CCM0118 — Computação I

Curso de Ciências Moleculares — Turma 22 — Segundo Semestre de 2012

Segundo Exercício-Programa — Prazo de entrega: até **16 de outubro de 2012.**

Raízes de Equações Quadráticas

Escreva um programa em C que calcula as raízes de equações quadráticas. Seu programa deve ler um inteiro $n \geq 1$ e os coeficientes reais a, b e c de n equações do segundo grau ($ax^2 + bx + c = 0$, com $a \neq 0$). O programa deve calcular as raízes de cada equação e imprimir os resultados da maneira especificada a seguir.

Impressão dos resultados

Para cada equação deve-se imprimir o tipo de suas raízes (reais simples, real dupla ou complexas) e os valores das raízes. No caso de raízes complexas, deve-se imprimir a parte real e a parte imaginária.

O exemplo na página seguinte mostra como seu programa deverá funcionar. Nesse exemplo, os números que aparecem sublinhados foram digitados pelo usuário. Todo o resto é a saída do programa. Note que, para cada equação, os valores das raízes são impressos duas vêzes. Isto ocorre porque o programa implementa o cálculo de raízes de duas maneiras diferentes e apresenta os resultados obtidos com as duas implementações.

Cálculo das raízes de uma equação quadrática

Todos nós aprendemos a resolver equações quadráticas por meio da fórmula de Báskara:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Mesmo sendo matematicamente correta, essa fórmula nem sempre fornecerá resultados acurados se os números reais forem representados em ponto flutuante, como geralmente é o caso nos computadores. O problema conhecido como "cancelamento catastrófico" faz com que a tradução direta da fórmula de Báskara para uma linguagem de programação (como a linguagem C) não seja uma boa maneira de se calcular as raízes da equação quadrática.

O problema do cancelamento catastrófico se manifesta numa subtração de dois números reais (representados em ponto flutuante) próximos um do outro. Para entender exatamente que problema é esse, leia a seguinte entrada da Wikipedia: `http://en.wikipedia.org/wiki/Loss_of_significance`.

A fórmula de Báskara é suscetível a cancelamento catastrófico em dois pontos:

1. na soma dos termos $-b$ e $\sqrt{b^2 - 4ac}$ ou na subtração desses termos, dependendo do sinal de b, e
2. no cálculo do discriminante (a subtração $b^2 - 4ac$, dentro da raiz quadrada).

A primeira dessas possibilidades de cancelamento catastrófico pode ser evitada com um rearranjo da fórmula de Báskara. Basta calcular as raízes da maneira indicada na seção "*Floating-point implementation*" (`http://en.wikipedia.org/wiki/Quadratic_equation#Floating-point_implementation`) da

```
Digite o número de equações: 8
--- Equação 1 ---
a: 1
b: -2
c: -3
raizes simples: 3 e -1
               (3 e -1, pela fórmula de Báskara)
--- Equação 2 ---
a: 1
b: -2
c: 10
raizes complexas: 1+3i e 1-3i
                 (1+3i e 1-3i, pela fórmula de Báskara)
--- Equação 3 ---
a: 1
b: -2
c: 1
raiz dupla: 1
           (1, pela fórmula de Báskara)
--- Equação 4 ---
a: 7.18
b: 43.75
c: -31.21
raizes simples: 0.645079 e -6.73839
               (0.645079 e -6.73839, pela fórmula de Báskara)
--- Equação 5 ---
a: 3
b: -18
c: 27
raiz dupla: 3
           (3, pela fórmula de Báskara)
--- Equação 6 ---
a: 0
b: 10
c: 20
*** Erro: o coeficiente a não pode ser zero ***
--- Equação 7 ---
a: 1
b: 2
c: 3
raizes complexas: -1+1.41421i e -1-1.41421i
                 (-1+1.41421i e -1-1.41421i, pela fórmula de Báskara)
--- Equação 8 ---
a: 1
b: 0
c: -2
raizes simples: 1.41421 e -1.41421
               (1.41421 e -1.41421, pela fórmula de Báskara)
```

entrada da Wikipedia sobre a equação quadrática. É isso que você deve fazer neste EP. Visite também a página `http://introcs.cs.princeton.edu/java/91float/` e veja o exercício 17 ("*Quadratic formula*") da lista de "*Creative exercises*" ao final dessa página.

