UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO
DISCIPLINA: Cálculo Numérico para Computação - INE5409
PROFESSOR: Júlio Felipe Szeremeta

TRABALHO 02: ANÁLISE DOS MÉTODOS DE QUEBRA E ANÁLISE DA DIFRENÇA NA UTILIZAÇÃO DE DIVISÕES SUCESSIVAS E DA FORMA DE HORNER PARA OBTER VALORES DE $P_n^{(k)}(v)$

Glaucia De Pádua
Lucas Pereira Da Silva

Florianópolis
Maio de 2011

SUMÁRIO

1 ANÁLISE DOS MÉTODOS DE QUEBRA

Realizaremos a análise do desempenho da utilização de três métodos de quebra diferentes para solver a seguinte equação:

$$x-(\ln x)^x=0$$

Para solver esta equação usaremos os três métodos de quebra estudados: Método da bisseção, método da falsa posição e da falsa posição modificado.

1.1 Método Da Bisseção

Neste método tomaremos ***xm*** como sendo o ponto médio de ***a*** e ***b*** fornecidos e refinaremos a solução até a precisão desejada. O intervalo ***[a, b]*** irá sendo diminuído de forma que quando o sinal de ***f(a)*** for igual ao sinal de ***f(xm)*** então ***a*** passa a ser ***xm*** e caso contrário ***b*** passa a ser ***xm***. O pseudo código utilizado para a resolução do sistema através deste método é fornecido abaixo:

```
Procedimento métodoDaBisseção
    Ler a, b, e
    fa = f(a) : fb = f(b) : erro = |a-b|
    Se (fa>0 e fb>0) ou (fa<0 e fb<0) Faça:
        Escreva: "Solução não encontrada no intervalo"
        Interrompa Procedimento
    Fim Se
    Enquanto erro>=e Faça:
        xm = (a+b)/2 : fxm = f(xm)
        Se fxm igual a 0 Faça:
          Escreva: "Solução encontrada", xm
          Interrompa Procedimento
        Fim Se
        Se (fa>0 e fxm>0) ou (fa<0 e fxm<0) Faça:
          a = xm : fa = fxm
        Senão
          b = xm : fb = fxm
        Fim Se Senão
        erro = |a-b|
```

```
    Fim Enquanto
    Escreva: "Solução encontrada", xm
Fim Procedimento

Procedimento f
    Receber x
    Retornar x - (ln x)^x
Fim Procedimento
```

O pseudo código acima foi implementado na linguagem Java e obteve os seguintes resultados:

