**Matrícula:** 21554923

**Aluno:** Micael Levi Lima Cavalcante

**Curso:** Bacharelado em Ciência da Computação

**Marca do computador:** ASUS

**Marca e modelo do processador:** Intel Core i7-2600 CPU @ 3.40GHz

**Sistema operacional:** Windows 10 Home

**Horário do teste:** 21:00 horas

**Tempo de duração:** 37 minutos

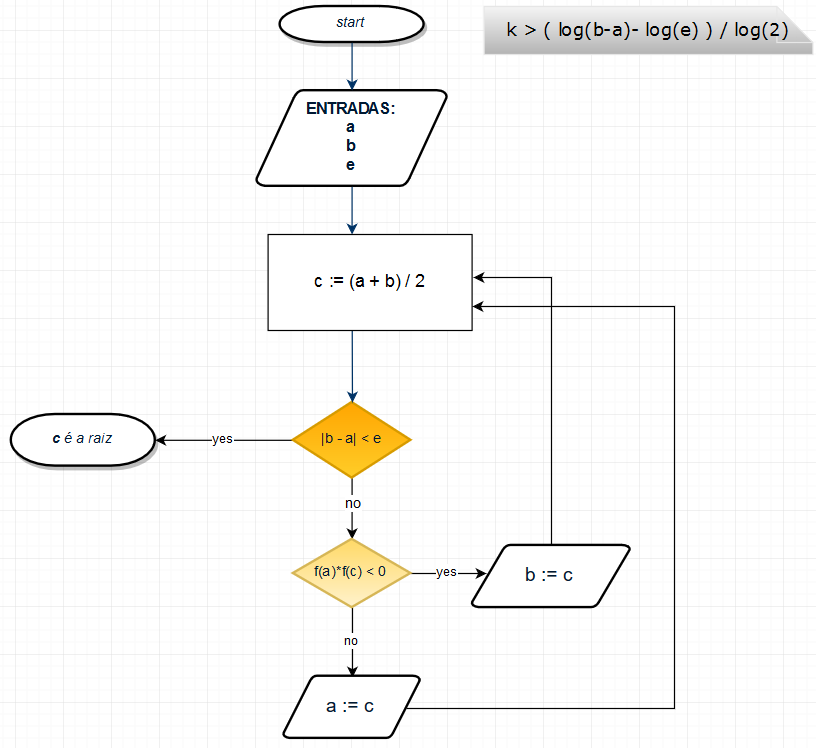
**Compilador:** GNU GCC

**Ambiente de compilação e execução utilizados:** Atom *(editor de texto)*; Cygwin *(bash no windows)*

Nas implementações dos algoritmos utilizados para encontrar uma raiz real de um polinômio qualquer, adotamos o número de iterações de no máximo 15 para evitar que o programa entre em *loop* infinito caso não haja convergência. Exceto no método da bissecção, pois este possui uma equação para determinar o número máximo de iterações. Foi utilizada a linguagem C++11 para melhor implementar tais métodos.

Cada método foi implementado em um módulo diferente para facilitar o uso do programa. Para uma comparação mais adequada, calculamos (a mão) o intervalo inicial (ou aproximação inicial) para cada função.

**Método da Bissecção**

Para calcular uma raiz do polinômio fornecido, utilizamos o seguinte algoritmo:

Dado os valores de entrada: intervalo inicial e a precisão a ser atingida, além que calcularmos o *k* que é o número máximo de iterações,

