MAC2166 Introdução à Computação - Grande área Elétrica

Escola Politécnica - Primeiro Semestre de 2015

Terceiro Exercício Programa **Entrega: até 24 de maio de 2015 às 23h55m**

Comportamento Dinâmico do Método de Newton

---

1. Objetivos

O objetivo deste terceiro exercício-programa é exercitar o uso de recursos de programação vistos na primeira e segunda partes da disciplina (ou seja, tudo o que está nos capítulos de 1 a 17 na **apostila do curso**). As **únicas construções** da linguagem C que você pode usar em seu programa são as dadas em aula. Acima de tudo, o objetivo de cada exercício, exercício-programa, aula e atividade de MAC2166 é desenvolver o **raciocínio aplicado na formulação e resolução de problemas computacionais**, como está descrito nos **objetivos** de MAC2166.

2. Introdução

2.1 O método de Newton

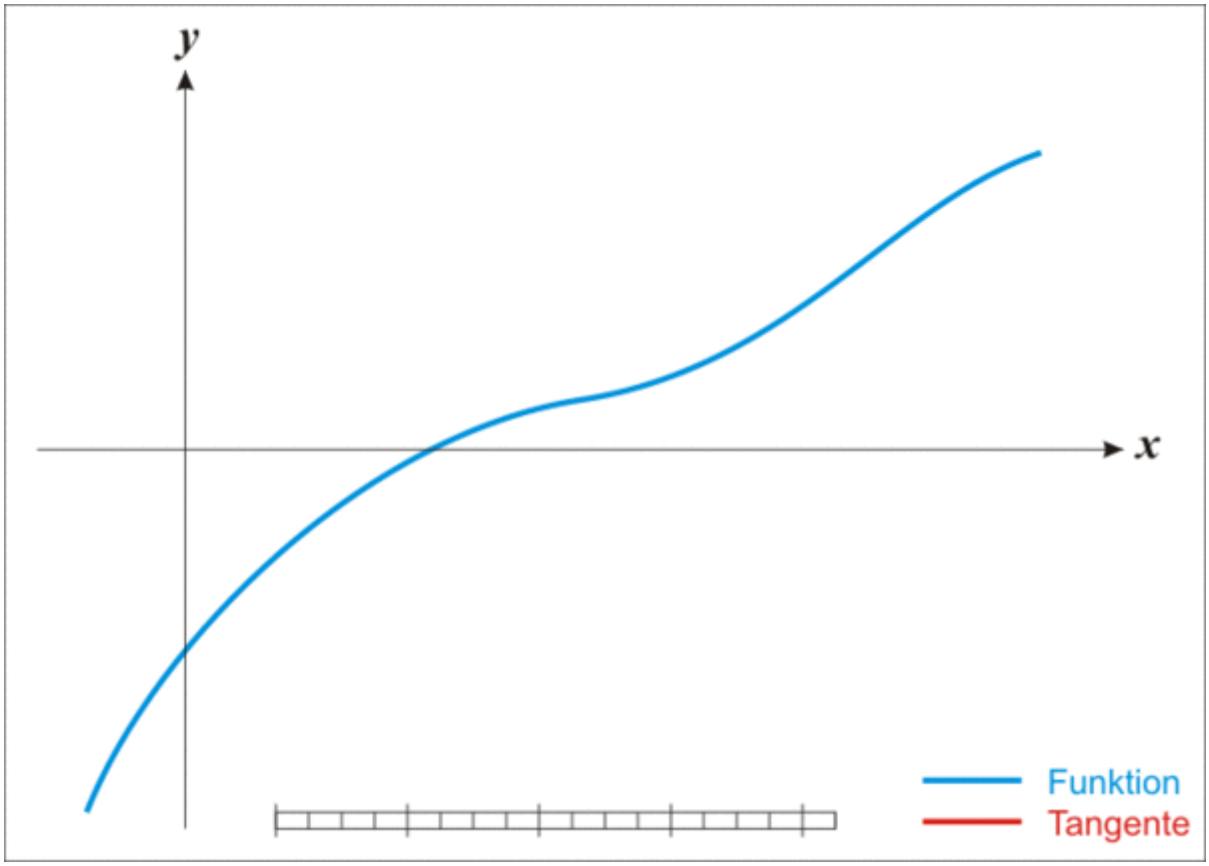
O método de Newton é um método numérico, muito simples, que pode ser usado para encontrar raízes de funções reais ou complexas das quais podemos calcular a primeira derivada.

Suponha que você tenha uma função real diferenciável f:R→R e que você queira encontrar uma raiz de f. O método de Newton pode ser usado para encontrar uma raiz da seguinte forma. Começamos com um chute inicial para a raiz que pode ser qualquer número, digamos x_0. Então calculamos iterativamente aproximações sucessivas para uma raiz de f gerando uma sequência de valores

$$x_{n+1} = x_n - f(x_n)\, f'(x_n)$$

para $n>0$.