a	xm	b	f(a)	f(xm)	f(b)	erro
3	4	5	1,6740310399	0,3066384227	-5,7986915783	2
4	4,5	5	0,3066384227	-1,7764754052	-5,7986915783	1
4	4,25	4,5	0,3066384227	-0,5571526285	-1,7764754052	0,5
4	4,125	4,25	0,3066384227	-0,0869586301	-0,5571526285	0,25
4	4,0625	4,125	0,3066384227	0,1187418587	-0,0869586301	0,125
4,0625	4,09375	4,125	0,1187418587	0,0181973447	-0,0869586301	0,0625
4,09375	4,109375	4,125	0,0181973447	-0,0337938553	-0,0869586301	0,03125
4,09375	4,1015625	4,109375	0,0181973447	-0,0076528677	-0,0337938553	0,015625
4,09375	4,09765625	4,1015625	0,0181973447	0,0053084231	-0,0076528677	0,0078125
4,09765625	4,099609375	4,1015625	0,0053084231	-0,0011631559	-0,0076528677	0,00390625
4,09765625	4,0986328125	4,099609375	0,0053084231	0,0020748977	-0,0011631559	0,001953125
4,0986328125	4,0991210938	4,099609375	0,0020748977	0,0004564372	-0,0011631559	0,0009765625
4,0991210938	4,0993652344	4,099609375	0,0004564372	-0,0003532177	-0,0011631559	0,00048828125
4,0991210938	4,0992431641	4,0993652344	0,0004564372	0,0000516451	-0,0003532177	0,00024414063
4,0992431641	4,0993041992	4,0993652344	0,0000516451	-0,0001507774	-0,0003532177	0,00012207031
4,0992431641	4,0992736816	4,0993041992	0,0000516451	-0,0000495639	-0,0001507774	0,00006103516
4,0992431641	4,0992584229	4,0992736816	0,0000516451	0,0000010412	-0,0000495639	0,00003051758
4,0992584229	4,0992660522	4,0992736816	0,0000010412	-0,0000242613	-0,0000495639	0,00001525879
4,0992584229	4,0992622375	4,0992660522	0,0000010412	-0,0000116100	-0,0000242613	0,00000762939
4,0992584229	4,0992603302	4,0992622375	0,0000010412	-0,0000052844	-0,0000116100	0,00000381470
4,0992584229	4,0992593765	4,0992603302	0,0000010412	-0,0000021216	-0,0000052844	0,00000190735
4,0992584229	4,0992588997	4,0992593765	0,0000010412	-0,0000005402	-0,0000021216	0,00000095367
4,0992584229	4,0992586613	4,0992588997	0,0000010412	0,0000002505	-0,0000005402	0,00000047684
4,0992586613	4,0992587805	4,0992588997	0,0000002505	-0,0000001449	-0,0000005402	0,00000023842
4,0992586613	4,0992587209	4,0992587805	0,0000002505	0,0000000528	-0,0000001449	0,00000011921
4,0992587209	4,0992587507	4,0992587805	0,0000000528	-0,0000000461	-0,0000001449	0,00000005960
4,0992587209	4,0992587358	4,0992587507	0,0000000528	0,0000000034	-0,0000000461	0,00000002980
4,0992587358	4,0992587432	4,0992587507	0,0000000034	-0,0000000213	-0,0000000461	0,00000001490
4,0992587358	4,0992587395	4,0992587432	0,0000000034	-0,0000000090	-0,0000000213	0,00000000745
4,0992587358	4,0992587376	4,0992587395	0,0000000034	-0,0000000028	-0,0000000090	0,00000000373
4,0992587358	4,0992587367	4,0992587376	0,0000000034	0,0000000003	-0,0000000028	0,00000000186
4,0992587367	4,0992587372	4,0992587376	0,0000000003	-0,0000000013	-0,0000000028	0,00000000093
4,0992587367	4,0992587369	4,0992587372	0,0000000003	-0,0000000005	-0,0000000013	0,00000000047
4,0992587367	4,0992587368	4,0992587369	0,0000000003	-0,0000000001	-0,0000000005	0,00000000023
4,0992587367	**4,0992587368**	4,0992587368	0,0000000003	0,0000000001	-0,0000000001	0,00000000012

Portanto o ***x*** encontrado com 10 dígitos de segurança é ***4,0992587368***. Para encontrar o valor de ***x*** com 10 dígitos de segurança neste método precisamos de 35 iterações.

1.2 Método Da Falsa Posição

Este método se assemelha ao método da bisseção com a diferença que não usaremos o ***x*** médio de ***[a, b]***, mas sim um ***x*** por onde passa a reta que liga ***f(a)*** a ***f(b)***. Outra diferença é que controlaremos a saída do algorítimo por ***x*** e x_0 ao invés de controlar por ***a*** e ***b***. Assim o pseudo código ficou da seguinte maneira:

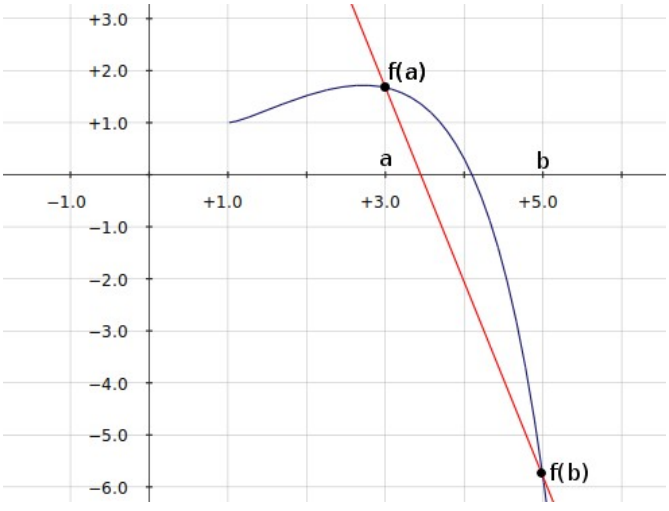
```
Procedimento métodoDaFalsaPosição
    Ler a, b, e
    fa = f(a) : fb = f(b) : erro = |a-b|
    Se (fa>0 e fb>0) ou (fa<0 e fb<0) Faça:
        Escreva: "Solução não encontrada no intervalo"
        Interrompa Procedimento
    Fim Se
    xx = (a-(fa*(b-a))/(fb-fa))*10
    Enquanto erro>=e Faça:
        xx0 = xx
        xx = a-(fa*(b-a))/(fb-fa) : fxx = f(xx)
        Se fxx igual a 0 Faça:
          Escreva: "Solução encontrada", xx
          Interrompa Procedimento
        Fim Se
        Se (fa>0 e fxx>0) ou (fa<0 e fxx<0) Faça:
          a = xm : fa = fxx
        Senão
          b = xx : fb = fxx
        Fim Se Senão
        erro = |xx-xx0|
    Fim Enquanto
    Escreva: "Solução encontrada", xx
Fim Procedimento

Procedimento f
    Receber x
    Retornar x - (ln x)^x
Fim Procedimento
```

Ao tentarmos resolver o problema proposto por esse método fizemos 33 iterações. O que percebemos analisando a tabela mais abaixo contendo as 33 iterações é uma estagnação no ponto ***b***. Estagnação esta que aumenta o tempo de resolução do método já que um lado da função fica fixo e apenas o outro lado vai se aproximando da solução.

Assim a solução encontrada foi: **4,0992587367**.

a	xm	b	f(a)	f(xm)	f(b)	erro
3,0000000000	3,4480377837	5,0000000000	1,6740310399	1,3612956981	-5,7986915783	2,0000000000
3,4480377837	3,7431052757	5,0000000000	1,3612956981	0,9168139982	-5,7986915783	4,0323400529
3,7431052757	3,9146990097	5,0000000000	0,9168139982	0,5365573761	-5,7986915783	0,2950674920
3,9146990097	4,0066174577	5,0000000000	0,5365573761	0,2875493978	-5,7986915783	0,1715937340
4,0066174577	4,0535506227	5,0000000000	0,2875493978	0,1467092684	-5,7986915783	0,0919184480
4,0535506227	4,0769052962	5,0000000000	0,1467092684	0,0729563673	-5,7986915783	0,0469331650
4,0769052962	4,0883749274	5,0000000000	0,0729563673	0,0358150307	-5,7986915783	0,0233546735
4,0883749274	4,0939709241	5,0000000000	0,0358150307	0,0174703167	-5,7986915783	0,0114696312
4,0939709241	4,0966924121	5,0000000000	0,0174703167	0,0084953806	-5,7986915783	0,0055959967
4,0966924121	4,0980138681	5,0000000000	0,0084953806	0,0041248283	-5,7986915783	0,0027214880
4,0980138681	4,0986550288	5,0000000000	0,0041248283	0,0020012840	-5,7986915783	0,0013214560
4,0986550288	4,0989659998	5,0000000000	0,0020012840	0,0009706355	-5,7986915783	0,0006411607
4,0989659998	4,0991167975	5,0000000000	0,0009706355	0,0004706827	-5,7986915783	0,0003109710
4,0991167975	4,0991899167	5,0000000000	0,0004706827	0,0002282253	-5,7986915783	0,0001507977
4,0991899167	4,0992253694	5,0000000000	0,0002282253	0,0001106577	-5,7986915783	0,0000731192
4,0992253694	4,0992425587	5,0000000000	0,0001106577	0,0000536526	-5,7986915783	0,0000354527
4,0992425587	4,0992508930	5,0000000000	0,0000536526	0,0000260133	-5,7986915783	0,0000171893
4,0992508930	4,0992549338	5,0000000000	0,0000260133	0,0000126124	-5,7986915783	0,0000083342
4,0992549338	4,0992568929	5,0000000000	0,0000126124	0,0000061151	-5,7986915783	0,0000040408
4,0992568929	4,0992578428	5,0000000000	0,0000061151	0,0000029648	-5,7986915783	0,0000019592
4,0992578428	4,0992583033	5,0000000000	0,0000029648	0,0000014375	-5,7986915783	0,0000009499
4,0992583033	4,0992585266	5,0000000000	0,0000014375	0,0000006970	-5,7986915783	0,0000004606
4,0992585266	4,0992586349	5,0000000000	0,0000006970	0,0000003379	-5,7986915783	0,0000002233
4,0992586349	4,0992586874	5,0000000000	0,0000003379	0,0000001638	-5,7986915783	0,0000001083
4,0992586874	4,0992587128	5,0000000000	0,0000001638	0,0000000794	-5,7986915783	0,0000000525
4,0992587128	4,0992587252	5,0000000000	0,0000000794	0,0000000385	-5,7986915783	0,0000000255
4,0992587252	4,0992587312	5,0000000000	0,0000000385	0,0000000187	-5,7986915783	0,0000000123
4,0992587312	4,0992587341	5,0000000000	0,0000000187	0,0000000091	-5,7986915783	0,0000000060
4,0992587341	4,0992587355	5,0000000000	0,0000000091	0,0000000044	-5,7986915783	0,0000000029
4,0992587355	4,0992587361	5,0000000000	0,0000000044	0,0000000021	-5,7986915783	0,0000000014
4,0992587361	4,0992587365	5,0000000000	0,0000000021	0,0000000010	-5,7986915783	0,0000000007
4,0992587365	4,0992587366	5,0000000000	0,0000000010	0,0000000005	-5,7986915783	0,0000000003
4,0992587366	**4,0992587367**	5,0000000000	0,0000000005	0,0000000002	-5,7986915783	0,0000000002