a = extremante esquerdo

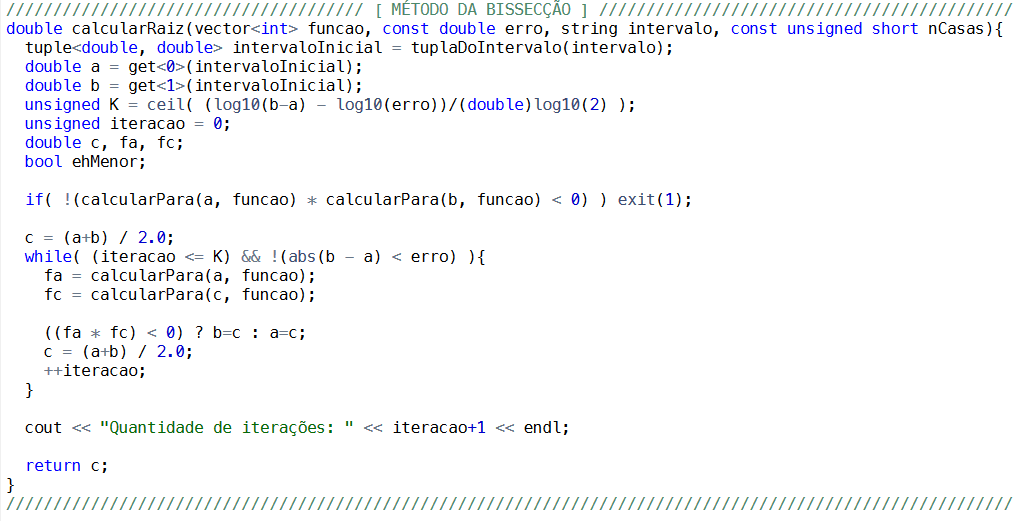
b = extremante direito

e = erro

Onde a função *f(x)* passada é contínua no intervalo *[a, b]* e tal que *f(a).f(b) < 0*, temos que ir reduzindo a amplitude do intervalo (que com certeza contém uma raiz) até se atingir a precisão fornecida. Por isso, utilizamos sucessivas divisões entre *a* e *b* e perguntamos se o módulo da diferença entre eles é menor que o erro. Se for então encerramos o programa e a raiz (aproximada) é *c* (a média entre os extremantes).

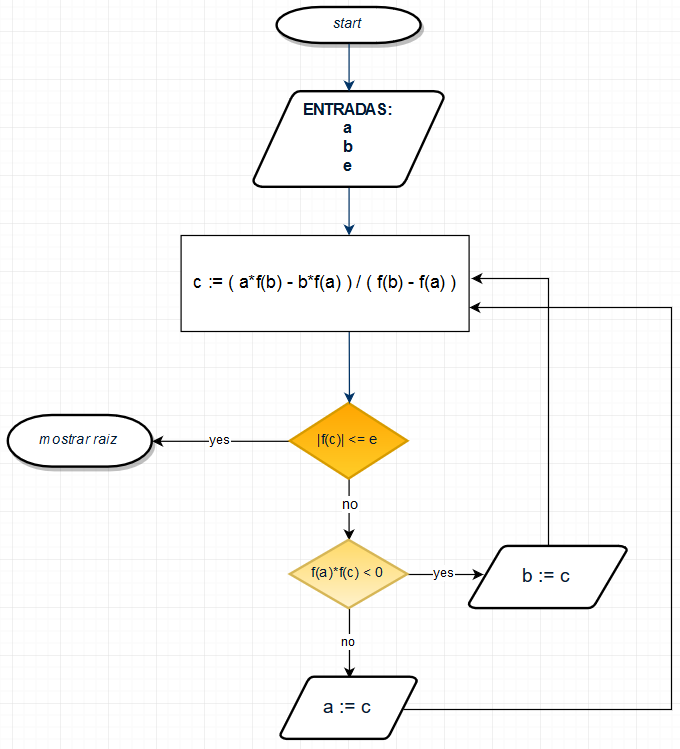
Caso contrário, atualizamos o intervalo com o objetivo de reduzir a distância entre *c* e uma raiz.