Sob certas condições muitas vezes satisfeitas temos que o limite $\lim_{n\to\infty} x_n$ existe e que é igual a uma raiz de f. Não é difícil entender a ideia por trás do método de Newton. Começamos uma iteração com um chute para o valor de uma raiz. Calculamos a tangente da função no ponto atual (usando a derivada) e andamos pela tangente na direção do zero. A figura abaixo (roubada da Wikipedia) ilustra esse processo.



Para alguns valores iniciais, o método de Newton pode falhar em convergir para uma raiz. Por exemplo, se uma das derivadas que calculamos é igual a zero, não podemos fazer a divisão e o método falha.

2.2 Bacias de atração no plano complexo

Funções complexas também possuem derivadas e as regras de derivação que você aprendeu em cálculo são iguais para elas. Por exemplo a derivada do polinômio

$$(5+i)x^3 + 3x - (2-i)$$

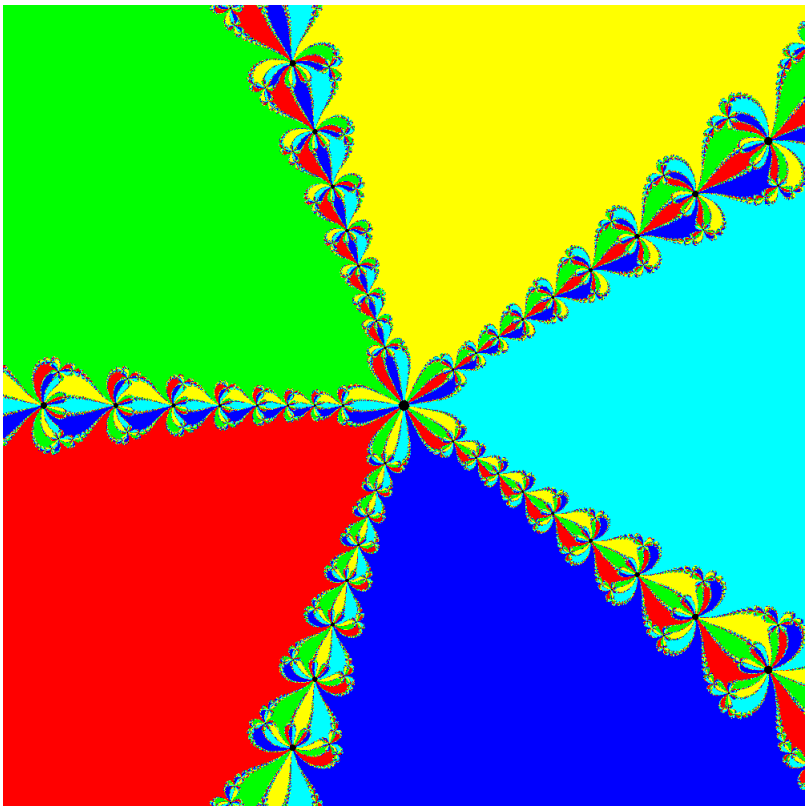
é

$$(15+3i)x^2+3.$$

O método de Newton, exatamente como explicado na seção anterior, também pode ser usado para encontrar raízes de funções complexas. Basta que possamos calcular a derivada da função.

Um fenômeno interessante ocorre quando executamos o método de Newton para encontrar raízes complexas de um polinômio. Suponha, por exemplo, que queremos encontrar raízes complexas do polinômio $p(x)=x^5-1$. Podemos executar o método de Newton para cada possível valor inicial $x_0 \in \mathbb{C}$. Este nos fornecerá (se convergir) uma raiz do polinômio p. Sabemos que p possui 5 raízes complexas. Para cada ponto inicial diferente para o qual o método converge seremos levados a uma dessas raízes. O interessante é ver quais pontos são levados a qual raiz.

Por exemplo, suponha que você execute o método de Newton para o polinômio p usando como pontos iniciais todos os pontos no quadrado $[-5,5]^2$ do plano complexo, ou seja, todos os números complexos da forma $a+bi$ com $a,b \in [-5,5]$. Para cada ponto inicial, chegamos numa raiz. Se atribuimos uma cor a cada raiz, podemos desenhar uma figura na qual cada ponto do quadrado recebe a cor da raiz à qual o método de Newton converge quando começa naquele ponto. Obtemos então a figura abaixo.



Um conjunto de pontos no plano complexo que são levados a uma dada raiz de *p* pelo método de Newton é uma *bacia de atração*. Logo, na figura acima, cada cor representa uma bacia de atração.