O gráfico abaixo mostra em azul a função inicial do nosso problema e em vermelho a reta ligando os primeiros pontos tomados: ***a*** e ***b*** (3 e 5). Analisando o gráfico podemos perceber claramente que ***b*** ficará estagnado.



1.3 Método Da Falsa Posição Modificado

No método anterior nos deparamos com o problema da estagnação em ***b***. Neste método contornaremos esse problema dividindo ***f(x)*** por 2 sempre que houver um principio de estagnação. Segue abaixo o algorítimo utilizado:

```
Procedimento métodoDaFalsaPosiçãoModificado
    Ler a, b, e
    fa = f(a) : fb = f(b) : erro = |a-b| : fxx0 = fa
    Se (fa>0 e fb>0) ou (fa<0 e fb<0) Faça:
        Escreva: "Solução não encontrada no intervalo"
        Interrompa Procedimento
    Fim Se
    Enquanto erro>=e Faça:
        xx = a-(fa*(b-a))/(fb-fa) : fxx = f(xx)
        Se fxx igual a 0 Faça:
          Escreva: "Solução encontrada", xx
          Interrompa Procedimento
        Fim Se
        Se (fa>0 e fxx>0) ou (fa<0 e fxx<0) Faça:
          a = xm : fa = fxx
          Se (fxx>0 e fxx0>0) ou (fxx<0 e fxx0<0) Faça:
               fb = fb/2
          Fim Se
        Senão
          b = xx : fb = fxx
          Se (fxx>0 e fxx0>0) ou (fxx<0 e fxx0<0) Faça:
               fa = fa /2
          Fim Se
        Fim Se Senão
        erro = |a-b|
    Fim Enquanto
    Escreva: "Solução encontrada", xx
Fim Procedimento

Procedimento f
    Receber x
    Retornar x - (ln x)^x
Fim Procedimento
```

Ao executador o método da falsa posição modificada percebemos que ele converge muito mais rapidamente que o método da bisseção, além de não possuir o problema da estagnação encontrado no método da falsa posição. Para solver o problema com este método utilizamos apenas 10 iterações.

