satisfeita, define-se um ponto

$p = g(initPT)$. Se $p = g(p)$, então uma raiz foi encontrada, senão fazemos $initPT = p$, $p = g(initPT)$ e repetimos o procedimento.

Newton-Raphson

Como visto em sala que:

$$0 \approx f(p_0) + (p - p_0)f`(p_0)$$

isolando p:

$$p \approx p_0 - \frac{f(p_0)}{f`(p_0)} \tag{2}$$

Na implementação deste método no Apêndice C, a função Newton(initPT, tol) recebe o ponto inicial p_0 e a tolerância. Agora, p é dado pela equação (2) e a função entra no *Loop*, enquanto $|p - p0| \geq tol$ atualize o p. Se $|p - p0| < tol$, então p é a raiz. Senão, o valor de p é atribuído a p_0.

Desse modo, o processo iterativo é dado por:

$$x_{k+1} = x_k - \frac{f(x_k)}{f`(x_k)}$$

Secante

O método secante é similar ao método de Newton com a diferença de que a derivada de $f(x)$ é dada por:

$$f`(p_{n-1}) = \lim_{x \to p_{n-1}} \frac{f(x) - f(p_{n-1})}{x - p_{n-1}}$$

Se p_{n-2} é suficientemente próximo de p_{n-1}, então

$$f`(p_{n-1}) \approx \frac{f(p_{n-2}) - f(p_{n-1})}{p_{n-2} - p_{n-1}} = \frac{f(p_{n-1}) - f(p_{n-2})}{p_{n-1} - p_{n-2}}$$

substituindo $f`(p_{n-1})$ na formula de Newton, temos:

$$p_n = p_{n-1} - \frac{f(p_{n-1})(p_{n-1} - p_{n-2})}{f(p_{n-1}) - f(p_{n-2})} \tag{3}$$

Na implementação do Apêndice D, o processo iterativo é dado pela equação (3).

Regula Falsi

A implementação e o processo iterativo do método Regula Falsi no Apêndice E é similar ao método Secante, com a diferença de que o método em questão garante que a raiz esteja entre os pontos nas sucessivas iterações

Questão 2

Listing 1: Comparação dos Resultados

```
>>>gcc q1.c -o q1 -lm
>>>./q1

Intervalo [-2.5, -1.5]
input
>>>Digite a e b para [a, b] e o erro: -2.5 -1.5 0.00000001

output
>>>Bissecao: Apos 26 iteracoes temos, -2.100000 como raiz
>>>Fix Point: Apos 20 iteracoes temos, -2.100000 como uma aproximacao da raiz
>>>Newton: Apos 4 iteracoes temos, -2.100000 como raiz
>>>Secante: Apos 6 iteracoes temos, -2.100000 como raiz
>>>Ponto Falso: Apos 16 iteracoes temos, -2.100000 como raiz


Intervalo [-1.5, 0]
input
>>>Digite a e b para [a, b] e o erro: -1.5 0 0.00000001

output
>>>Bissecao: Apos 27 iteracoes temos, -1.000000 como raiz
>>>Fix Point: Apos 22 iteracoes temos, -2.100000 como uma aproximacao da raiz
>>>Newton: Apos 6 iteracoes temos, -1.000000 como raiz
```

```
>>>Secante: Apos 8 iteracoes temos, -1.000000 como raiz
>>>Ponto Falso: Apos 18 iteracoes temos, -1.000000 como raiz

Intervalo [0, 10]
input
>>>Digite a e b para [a, b] e o erro: 0 10 0.00000001

output
>>>Bissecao: Apos 29 iteracoes temos, 4.800000 como raiz
>>>Fix Point: Apos 23 iteracoes temos, 4.800000 como uma aproximacao da raiz
>>>Newton: Apos 5 iteracoes temos, -1.000000 como raiz
>>>Secante: Apos 6 iteracoes temos, -1.000000 como raiz
>>>Ponto Falso: Apos 63 iteracoes temos, 4.800000 como raiz
```

A raiz -1 não pode ser encontrada pelo método de Ponto Fixo, além disso, no intervalo $[0, 10]$ as raízes no método de Newton e Secante foram -1 que é o raiz mais próxima do ponto 0, mas ao escolher um ponto próximo à raiz -4.8 teremos o mesmo resultado para todos os métodos.

```
input
>>>Digite a e b para [a, b] e o erro: 3 10 0.0000001

output
>>>Bissecao: Apos 26 iteracoes temos, 4.800000 como raiz
>>>Fix Point: Apos 19 iteracoes temos, 4.800000 como uma aproximacao da raiz
>>>Newton: Apos 8 iteracoes temos, 4.800000 como raiz
>>>Secante: Apos 11 iteracoes temos, 4.800000 como raiz
>>>Ponto Falso: Apos 46 iteracoes temos, 4.800000 como raiz
```

Note que em todos os intervalos o métodos que precisaram de menos iterações para encontrar as raízes foram o de Newton e o Secante, além da vantagem de que não precisam de pontos satisfazendo as condições do **Teorema de Bolzano**.