3. Tarefa

O objetivo deste exercício-programa é produzir figuras como a acima. **Você vai escrever dois programas**:

1. Um programa que, dada uma função C que calcula o valor de um polinômio complexo e o de sua derivada num determinado ponto, executa o método de Newton para uma grade fina de pontos dentro de um quadrado do plano complexo e grava os resultados obtidos num arquivo. A partir desses dados é possível gerar a figura das bacias de atração. **Mais detalhes sobre esse programa são fornecidos na Seçao 3.1.**
2. Um programa que, dado um polinômio complexo, gera uma função C capaz de calcular o valor do polinômio dado e o de sua derivada num determinado ponto. **Mais detalhes sobre esse programa são fornecidos na Seçao 3.2.**

3.1. Implementação do método de Newton

Você deve escrever um programa chamado `newton.c` que implementa o método de Newton para polinômios complexos e que o executa para muitos pontos num determinado quadrado do plano complexo.

A estrutura geral do seu programa deve ser a seguinte:

```
#include <stdio.h>



/*

  Prototipos

*/

(...)



/*

  Funcao que calcula o polinomio

*/

#include "polinomio.c"



/*

  Funcao main

*/

int main()

{

  (...)

}



/*

  Implementacao das funcoes

*/

(...)
```

O arquivo `polinomio.c` contém uma função de protótipo

```
void calcula(double x_real, double x_imag, double *p_real,
             double *p_imag, double *d_real, double *d_imag);
```

que recebe um número complexo x de parte real `x_real` e parte imaginária `x_imag` e calcula o valor de um polinômio e o de sua derivada em x. As partes real e imaginária do valor do polinômio em x são colocadas em `*p_real` e `*p_imag`, respectivamente. As partes real e imaginária do valor da derivada em x são colocadas em `*d_real` e `*d_imag`, respectivamente. A diretiva `include` que usamos acima apenas insere o conteúdo desse arquivo no arquivo `newton.c` quando este é compilado.

Seu programa deve pedir que o usuário digite os seguintes dados:

- Dois números reais, $l \leq u$;
- Um número inteiro $N \geq 1$;
- Um número inteiro $max\_iter \geq 1$;
- Um número real $\varepsilon > 0$.

Os parâmetros max_iter e ε fornecem um critério de parada para o método de Newton. O método de Newton apenas converge para uma raiz, não necessariamente a atinge, portanto precisamos decidir quando parar os cálculos. Você deve escrever uma função de protótipo:

```
  int newton(double *x_real, double *x_imag, int max_iter, double
eps);
```

Sua função recebe ponteiros `x_real` e `x_imag` para variáveis `double` que contêm as partes real e imaginária de um número complexo x, o número máximo de iterações `max_iter` e a precisão desejada `eps`. A função executa o método de Newton começando no ponto x, usando a função `calcula` para calcular o polinômio e sua derivada a cada iteração. Ao final, `*x_real` e `*x_imag` devem conter as partes real e imaginária do resultado.

A função `newton` deve adotar o seguinte critério de parada:

1. Você deve executar no máximo max_iter iterações do método de Newton;

2. No começo de cada iteração, calcula-se o valor do polinômio e o de sua derivada no ponto atual dado por `*x_real` e `*x_imag`;

3. Se o valor do polinômio for um número complexo $a+bi$ com $|a|\leq\varepsilon$ e $|b|\leq\varepsilon$, então o ponto atual está bem próximo de uma raiz. A função deve devolver um número NÃO-ZERO para indicar que teve sucesso;

4. Se o valor da derivada do polinômio for um número complexo $a+bi$ tal que $|a|\leq\varepsilon$ e $|b|\leq\varepsilon$, então o método falhou em encontrar uma raiz. A função deve devolver ZERO para indicar que falhou e os valores guardados em `*x_real` e `*x_imag` não importam;
5. Se não ocorreu nem o item 3 e nem o item 4, então atualiza-se o ponto atual usando-se a regra descrita na **Seção 2.1**;
6. Se você chegar ao número máximo de iterações sem ter chegado perto o bastante de uma raiz, ou seja, sem que o item 3 tenha ocorrido, então a função deve devolver ZERO para indicar que falhou e os valores guardados em `*x_real` e `*x_imag` não importam.

O seu programa vai executar o método de Newton para uma grade de pontos dentro do quadrado $[l,u]^2$ no plano complexo da seguinte forma. Você deve definir $\delta=(u-l)/N$ e considerar todos os números complexos da forma $x_{rs}=(l+r\delta)+(l+s\delta)i$ para $0\leq r,s\leq N$. Para cada ponto x_{rs}, você deverá executar o método de Newton usando-o como ponto inicial, obtendo uma aproximação y_{rs} de uma raiz do polinômio.