a	xx	b	f(a)	f(xx)	f(b)	erro
3	3,4480377837	5	1,6740310399	1,3612956981	-5,7986915783	2
3,4480377837	3,9438972675	5,0000000000	1,3612956981	0,4611500238	-2,8993457892	1,5519622164
3,9438972675	4,1987726908	5,0000000000	0,4611500238	-0,3544771913	-1,4496728946	1,0561027325
3,9438972675	4,0880020808	4,1987726908	0,4611500238	0,0370320587	-0,3544771913	0,2548754233
4,0880020808	4,1071463903	4,1987726908	0,0370320587	-0,0263070294	-0,1772385957	0,1107706101
4,0880020808	4,0991950616	4,1071463903	0,0370320587	0,0002111642	-0,0263070294	0,0191443096
4,0991950616	4,0993206939	4,1071463903	0,0002111642	-0,0002054850	-0,0131535147	0,0079513288
4,0991950616	4,0992587340	4,0993206939	0,0002111642	0,0000000094	-0,0002054850	0,0001256324
4,0992587340	4,0992587396	4,0993206939	0,0000000094	-0,0000000094	-0,0001027425	0,0000619600
4,0992587340	**4,0992587368**	4,0992587396	0,0000000094	0,0000000000	-0,0000000094	0,0000000057

Na décima iteração deste método encontramos a solução do sistema com ***x*** igual a ***4,0992587368***.

2 ANÁLISE DO MÉTODO DE DIVISÃO SUCESSIVAS E FORMA DE HORNER

Faremos aqui uma análise para comprovar que existe uma diferença na precisão ao se calcular $P_n^{(k)}(v)$ pelo forma da divisão sucessiva e pela forma de Horner calculado diretamente os coeficientes. Para realizar essa análise usaremos um polinômio de grau 20 onde $a_1 = 1$ e $a_i = \ln i$. Tomaremos também $v = \sqrt{2}$.

2.1 Forma De Horner Com Coeficientes Calculados Diretamente

A forma de **Horner** consiste em "aninhar" um polinômio a fim de diminuir o número de operações para se obter $P_n(v)$. Na forma "tradicional" calcularíamos $P_n(v)$ da seguinte maneira:

$$p_n(v) = a_1 v^n + \ldots + a_n v + a_{n+1}$$

Assim, teríamos ***n*** somas e ***2n-1*** multiplicações para encontrar $P_n(v)$. Porém se fizermos pela forma de Horner teremos ***n*** somas e ***n*** multiplicações, sendo uma ótima melhora. A forma de Horner é apresentada da seguinte maneira:

$$p_n(v) = (((a_1 v + a_2) v + a_3) v + \cdots + a_n) v + a_{n+1}$$

No pseudo código abaixo geraremos $P_n^{(k)}(v)$ para o problema apresentado anteriormente utilizando a forma de Horner com os coeficientes calculados diretamente. A cada ***k*** iteração geraremos os coeficientes do polinômio da derivada **k+1** e obteremos $P_n^{(k)}(v)$ através da multiplicação "aninhada".

```
Procedimento obterValoresPorHorner
    Ler grau, v, coeficientes
    valores₀ = calcularValor(grau, v, coeficientes)
    novoGrau = grau
    Repita i = 1 até grau
        coeficientes = calcularDerivada(novoGrau, coeficientes)
        valoresᵢ = calcularValor(novoGrau-1, v, coeficientes)
        novoGrau = novoGrau-1
    Fim Repita
    Escrever: "Valores de Pₙ⁽ᵏ⁾(x): ", (valoresₖ, k = 0, grau)
Fim Procedimento
```

```
Procedimento calcularValor
    Receber grau, v, (coeficientes_i, i = 1, grau+1)
    valor = coeficientes_1
    Repita i = 1 até grau
        valor = valor*v + coeficientes_i+1
    Fim Repita
    Retornar valor
Fim Procedimento

Procedimento calcularDerivada
    Receber grau, coeficientes
    coeficientes_grau+1 = 0
    Repita i = 1 até grau
        posicao = grau - i
        coeficientes_posicao+1 = coeficientes_posicao+1*i
    Fim Repita
    Retornar coeficientes
Fim Procedimento
```

O pseudo código acima foi implementado (assim como os demais códigos neste trabalho) na linguagem Java e se encontra anexo ao trabalho.