Questão 3

Para achar a raiz da função $s(t)$ basta aplicar um dos métodos estudados acima. Desse modo, o método Secante é o melhor para esse caso.

```
>>>gcc q3.c -o q3 -lm
>>>./q3

input
>>>Digite um ponto e o erro: 0 0.00001

output
>>>Secante: Apos 6 iteracoes temos, 6.003726 como raiz
>>>g(6.003726) = -0.000000
```

Como visto acima, usando o método Secante, a raiz da função é 6.003726.

Questão 4

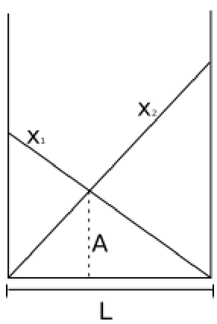

Figura 1:

Primeiramente, considere r a altura do chão até o ponto que a escada x_1 toca a parede esquerda, s a altura do chão até o ponto que a escada x_2 toca a parede direita, l_1 a parte a esquerda de A e l_2 a direita de parte a direita de A. O Teorema de pitágoras nos oferece as seguintes equações:

$$x_1^2 = r^2 + L^2 \tag{4}$$

$$x_2^2 = s^2 + L^2 \tag{5}$$

e pela semelhança de triângulos temos:

$$\begin{aligned}
&\frac{r}{L} = \frac{A}{l_2}\\
\Longrightarrow\ &\frac{r}{l_1 + l_2} = \frac{A}{l_2}\\
\Longrightarrow\ &r * l_2 = A(l_1 + l_2)\\
\Longrightarrow\ &l_2 = \frac{A(l_1 + l_2)}{r} \qquad (6)
\end{aligned}$$

e também

$$\begin{aligned}
&\frac{s}{L} = \frac{A}{l_1}\\
\Longrightarrow\ &\frac{s}{l_1 + l_2} = \frac{A}{l_1}\\
\Longrightarrow\ &s * l_1 = A(l_1 + l_2)\\
\Longrightarrow\ &l_1 = \frac{A(l_1 + l_2)}{s} \qquad (7)
\end{aligned}$$

Agora, calculamos:

$$\begin{aligned}
&L = l_1 + l_2\\
\Longrightarrow\ &L = \frac{A(l_1 + l_2)}{s} + \frac{A(l_1 + l_2)}{r} \dots, \quad \text{Pelas equações (6) e (7).}\\
\Longrightarrow\ &l_1 + l_2 = (\frac{A}{s} + \frac{A}{r})(l_1 + l_2)\\
\Longrightarrow\ &1 = \frac{A}{s} + \frac{A}{r}\\
\Longrightarrow\ &\frac{1}{A} = \frac{1}{s} + \frac{1}{r}\\
\Longrightarrow\ &\frac{1}{A} = \frac{1}{\sqrt{x_2^2 - L^2}} + \frac{1}{\sqrt{x_1^2 - L^2}} \dots, \quad \text{Pelas equações (4) e (5).}
\end{aligned}$$

Logo pela equação anterior, definimos a seguinte função:

$$f(L) = \frac{1}{\sqrt{x_2^2 - L^2}} + \frac{1}{\sqrt{x_1^2 - L^2}} - \frac{1}{A} \tag{8}$$

substituindo o valor de x_1, x_2 e A, obtemos uma função de uma variável. Para achar a raiz da função, utilizaremos o *método de Bisseção* no intervalo $[0, 20)$. Note que pela na segunda parcela da função, a mesma não possui limite e não é definida nos pontos maiores que 20. A raiz encontrada foi 16.212130, então esta é a distância L entre as paredes da figura 1.

```
input
>>>Digite um ponto e o erro: 0 20 0.00001
output
>>>Bissecao: Apos 20 iteracoes temos, 16.212130 como raiz
>>>f(16.212130) = 0.000000
```