O arquivo usarBisseccao.hpp contém o seguinte código que implementa o algoritmo anterior:



Como o programa só aceita funções polinomiais, temos que a hipótese de continuidade de *f(x)* é sempre satisfeita. Além disso, garantimos que no intervalo inicial *[a, b]* há troca de sinal. Com isso, esse programa sempre irá convergir, i.e., é sempre possível obter um intervalo que contém pelo menos uma raiz. Então temos que a vantagem desse método é que a fácil implementação. Porém a convergência é claramente lenta e precisa de dois pontos para iniciarmos.

**Método da Falsa Posição**

Para calcular uma raiz do polinômio fornecido, utilizamos o seguinte algoritmo:

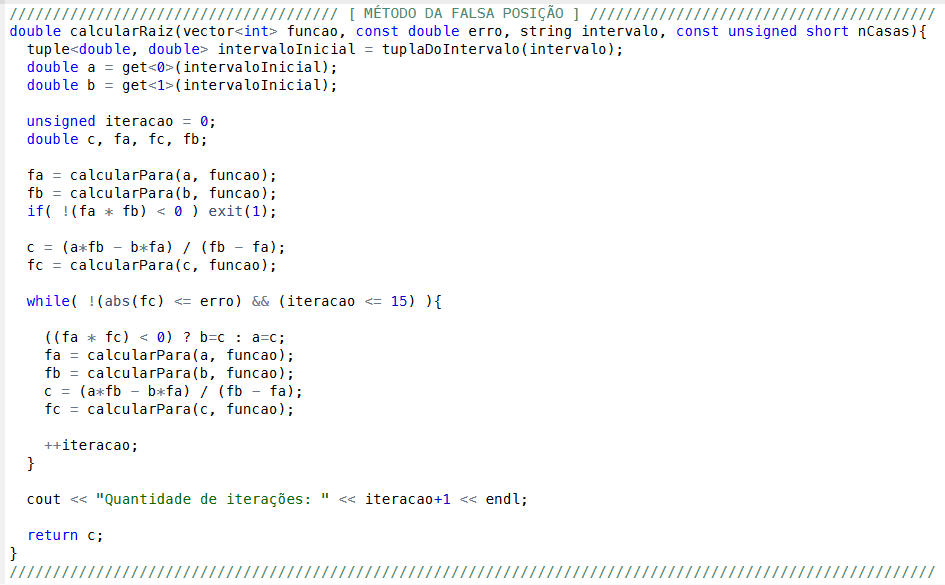
Este método é praticar idêntico ao anterior. As únicas diferenças estão na forma de calcular a aproximação da raiz e na condição de parada do programa (caso uma raiz tenha sido encontrada).

Em vez de calcularmos a média aritmética entre os extremos do intervalo (*a* e *b*), tomamos a média aritmética ponderada entre eles. Assim garantimos melhor precisão para a raiz (a variável *c*).

Caso o módulo de *f(c)* seja menor ou igual ao erro *e*, então *c* será a raiz buscada. Senão, continuamos o processo iterativo da mesma forma que o método da bissecção.

Porém, dessa vez não podemos estimar o número de iterações, então foi definido que será no máximo 15.

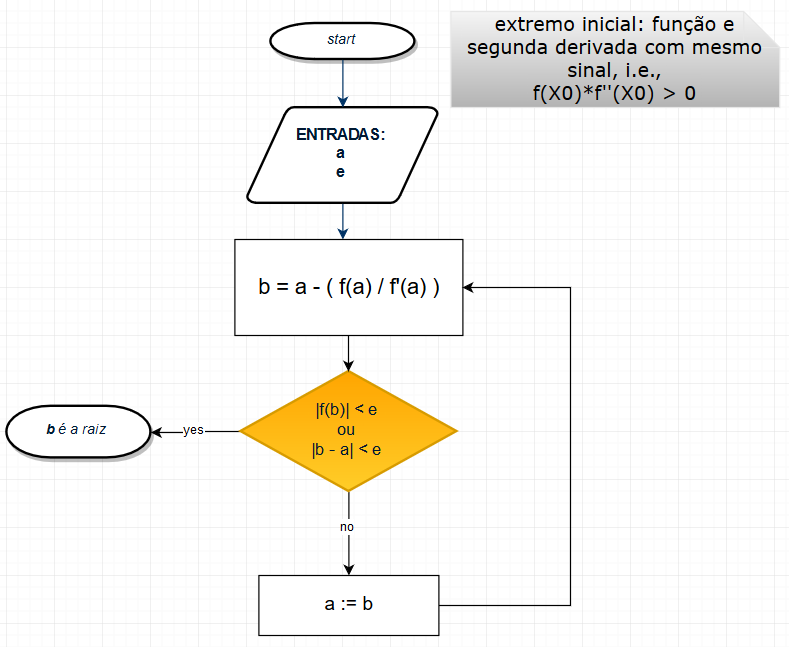
O arquivo usarFalsaPoiscao.hpp contém o seguinte trecho de código que implementa o algoritmo anterior:



Esse método “herda” as vantagens e desvantagens do método da bissecção, exceto a convergência lenta, pois o novo cálculo de *c* acelera esse processo.

**Método de Newton-Raphson**

Para calcular uma raiz do polinômio fornecido, utilizamos o seguinte algoritmo:



As entradas agora são diferentes. Não é mais necessário um intervalo inicial, basta apenas um valor aproximado da raiz para começar a iteração *i*.

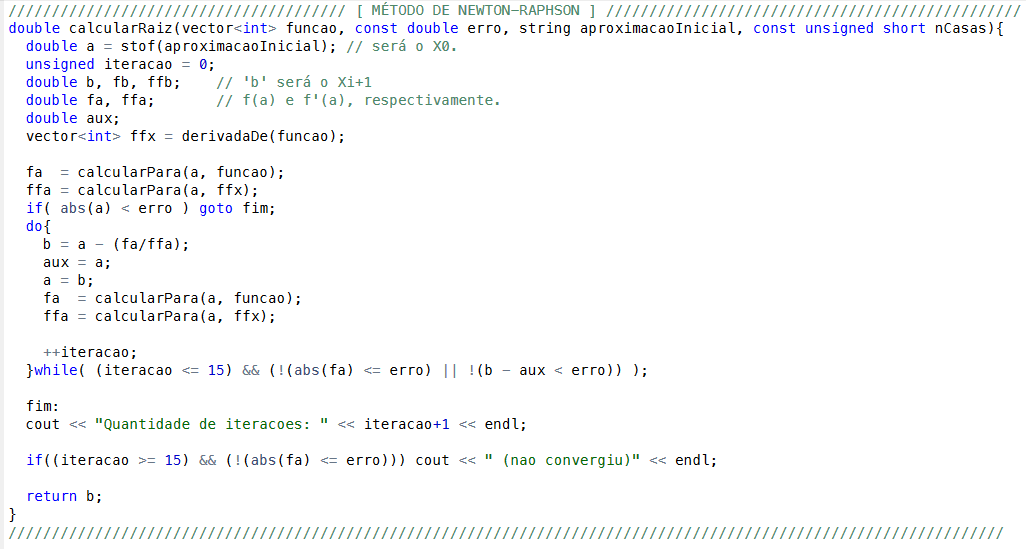
a = extremante esquerdo

e = erro

Com ele podemos iniciar o processo calculando *b*, que será o *a* da próxima iteração. Ou seja, *Xi = a* e *Xi+1 = b*, onde:

E agora temos duas condições de parada (em destaque ao lado).