2.2 Forma Das Divisões Sucessivas

Para obter os valores de $P_n^{(k)}(v)$ nesta forma usaremos divisões sucessivas do polinômio e a cada divisão pegaremos o valor de $P_n^{(k)}(v)$. Esta forma se assemelha com a de Horner com a exceção que não calcularemos os coeficientes da derivada. O que faremos neste caso é usar como coeficientes da derivada os coeficientes resultantes da divisão do polinômio da derivada anterior. Ou seja a derivada ***k+1*** será calculada dividindo-se o polinômio da derivada ***k***.

```
Procedimento obterValoresPorDivisõesSucessivas
    Ler grau, v, (coeficientes_i, i = 1, grau+1)
    novoGrau = grau
    coeficientes = dividirPolinômio(grau, v, coeficientes)
    valores_0 = coeficientes_novoGrau+1
    fatorial = 1
    Repita i = 1 até grau
        novoGrau = novoGrau-1
        fatorial = fatorial*i
```

```
        coeficientes = dividirPolinô(novoGrau, v, coeficientes)
        valores_i = coeficientes_novoGrau+1*fatorial
    Fim Repita
    Escrever: "Valores de P_n^(k)(x): ", (valores_k, k = 0, grau)
Fim Procedimento

Procedimento dividirPolinômio
    Receber grau, v, coeficientes
    novosCoeficientes_1 = coeficientes_1
    Repita i = 1 até grau
       novosCoeficientes_i+1 = novosCoeficientes_i*v+coeficientes_i+1
    Fim Repita
    Retornar novosCoeficientes_1
Fim Procedimento
```

O código implementado na linguagem Java se encontra no Anexo E deste trabalho.

2.3 Comparação Entre Forma De Horner e Forma Das Divisões Sucessivas

Analisaremos aqui as duas formas utilizadas para calcular todos os valores de $P_n^{(k)}(v)$. Ambos os métodos calculam $P_n^{(k)}(v)$ de forma semelhante, através da multiplicação "aninhada". A diferença está na geração dos coeficientes de cada polinômio. Na forma de Horner cada ***k*** derivada do polinômio é calculada separadamente, utilizando apenas a derivada anterior. Já na forma de divisões sucessivas calculamos cada ***k*** derivada através da divisão do polinômio ***k-1***.

A diferença nesse caso é que para a forma de divisões sucessivas será utilizado o valor ***v*** para calcular a próxima derivada, enquanto que na forma de Horner não. Esse é um fator que influenciará na precisão de cada valor $P_n^{(k)}(v)$ já que na forma das divisões sucessivas teremos recursão dentro de recursão e devido a isso se ***v*** for afetado por erro de arredondamento então nosso algorítimo sofrerá de instabilidade numérica.

Assim se espera que o algorítimo da forma de Horner com os coeficientes calculados diretamente seja menos afetado por erros de arredondamento em relação a forma das divisões sucessivas.

Na tabela abaixo calculamos cada valor de $P_n^{(k)}(v)$ para cada forma,

mostrando assim existir diferença na precisão do cálculo de $P_n^{(k)}(v)$.

Forma Das Divisões Sucessivas	Forma De Horner
4282,6275668966155	4282,6275668966155
50206,558213253378	50206,558213253379
578391,7684093146	578391,7684093146
6454789,165637489	6454789,165637489
69182126,18057689	69182126,18057688
707264945,7482531	707264945,7482531
6853954583,530083	6853954583,530082
62574877214,44345	62574877214,44346
534775517727,4062	534775517727,40594
4248435276224,7227	4248435276224,724
31130027967516,01	31130027967516,008
208503695579257,00	208503695579257,03
1263015589672722,2	1263015589672722,2
6830428553201302	6830428553201303
32448919743584132	32448919743584144
132599876673622112	132599876673622160
452996266149254530	452996266149254590
12416392424602380 80	12416392424602383 40
2559179334895205400	2559179334895205400
3524960974265457700	3524960974265456600
2432902008176640000	2432902008176640000

Os dígitos em vermelho na tabela representam os dígitos que diferem em cada forma. O que percebemos aqui ao se fazer uma análise na tabela é que neste caso em especifico a diferença na precisão da utilização de um método ou de outro não foi tão grande, podendo ser até indiferente o uso de uma forma ou de outra dependendo da precisão desejada.