A segunda possibilidade de cancelamento catastrófico não pode ser evitada com um rearranjo da fórmula de Báskara. Felizmente é possível provar que, no pior caso, um cancelamento catastrófico na subtração $b^2 - 4ac$ pode acarretar a perda de no máximo a metade dos dígitos significativos do

resultado da subtração. Assim sendo, tal cancelamento não trará prejuízo algum (não tornará menos acurados os resultados finais) se no cálculo do discriminante $b^2 - 4ac$ usarmos o dobro dos dígitos significativos empregados no restante do programa. Neste EP você também deve tomar esse cuidado. Na maior parte do programa você trabalhará com `float`s. Os coeficientes de cada equação quadrática serão guardados em variáveis do tipo `float`. As raízes das equações quadráticas (ou a parte real e a parte imaginária de cada raiz, no caso de raízes complexas) também serão `float`s. No cálculo do discriminante, entretanto, deve ser usado o tipo `double`. Todas as operações aritméticas que o programa efetuará para calcular o discriminante (três multiplicações e uma subtração) devem ter operandos do tipo `double`.

Pelo que foi dito nos dois parágrafos anteriores, fica claro que seu programa deve implementar um "método cuidadoso" para calcular as raízes de cada equação. Esse método evita a primeira possibilidade de cancelamento catastrófico e se previne contra a eventualidade de perda de dígitos significativos em conseguência da segunda possibilidade de cancelamento catastrófico. Para efeito de comparação, seu programa deve implementar também um "método ingênuo" que simplesmente aplica a fórmula de Báskara, sem tomar nenhum cuidado com cancelamento catastrófico. Os valores das raízes de cada equação devem ser impressos duas vêzes — primeiro os valores obtidos pelo método cuidadoso, depois os obtidos pela aplicação direta da fórmula de Báskara.

Note que, em todas as equações do exemplo da página anterior, o método cuidadoso e a fórmula de Báskara produziram os mesmos resultados. Isso não vai acontecer sempre! Depois de ter seu programa em funcionamento, procure valores dos coeficientes a, b e c que façam os dois métodos produzirem resultados diferentes porque a primeira possibidade de cancelamento catastrófico se manifestou. Procure também valores desses coeficientes que façam os dois métodos produzirem resultados diferentes porque a segunda possibidade de cancelamento catastrófico se manifestou.

Teste de igualdade de números reais em ponto flutuante

Pelos motivos que já foram discutidos em classe, em geral não se deve usar "`==`" para comparar números reais em ponto flutuante:

```
float x, y;
...
if (x == y) { /* Não! */
    ...
}
```

Este fragmento de programa é ainda pior:

```
float x, y;
...
if (x - y == 0) { /* NÃO!!! */
    ...
}
```

O código acima subtrai os números que se deseja comparar e usa "`==`" para ver se o resultado é zero. Essa é uma péssima idéia, pois o valor comparado com zero é o resultado de uma subtração na qual pode ocorrido cancelamento catastrófico!

Em alguns (poucos) pontos do EP você precisará verificar se dois números em ponto flutuante são (aproximadamente) iguais. Para fazer isso, use a função `neary_equal`, cujo código é fornecido em seguida. Embora essa função receba parâmetros do tipo `double`, ela pode ser usada para fazer comparações de `float`s e para fazer comparações de `double`s.

```
#include <math.h> /* fabs */

typedef enum {FALSE, TRUE} boolean;

boolean nearly_equal(double x, double y, double min, double epsilon)
{
    double diff;

    if (x == y) {
        return TRUE;
    } else {
        diff = fabs(x - y);
        x = fabs(x);
        y = fabs(y);
        if (x == 0 || y == 0) { /* Relative error makes sense in this case?    */
            /* No, use absolute error */
            return diff < min / 2;
        } else {
            /* Yes, use relative error */
            return diff < epsilon * ((x < y) ? x : y);
        }
    }
}
```

Para comparar dois valores do tipo `float`, faça uma chamada a `nearly_equal` com os parâmetros `min` e `epsilon` iguais às constantes `FLT_MIN` e `FLT_EPSILON`:

```
float x, y;
...
if (nearly_equal(x, y, FLT_MIN, FLT_EPSILON)) {
    ...
}
```