O arquivo usarNewtonRaphson.hpp possui o seguinte trecho de código que implementa o algoritmo anterior:

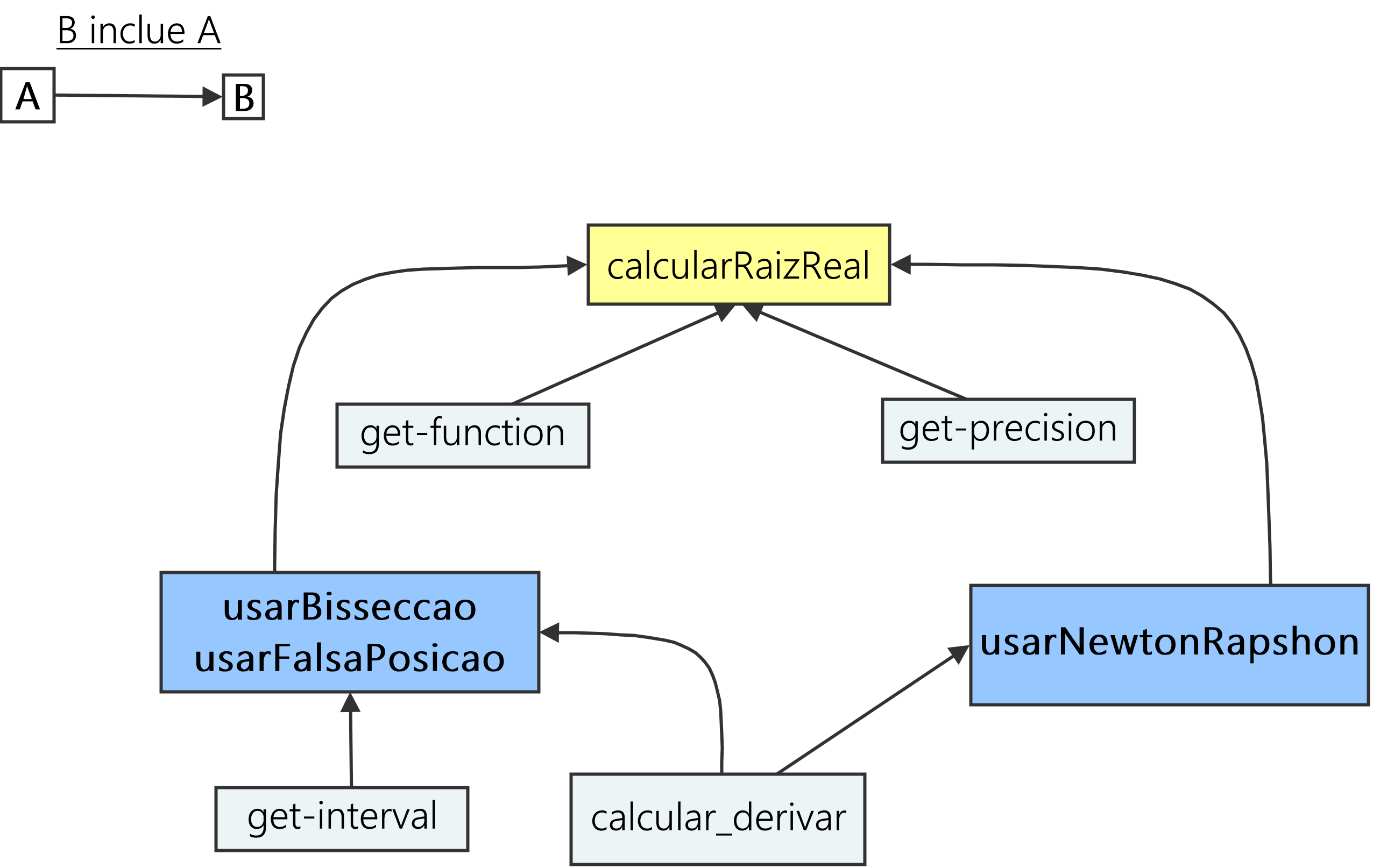


Esse método possui uma convergência mais rápida (em relação aos anteriores) porém não é garantida e necessidade do conhecimento da segunda derivada da equação.

**Implementação e execução do programa**

Antes de verificarmos a implementação da parte principal do programa, precisamos entender como os módulos se conectam (as implementações dos mesmos não serão mostradas aqui).