O importante a se ressaltar é que apesar de nesse caso a diferença de precisão ser relativamente pequena, existe sim diferença ao se usar um método ou outro e essa diferença dependerá de cada problema e tende a aumentar se o grau do polinômio for maior.

ANEXO A – MÉTODO DA BISSEÇÃO

```
public class MetodoDaBissecao
{
	private static final double ZERO = 0.0;
	private static final int DOIS = 2;

	public static double calcular(double a, double b, double e)
	{
		double fa = f(a);
		double fb = f(b);
		double xm = Double.NaN;
		if (verificarSeTemMesmoSinal(fa, fb)) return xm;
		double erro = Math.abs(a-b);

		while (erro >= e)
		{
			xm = (a+b)/DOIS;
			double fxm = f(xm);
			if (fxm == ZERO)
			{
				return xm;
			}
			System.out.printf("%.10f %.10f %.10f %.10f %.10f %.10f
				%.11f\n", a, xm, b, fa, fxm, fb, erro);
			if (verificarSeTemMesmoSinal(fa, fxm))
			{
				a = xm;
				fa = fxm;
			}
			else
			{
				b = xm;
				fb = fxm;
			}
			erro = Math.abs(a-b);
		}

		return xm;
	}

	private static double f(double x)
	{
		double ln = Math.log(x);
		double elevadoLn = Math.pow(ln, x);

		return x-elevadoLn;
	}

	private static boolean verificarSeTemMesmoSinal(double a, double b)
	{
		return (a > ZERO && b > ZERO) || (a < ZERO && b < ZERO);
	}
}
```

ANEXO B – MÉTODO DA FALSA POSIÇÃO

```
public class MetodoDaFalsaPosicao
{
	private static final double ZERO = 0.0;

	public static double calcular(double a, double b, double e)
	{
		double fa = f(a);
		double fb = f(b);
		double xx = (a-fa*(b-a)/(fb-fa))*10;
		if (verificarSeTemMesmoSinal(fa, fb))
		{
			return Double.NaN;
		}
		double erro = Math.abs(a-b);
		while (erro >= e)
		{
			double xx0 = xx;
			xx = a-fa*(b-a)/(fb-fa);
			double fxx = f(xx);
			if (fxx == ZERO)
			{
				return xx;
			}
			System.out.printf("%.10f %.10f %.10f %.10f %.10f %.10f
				%.11f\n", a, xx, b, fa, fxx, fb, erro);
			if (verificarSeTemMesmoSinal(fa, fxx))
			{
				a = xx;
				fa = fxx;
			}
			else
			{
				b = xx;
				fb = fxx;
			}
			erro = Math.abs(xx-xx0);
		}

		return xx;
	}

	private static double f(double x)
	{
		double ln = Math.log(x);
		double elevadoLn = Math.pow(ln, x);

		return x-elevadoLn;
	}

	private static boolean verificarSeTemMesmoSinal(double a, double b)
	{
		return (a > ZERO && b > ZERO) || (a < ZERO && b < ZERO);
	}
}
```

