Gabriel Barbosa da Silva

Professora Juliana

Métodos Numéricos

3º Lista de Exercícios

1. Construa uma função $f(x)$ que contenha ao menos uma raiz real e tal que a função $f(x)$ possa ser escrita como $f(x) = g(x) - h(x)$, onde os gráficos de $g(x)$ e $h(x)$ possam ser esboçados facilmente de modo a localizar a(s) raiz(es). Após localizar um intervalo $[a, b]$ graficamente que contenha uma raiz real, verifique quantas iterações são necessárias utilizando o método da bissecção para obter uma aproximação para a raiz contida no intervalo escolhido com erro inferior a 10^{-3}. Encontre tal aproximação.

Foi escolhida a função $f(x) = 6x^2 - cos^{-1}(x)$, antes de tudo vamos montar uma tabela de pontos com possíveis raízes.

x	-1	-0.75	-0.5	-0.25	0	0.25	0.5
f(x)	2,8584	0,9561	-0,5944	-1,4485	-1,5708	-0,9431	0,4528

Agora podemos observar dois pontos interessantes nos intervalos $[-\ 0.75,\ \ -\ 0.5]$e $[0.25,\ 0.5]$. Para aplicar o método da bissecção, foi feito um algoritmo em java que recebe como parâmetros o início e fim do intervalo em que se quer achar a raíz.

```java
//MetodoBisseccao.java
public class MetodoBisseccao{

    public static double Fx(double x){
        double y = (6*(Math.pow(x,2.0) ))-(Math.acos(x));
        return y;
    }

    public static void main(String args[]){
        int k = 0;

        double a = Double.parseDouble(args[0]); //Inicio do intervalo pego por parametro
        double b = Double.parseDouble(args[1]);  //Fim do intervalo pego por parametro
        double e = 0.001; //Precissão
        double Xk = (a+b)/2;

        System.out.println("k   |       a       |       b       |       Xk      |       F(Xk)");
        while( Math.abs(Fx(Xk)) >= e ){
            Xk = (a+b)/2;
            if(Fx(a)*Fx(Xk)<0){
                b = Xk;
            }else{
                a = Xk;
            }
            System.out.println(k+"  |       "+a+"       |       "+b+"       |       "+Xk+"      |       "+Fx(Xk));
            k++;
        }

    }
}
```

Imagem 1 - Algoritmo em Java™ para método da bissecção.

Para o primeiro intervalo, foi realizado 9 iterações:

```
barbosa@Lunella:~/Faculdade/5ºsemestre/Metodos_Numericos/MBissec-Algoritmo$ java MetodoBisseccao -0.75 -0.5
k       |       a       |       b       |       Xk      |       F(Xk)
0       |       -0.625  |       -0.5    |       -0.625  |       0.09782214026807168
1       |       -0.625  |       -0.5625 |       -0.5625 |       -0.2697652434402471
2       |       -0.625  |       -0.59375        |       -0.59375        |       -0.09127323719619884
3       |       -0.609375       |       -0.59375        |       -0.609375       |       0.001958928454161235
4       |       -0.609375       |       -0.6015625      |       -0.6015625      |       -0.04498734600734666
5       |       -0.609375       |       -0.60546875     |       -0.60546875     |       -0.0215965977801158
6       |       -0.609375       |       -0.607421875    |       -0.607421875    |       -0.009839411631066497
7       |       -0.609375       |       -0.6083984375   |       -0.6083984375   |       -0.003945383268207259
8       |       -0.609375       |       -0.60888671875  |       -0.60888671875  |       -9.945125049735104E-4
```

Imagem 2 - Intervalo $[-\ 0.75,\ \ -\ 0.5]$.

Para o segundo intervalo, foi realizado 8 iterações :

```
barbosa@Lunella:~/Faculdade/5ºsemestre/Metodos_Numericos/MBissec-Algoritmo$ java MetodoBisseccao 0.25 0.5
k       |       a       |       b       |       Xk      |       F(Xk)
0       |       0.375   |       0.5     |       0.375   |       -0.34264955229925764
1       |       0.375   |       0.4375  |       0.4375  |       0.03045776795002908
2       |       0.40625 |       0.4375  |       0.40625 |       -0.16221556535142856
3       |       0.421875        |       0.4375  |       0.421875        |       -0.06741286198046281
4       |       0.4296875       |       0.4375  |       0.4296875       |       -0.01886156965203445
5       |       0.4296875       |       0.43359375      |       0.43359375      |       0.005702025249524123
6       |       0.431640625     |       0.43359375      |       0.431640625     |       -0.00660378206476131
7       |       0.4326171875    |       0.43359375      |       0.4326171875    |       -4.568819447607897E-4
```

Imagem 3 - Intervalo $[0.25,\ 0.5]$.