O código-fonte do programa principal (que deverá ser compilado) está no arquivo calcularRaizReal.cpp, os demais são módulos necessários para o funcionamento do programa e estão com extensão .hpp.



A seguir temos o trecho do código principal (arquivo calcularRaizReal.cpp) que deveremos manipular.



Antes de compilar, é preciso definir qual método será testado. Para tal, basta comentar duas das 3 linhas que importam os métodos que não usaremos. Na imagem acima vemos que a linha 35 importa o arquivo usarBisseccao.hpp que está na pasta bisseccao. Isso significa que ao compilar esse código-fonte iremos utilizar o método da Bissecção.

Para compilar afim de gerar o programa executável, precisamos executar na linha de comando (prompt de comando, Cygwin ou Terminal):

$ g++ -std=c++11 -o calcularRaizReal calcularRaizReal.cpp

E para enfim executar o programa temos dois formatos:

1. $ ./calcularRaizReal “função” “precisão” “intervalo fechado”
2. $ ./calcularRaizReal “função” “precisão” “aproximação inicial”

Ambos devem ser executados na linha de comando, porém o primeiro serve para programas que implementam o método da Bissecção ou o da Falsa Posição. Já o segundo é utilizado no método de Newton-Raphson.

**Testes Realizados**

Os dados de entrada dos testes realizados foram:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Função | Intervalo inicial | Aproximação inicial | Erro |
| f(x) | x^3 – 9x + 3 | [0, 1] | 1 | 0.001 |
| g(x) | x^3 – x – 4 | [0, 2] | 2 | 0.003 |
| h(x) | x^3 – 4x^2 + 2 | [-1, 0] | -1 | 0.001 |
| j(x) | x^2 – 2 | [1, 2] | 2 | 0.1 |
| k(x) | 3x^2 – 5x + 1 | [0, 1] | 2 | 0.02 |

Para utilizarmos corretamente o programa, basta seguir o formato exposto na tabela acima. Para testar os métodos da bissecção de da falsa posição devemos executar, por exemplo, o comando:

$ ./calcularRaizReal “x^3 – 9x + 3” “0.001” “[0,1]”

Para testar o método de Newton-Raphson e verificarmos o tempo de execução, utilizamos o comando:

$ time ./calcularRaizReal “x^3 – 9x + 3” “0.001” “1”

A quantidade de espaços não faz diferença, mas é necessário deixar cada argumento (função, precisão e intervalo) entre aspas duplas e a precisão (erro) deve conter ‘.’ (ponto) e o intervalo inicial deve conter os colchetes e a vírgula que definem os extremantes. Além disso, os expoentes devem ser números naturais e podem ou não ser precedidos de ‘^’ (circunflexo). Assim o programa exibirá uma tabela (de acordo com o método) e indicará a quantidade de iterações realizadas e a raiz encontrada.

**Resultados Obtidos**

A tabela abaixo mostra a relação do tempo de execução, em milissegundos, (média aritmética de 10 execuções) para cada teste realizado em cada método. Podemos ver que o método de Newton-Raphson obteve o melhor tempo em 80% dos casos. Nos testes abaixo, todos os métodos convergiram para algum valor aproximado de uma raiz real das equações fornecidas.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TEMPO POR INSTÂNCIA (milissegundos)** | | | | |
| **MÉTODO** | **f(x)** | **g(x)** | **h(x)** | **j(x)** | **k(x)** |
| **Bissecção** | 139 | 37 | 44 | 37 | 54 |
| **Falsa Posição** | 52 | 40 | 38 | 38 | 41 |
| **Newton-Raphson** | 134 | 36 | 37 | 37 | 40 |
| Método com menor tempo: | falsa posição | newton-raphson | newton-raphson | newton-rapshon | newton-raphson |

Podemos observar que diferença de tempo entre os métodos da Falsa Posição e de Newton-Raphson foi pouca para essas funções. Além disso, notamos que o método da Falsa Posição convergiu mais rápido que o da Bissecção (o que já era esperado).