ANEXO C – MÉTODO DA FALSA POSIÇÃO MODIFICADO

```
public class MetodoDaFalsaPosicaoModificado
{
	private static final double ZERO = 0.0;
	private static final int DOIS = 2;

	public static double calcular(double a, double b, double e)
	{
		double fa = f(a);
		double fb = f(b);
		double xx = Double.NaN;
		double fxx0 = fa;
		if (verificarSeTemMesmoSinal(fa, fb))
		{
			return xx;
		}
		double erro = Math.abs(a-b);

		while (erro >= e)
		{
			xx = a-fa*(b-a)/(fb-fa);
			double fxx = f(xx);
			System.out.printf("%.10f %.10f %.10f %.10f %.10f %.10f
				%.11f\n", a, xx, b, fa, fxx, fb, erro);
			if (fxx == ZERO)
			{
				return xx;
			}
			if (verificarSeTemMesmoSinal(fa, fxx))
			{
				a = xx;
				fa = fxx;
				if (verificarSeTemMesmoSinal(fxx, fxx0))
				{
					fb = fb/DOIS;
				}
			}
			else
			{
				b = xx;
				fb = fxx;
				if (verificarSeTemMesmoSinal(fxx, fxx0))
				{
					fa = fa/DOIS;
				}
			}
			erro = Math.abs(a-b);
		}

		return xx;
	}

	private static double f(double x)
	{
		double ln = Math.log(x);
```

```
            double elevadoLn = Math.pow(ln, x);

            return x-elevadoLn;
      }

      private static boolean verificarSeTemMesmoSinal(double a, double b)
      {
            return (a > ZERO && b > ZERO) || (a < ZERO && b < ZERO);
      }
}
```

ANEXO D – FORMA DE HORNER COM COEFICIENTES CALCULADOS DIRETAMENTE

```
public class ValoresDoPolinomioPorHorner
{
	private static final int ZERO = 0;
	private static final int UM = 1;

	public static double[] calcular(int grau, double v, double[]
		coeficientes)
	{
		double[] valoresDoPolinomio = new double[21];
		valoresDoPolinomio[ZERO] = calcularValor(grau, v,
			coeficientes);
		int novoGrau = grau;
		for (int cont = UM; cont <= grau; cont++, novoGrau--)
		{
			coeficientes = calcularCoeficientesDaDerivada(novoGrau,
				coeficientes);
			valoresDoPolinomio[cont] = calcularValor(novoGrau-UM, v,
				coeficientes);
		}

		return valoresDoPolinomio;
	}

	private static double calcularValor(double grau, double v, double[]
		coeficientes)
	{
		double valor = coeficientes[ZERO];
		for (int cont = UM; cont <= grau; cont++)
		{
			valor = valor*v+coeficientes[cont];
		}

		return valor;
	}

	private static double[] calcularCoeficientesDaDerivada(int grau,
		double[] coeficientes)
	{
		double[] coeficientesDaDerivada = new double[grau];
		for (int cont = UM; cont <= grau; cont++)
		{
			int posicao = grau-cont;
			coeficientesDaDerivada[posicao] =
				coeficientes[posicao]*cont;
		}

		return coeficientesDaDerivada;
	}
}
```

ANEXO E – FORMA DAS DIVISÕES SUCESSIVAS

```
public class ValoresDoPolinomioPorDivisoesSucessivas
{
	private static final int ZERO = 0;
	private static final int UM = 1;

	public static double[] calcular(int grau, double v, double[]
		coeficientes)
	{
		double[] valoresDoPolinomio = new double[grau+UM];
		double[] polinomioDividido = dividirPolinomio(grau, v,
			coeficientes);
		valoresDoPolinomio[ZERO] =
			polinomioDividido[polinomioDividido.length-UM];
		long fatorial = 1;
		for (int cont = UM; cont <= grau; cont++)
		{
			int novoGrau = grau-cont;
			fatorial *= cont;
			polinomioDividido = dividirPolinomio(novoGrau, v,
				polinomioDividido);
			valoresDoPolinomio[cont] =
				fatorial*polinomioDividido
					[polinomioDividido.length-UM];
		}

		return valoresDoPolinomio;
	}

	public static double[] dividirPolinomio(int grau, double v, double[]
		coeficientes)
	{
		double[] coeficientesDoPolinomioDividido = new double[grau+UM];
		coeficientesDoPolinomioDividido[ZERO] = coeficientes[ZERO];
		for (int cont = UM; cont <= grau; cont++)
		{
			coeficientesDoPolinomioDividido[cont] =
				coeficientesDoPolinomioDividido[cont-
					UM]*v+coeficientes[cont];
		}

		return coeficientesDoPolinomioDividido;
	}
}
```