Podemos ter uma melhor visualização do resultado, pelo gráfico da função feito pelo Geogebra:

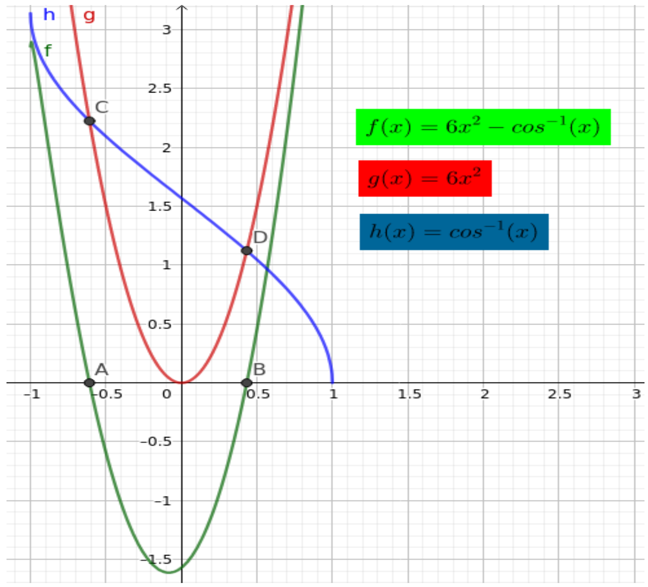

Imagem 4 - Gráfico da f(x), g(x) e h(x).

2. Considere a função $f(x)$ construída no exercício 1. Encontre uma aproximação para uma raiz de $f(x)$ localizada anteriormente com precisão de $\epsilon_1 = \epsilon_2 = 10^{-3}$:

a) Utilizando o método do ponto fixo, mostrando que a função iteração utilizada é válida.

```java
//MetodoPontoFixo.java
public class MetodoPontoFixo{

    public static double Fx(double x){
        double y = (6*(Math.pow(x,2.0) ))-(Math.acos(x));
        return y;
    }

    public static double Phix(double x){
        double y = Math.sqrt(( Math.acos(x) )/6);
        return y;
    }

    public static void main(String args[]){
        int k = 0;
        double a = Double.parseDouble(args[0]); //Inicio do intervalo pego por parametro
        double b = Double.parseDouble(args[1]);  //Fim do intervalo pego por parametro
        double e = 0.001; //Precissão

        System.out.println("k   |   Phi(Xk)     |   F(Phi)");

        double Xk = Phix(a);
        System.out.println(k+"   |   "+Xk+"   |   "+Fx(Xk));
        k++;
        while( Math.abs(Fx(Xk)) > e ){
            Xk = Phix(Xk);
            System.out.println(k+"   |   "+Xk+"   |   "+Fx(Xk));
            k++;
        }

    }
}
```

Imagem 5 - Algoritmo em Java™ para método do Ponto Fixo.

Escolhi o intervalo $[0.25,\ 0.5]$que se mostrou válido:

```
barbosa@Lunella:~/Faculdade/5ºsemestre/Metodos_Numericos/MPF-Algoritmo$ java MetodoPontoFixo 0.25 0.5
k       |       Phi(Xk)         |       F(Phi)
0       |       0.46870674407579876     |       0.23514592392744582
1       |       0.42484706027098196     |       -0.049033244838060375
2       |       0.4343584143238225      |       0.010532932889834257
3       |       0.43233290792967993     |       -0.002247497389672226
4       |       0.43276590228114303     |       4.802456124362031E-4
```

Imagem 6 - Intervalo $[0.25,\ 0.5]$.

b) Utilizando o método de Newton-Raphson escolhendo uma aproximação inicial adequada.

```java
//MetodoNewtonRaphson.java
public class MetodoNewtonRaphson{

    public static double Fx(double x){
        double y = (6*(Math.pow(x,2.0) ))-(Math.acos(x));
        return y;
    }

    public static double FxDerivada(double x){
        double y = (12*x)-( 1/( Math.sqrt( 1-Math.pow(x,2) ) ) );
        return y;
    }

    public static void main(String args[]){
        int k = 0;

        double a = Double.parseDouble(args[0]); //Inicio do intervalo pego por parametro
        double b = Double.parseDouble(args[1]);  //Fim do intervalo pego por parametro
        double e = 0.001; //Precissão
        double Xk = a;

        System.out.println("k   |       Xk      |   F(Xk)    |   F'(Xk)");
        Xk = Xk - (Fx(Xk)/FxDerivada(Xk));
        System.out.println(k+"  |   "+Xk+"  |   "+Fx(Xk)+"  |   "+FxDerivada(Xk) );
        k++;

        while( Math.abs(Fx(Xk)) > e ){
            Xk = Xk - (Fx(Xk)/FxDerivada(Xk));
            System.out.println(k+"  |   "+Xk+"  |   "+Fx(Xk)+"  |   "+FxDerivada(Xk) );
            k++;
        }

    }
}
```