Para comparar dois valores do tipo `double`, faça uma chamada a `nearly_equal` com os parâmetros `min` e `epsilon` iguais às constantes `DBL_MIN` e `DBL_EPSILON`:

```
double x, y;
...
if (nearly_equal(x, y, DBL_MIN, DBL_EPSILON)) {
    ...
}
```

As constantes `FLT_MIN`, `FLT_EPSILON`, `DBL_MIN` e `DBL_EPSILON` estão definidas em `<float.h>` e correspondem aos seguintes números:

`FLT_MIN` é o menor `float` positivo normalizado.

`FLT_EPSILON` é a distância de 1.0 ao menor `float` maior que 1.0.

`DBL_MIN` é o menor `double` positivo normalizado.

`DBL_EPSILON` é a distância de 1.0 ao menor `double` maior que 1.0.

Para mais informações sobre teste de igualdade em ponto flutuante, veja estas páginas:

`http://floating-point-gui.de/errors/comparison/` (contém informações básicas)

`http://www.altdevblogaday.com/2012/02/22/comparing-floating-point-numbers-2012-edition/` (faz uma discussão mais aprofundada)

Mesmo usando a função `nearly_equal`, você deve continuar evitando comparações com resultados de operações suscetíveis a cancelamento catastrófico. Neste EP, por exemplo, quando você quiser verificar se uma equação quadrática tem raiz dupla, chame `nearly_equal` para comparar b^2 e $4ac$ e não para comparar $b^2 - 4ac$ e zero!

Cálculo da raiz quadrada de um número real não negativo

Neste EP você não usará a função `sqrt(x)` da biblioteca `math.h`. Seu programa deve ter obrigatoriamente uma função com protótipo

```
double square_root(double x);
```

que calcula a raiz quadrada de `x` usando o método de Newton, descrito a seguir. Note que a função `square_root` recebe um argumento do tipo `double` e devolve um valor do tipo `double`. O cálculo da raiz quadrada deve ser feito com precisão dupla. (Além de usar precisão dupla no cálculo do discriminante, seu programa também trabalhará com `double`s quando estiver calculando uma raiz quadrada.)

O método de Newton. Suponha que desejamos extrair a raiz quadrada de um número real $x > 0$. Escolhe-se como chute inicial para $\sqrt{x}$ o número $r_0 = x$ e calcula-se a seguinte seqüência de números:

$$r_{n+1} = \frac{1}{2}\left(r_n + \frac{x}{r_n}\right) \qquad n = 0, 1, 2, \ldots$$

(Ou seja: obtemos

$$r_1 = \frac{1}{2}\left(r_0 + \frac{x}{r_0}\right) = \frac{1}{2}\left(x + \frac{x}{x}\right) = \frac{x+1}{2},$$

a partir de r_1 obtemos r_2, e assim por diante. É possível provar que essa sequência de números converge para $\sqrt{x}$.)

Esse processo deve ser repetido enquanto r_n e r_{n+1} não forem suficientemente próximos. A aproximação de $\sqrt{x}$ será o primeiro valor r_{n+1} que for suficientemente próximo de r_n. Como critério de proximidade, considere que dois valores `x` e `y` são suficientemente próximos se a chamada

```
nearly_equal(x, y, DBL_MIN, DBL_EPSILON)
```

devolver `TRUE`.

Questões que você deve responder

Coloque num comentário no início do programa, logo após o cabeçalho com seu nome e número USP, suas respostas para as seguintes questões:

(a) Para que valores dos coeficientes a, b e c os dois métodos (o "método cuidadoso" e a aplicação direta da fórmula de Báskara) produziram resultados diferentes porque a primeira possibidade de cancelamento catastrófico se manifestou? Tente achar valores de a, b e c que tornem grande a diferença relativa entre os resultados produzidos pelos dois métodos.

(b) Para que valores dos coeficientes a, b e c os dois métodos produziram resultados diferentes porque a segunda possibidade de cancelamento catastrófico se manifestou? Mais uma vez procure valores de a, b e c que maximizem a diferença relativa entre os resultados produzidos pelos dois métodos.

Bom trabalho!