A

Listing 2: Bisseção

```c
double bisec(double a, double b, double tol){
    double FA = f(a);
    double FP;
    double p;
    int count = 0;

    while((double)fabs(b-a) > tol){
        p = (b+a)/2;
        FP = f(p);

        if(FP==0 || (double)(b-a)/2 < tol){
            printf("Bissecao: Apos %d iteracoes temos, %lf como raiz\n", count, p);
            return p;
        }
        if(FA*FP > 0){
            a = p;
            FA = FP;
        }
        else{
            b=p;
        }

```

```
        count += 1;
    }

    printf("Bissecao: Apos %d iteracoes temos, %lf como uma aproximacao da raiz\n", count, p);
    return p;
}
```

B

Listing 3: Ponto Fixo

```
double fixPoint(double initPT, double tol){
    double p = g(initPT);
    int count = 0;

    while((double)fabs(p-initPT) >= tol){
        p = g(initPT);

        if((double)fabs(p-initPT) < tol || p == g(p)){
            printf("Fix Point: Apos %d iteracoes temos, %lf como raiz\n", count, p);

            return p;
        }

        initPT = p;
        p = g(initPT);
        count += 1;
    }

    printf("Fix Point: Apos %d iteracoes temos, %lf como uma aproximacao da raiz\n", count, p);
    return p;

}
```

C

Listing 4: Newton-Raphson

```c
double newton(double initPT, double tol){
    double p = initPT - (double)f(initPT)/hf(initPT);
    double p0 = initPT;
    int count = 0;

    while((double)fabs(p-p0) >= tol){
        p = initPT - (double)f(initPT)/hf(initPT);
        p0 = initPT;

        if((double)fabs(p-p0) < tol){
            printf("Newton: Apos %d iteracoes temos, %lf como raiz\n", count, p);
            return p;
        }

        initPT = p;
        count += 1;
    }

    printf("Newton: Apos %d iteracoes temos, %lf como uma aproximacao da raiz\n", count, p);
    return p;

}
```

D

Listing 5: Secante

```c
double secante(double p0, double p1, double tol){
    double q0 = f(p0);
    double q1 = f(p1);
    double p;
    int count = 0;

    while((double)fabs(p1-p0) >= tol){
        p = p1 - (double)q1*(p1-p0)/(q1-q0);

        if((double)fabs(p-p1) < tol){
```

```
            printf("Secante: Apos %d iteracoes temos, %lf como raiz\n", count, p);
            return p;
        }

        p0 = p1;
        p1 = p;
        q0 = q1;
        q1 = f(p);
        count += 1;
    }

    printf("Secante: Apos %d iteracoes temos, %lf como uma aproximacao da raiz\n", count, p);
    return p;

}
```

E

Listing 6: Regula Falsi

```
double regulaFalse(double p0, double p1, double tol){
        double q0 = f(p0);
        double q1 = f(p1);
        double p, q;
        int count = 0;

        while((double)fabs(p1-p0) >= tol){
                p = p1 - (double)q1*(p1-p0)/(q1-q0);

                if((double)fabs(p-p1) < tol){
                        printf("Ponto Falso: Apos %d iteracoes temos, %lf como raiz\n", count, p);
                        return p;
                }
                q = f(p);

                if(q*q1 < 0){
                        p0 = p1;
```

```
                    q0 = q1;
                }
                p1 = p;
                q1 = q;
                count += 1;
        }

        printf("Ponto Falso: Apos %d iteracoes temos, %lf como uma aproximacao da raiz\n", count, p);
        return p;
}
```

F

```
#include <stdio.h>
#include <math.h>

#define s0 300
#define g 32.17
#define m 0.25
#define k 0.1

double s(double t);
double secante(double p0, double p1, double tol);

double s(double t){
    return s0 - ((m*g)/k)*t + ((m*m*g)/(k*k)) * (1-exp(-k*t/m));
}

double secante(double p0, double p1, double tol){
    double q0 = s(p0);
    double q1 = s(p1);
    double p;
    int count = 0;

    while((double)fabs(p1-p0) >= tol){
        p = p1 - (double)q1*(p1-p0)/(q1-q0);

```

```c
        if((double)fabs(p-p1) < tol){
            printf("Secante: Apos %d iteracoes temos, %lf como raiz\n", count, p);
            return p;
        }

        p0 = p1;
        p1 = p;
        q0 = q1;
        q1 = s(p);
        count += 1;
    }

    printf("Secante: Apos %d iteracoes temos, %lf como uma aproximacao da raiz\n", count, p);
    return p;
}


int main(void){
    double a, r, tol;

    printf("Digite um ponto e o erro: ");
    scanf("%lf %lf", &a, &tol);
    r = secante(a, a+0.00001, tol);
    printf("f(%lf) = %lf\n", r , s(r));
    return 0;
}
```