Imagem 7 - Algoritmo em Java™ para método de Newton-Raphson.

Escolhi o intervalo $[0.25,\ 0.5]$que se mostrou válido:

```
barbosa@Lunella:~/Faculdade/5ºsemestre/Metodos_Numericos/MNR-Algoritmo$ java MetodoNewtonRaphson 0.25 0.5
k       |           Xk              |          F(Xk)    |         F'(Xk)
0       |       0.7294194502570126      |       2.438992971704515       |       7.291187674593284
1       |       0.3949070113297611      |       -0.22912040769743247    |       3.650414518512994
2       |       0.4576725976859005      |       0.16136469817387522     |       4.367364605566102
3       |       0.4207247555093578      |       -0.07449633216650331    |       3.946390126729273
4       |       0.43960183805168745     |       0.043857662279845444    |       4.161875288737958
5       |       0.4290638811843771      |       -0.022765282232869444   |       4.041682968896349
6       |       0.43469650561535417     |       0.012671234595810432    |       4.1059588558729025
7       |       0.4316104458079875      |       -0.006793551506609097   |       4.070751762881419
8       |       0.4332793148283378      |       0.003717659599463685    |       4.089793670459722
9       |       0.4323703057440126      |       -0.002011994777063375   |       4.079422611945058
10      |       0.43286351151562646     |       0.0010954824601170365   |       4.085049927704931
11      |       0.43259534283237794     |       -5.945128633157992E-4   |       4.081990280776831
```

Imagem 8 - Intervalo $[0.25,\ 0.5]$.

c) Utilizando o método da secante tomando como aproximação inicial os extremos do intervalo que contém a raiz.

```java
    public static double Fx(double x){
        double y = (6*(Math.pow(x,2.0) ))-(Math.acos(x));
        return y;
    }

    public static void main(String args[]){
        int k = 0;

        double a = Double.parseDouble(args[0]); //Inicio do intervalo pego por parametro
        double b = Double.parseDouble(args[1]);  //Fim do intervalo pego por parametro
        double e = 0.001; //Precissão
        double Xk,AuxA,AuxB;

        System.out.println("k   |       Xk      |   F(Xk)");
        Xk = a;
        AuxA = a;
        System.out.println(k+"  |   "+Xk+"  |   "+Fx(Xk) );
        k++;

        Xk = b;
        AuxB = b;
        System.out.println(k+"  |   "+Xk+"  |   "+Fx(Xk) );
        k++;

        while( Math.abs(Fx(Xk)) > e ){
                Xk = ( ( AuxA*Fx(AuxB) )-( AuxB*Fx(AuxA) ) )/ ( Fx(AuxB)-Fx(AuxA) );
                AuxA = AuxB;
                AuxB = Xk;
            System.out.println(k+"  |   "+Xk+"  |   "+Fx(Xk) );
            k++;
        }

    }
}
```

Imagem 9 - Algoritmo em Java™ para método da Secante.

Escolhi o intervalo $[0.25,\ 0.5]$que se mostrou válido:

```
barbosa@Lunella:~/Faculdade/5ºsemestre/Metodos_Numericos/MSec-Algoritmo$ java MetodoSecante 0.25 0.5
k       |               Xk              |               F(Xk)
0       |       0.25    |       -0.943116071652818
1       |       0.5     |       0.45280244880340215
2       |       0.41890600307827863     |       -0.0856627086834707
3       |       0.43180698903381476     |       -0.005557464430394621
4       |       0.43270202120347395     |       7.766429999445101E-5
```

Imagem 10 - Intervalo $[0.25,\ 0.5]$.

3. Analise os resultados do exercício 2 em relação a convergência.

Cada método apresentou diferentes iterações para o mesmo intervalo, entretanto a convergência era diferente para cada um. Sendo que do método da bissecção até o método da secante, o F(x) se afasta cada vez mais da precisão em sua última iteração, chegando cada vez mais próximo do zero.