A seguir temos o gráfico da tabela acima para melhor visualizar a discrepância de tempo entre os métodos.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

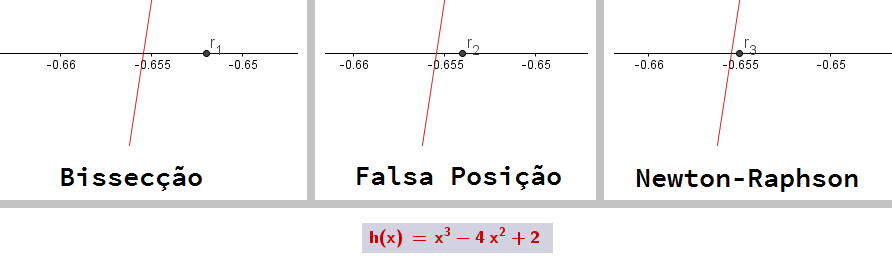
Observando agora o número de iterações realizadas por cada método para encontrar uma raiz real, novamente vemos que o método da Bissecção realiza mais iterações que os demais. Enquanto o método de Newton-Raphson conseguiu encontrar uma raiz em até ¼ da maior quantidade de iterações nos testes.

Agora veremos a precisão das raízes encontradas. A tabela a seguir mostra a raiz encontrada por cada método e compara com a raiz exata.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **RAÍZES ENCONTRADAS** | | | | |
| **MÉTODO** | **f(x)** | **g(x)** | **h(x)** | **j(x)** | **k(x)** |
| **Bissecção** | 0,340 | 1,789 | -0,652 | 1,410 | 0,227 |
| **Falsa Posição** | 0,339 | 1,795 | -0,654 | 1,400 | 0,234 |
| **Newton-Raphson** | 0,338 | 1,797 | -0,655 | 1,420 | 1,434 |
| Raiz Exata = | 0,338 | 1,796 | -0,655 | 1,414 | 0,232 |
| Maior Erro Absoluto = | 0,002 | 0,001 | 0,003 | 0,006 | 0,002 |

Assim podemos verificar que os métodos se aproximam de forma distinta. Por exemplo, na terceira coluna temos a função ***g***, no método da Falsa Posição o valor encontrado tendeu pela esquerda enquanto que no método de Newton-Raphson a raiz encontrada tendeu pela direita do valor exato. Porém, nos cinco casos testados, o método da Bissecção possui uma precisão menor que os demais, logo, não é o mais recomendado.

Para ver melhor como se comportam essas aproximações, vamos tomar a terceira instância (função ***h***) e plotar o gráfico da mesma:



Garantimos então que o último método dá uma precisão mais adequada em relação ao valor real exato da função (e não ao valor exposto na tabela anterior).

Vale ressaltar que na última instância os dois primeiros métodos se aproximaram da mesma raiz (*0.232*) enquanto o método de Newton-Raphson tendeu para uma outra raiz (visto que a função ***k*** tem grau 2). Isso acontece devido a forma de aproximação realizada pelos métodos.

**Conclusão**

Após analisar os resultados obtidos com as cinco instâncias expostas anteriormente, confirmamos que o método da Bissecção deve ser usado apenas para reduzir o intervalo que contém a raiz para enfim aplicarmos um outro método de refinamento que garanta uma melhor precisão. Apesar de não testarmos para polinômios com maior grau e um erro menor que *0.001*, precisamos admitir que o esforço computacional cresce demais quando aumentamos a exatidão com que se quer a raiz real.

Por melhor que seja o método de Newton, não devemos usá-lo para resolver equações cuja curva *y = f(x)*, próxima do ponto de intersecção com o eixo *x*, é quase horizontal. Devido a incapacidade de representação de um número tão grande que acontece no caso onde a derivada é igual a zero.