Você deve gravar essas aproximações num arquivo de texto chamado `saida.txt` da seguinte forma: a primeira linha do arquivo contém o número N. A r-ésima linha após a primeira, $r\geq 0$, contém os números y_{rs} para $0\leq s\leq N$, separados por espaços. Cada número complexo y_{rs} deve ser escrito como sua parte real separada da parte imaginária por um espaço. Se a função `newton` falhou em encontrar uma raiz para algum x_{rs}, então na saída você deve imprimir a letra `N` maiúscula.

Por exemplo, suponha que $N=2$ e que encontramos as raízes:

$$y_{00} \quad y_{01} \quad y_{02} \quad y_{10} \quad y_{11} \quad y_{12} \quad y_{20} \quad y_{21} \quad y_{22} =$$

$=$ $=$ $=$ $=$ $=$ $=$ $=$ $=$ i Newton falou!

5 $2.3+1.4i$ $-2.7+2.78i$ 0 $8.4-45i$ $-0.1-0.8i$ 0.7

Então o arquivo `saida.txt` seria assim:

```
2

0 1 N 5 0

2.3 1.4 -2.7 2.78 0 0

8.4 -45 -0.1 -0.8 0.7 0
```

Abaixo você encontra uma curta explicação de como ler e gravar arquivos de texto a partir de um programa C.

As mensagens exibidas pelo seu programa devem ser exatamente como as do exemplo a seguir. Os valores destacados em vermelho são digitados pelo usuário.

```
Digite o valor de l       : -5

Digite o valor de u       : 5

Digite o valor de N       : 1000

Digite o valor de max_iter: 5000

Digite o valor de epsilon : 1e-10
```

Os valores de *max_iter* e ε dados acima são bons para a geração das imagens, mas você pode brincar de fornecer outros valores e ver o que acontece.

O arquivo `saida.txt` não é uma imagem, mas pode ser processado de modo a gerar-se uma imagem. Para fazer isso, você pode usar o programa `fazimagem`, como explicado **aqui**.

Atenção: Aqui vão algumas observações importantes para que seus programas funcionem bem.

1. Você deve ter percebido que nos protótipos acima usamos sempre variáveis do tipo `double`. Neste exercício você deve usar *exclusivamente* variáveis do tipo `double` sempre que precisar armazenar números reais, pois é importante ter toda a precisão possível em seus cálculos.

2. Para ler um número de ponto flutuante fornecido pelo usuário e colocá-lo numa variável `double`, utilize o formato `"%lf"` com as funções `scanf` e `fscanf`, como no exemplo:

```
3.        double x;

4.        scanf("%lf", &x);
```

5. Para gravar o arquivo de saída, utilize o código de formato `"%.10g"` em vez de `"%lf"` para gravar o valor de variáveis do tipo `double`. O código `"%.10g"` imprime 10 dígitos significativos do número, o que será importante para a precisão do seu programa e para a geração da imagem.

3.2. Gerador de código

Para usar o programa `newton.c` da seção anterior você precisa de uma implementação da função `calcula` contida no arquivo `polinomio.c`. Para cada polinômio que você quiser testar, você pode reescrever o arquivo `polinomio.c` com a função apropriada. Isso é, entretanto, muito trabalhoso. Uma forma melhor seria escrever um programa que, dado um polinômio, *gera* o arquivo `polinomio.c` com uma função para calculá-lo. Depois de gerar a função, você pode compilar novamente o programa `newton.c` para gerar uma nova imagem.

Você deve escrever um programa chamado `gerador.c` que lê o arquivo `polinomio.txt` que contém o polinômio a ser processado. O polinômio é especificado no seguinte formato. A primeira linha do arquivo contém o grau n do polinômio. Cada uma das próximas $n+1$ linhas contém um coeficiente do polinômio. O primeiro coeficiente é o do monômio de grau n, o segundo o do monômio de grau $n-1$ e assim por diante até o último coeficiente que é o coeficiente do monômio de grau zero.

Cada coeficiente é um número complexo representado por dois reais a e b separados por um espaço. Esses números representam o número complexo $a+bi$.

Por exemplo, o arquivo

```
5

1 0

0 0
```

```
0 0

0 0

0 0

-1 0
```

representa o polinômio $x^5 - 1$. Já o arquivo

```
4

3.2 -1.5

0 1

2 0

1 1

0 0
```

representa o polinômio $(3.2-1.5i)x^4 + ix^3 + 2x^2 + (1+i)x$.

Seu programa deve gerar um arquivo `polinomio.c` que contém uma função C de protótipo

```
void calcula(double x_real, double x_imag, double *p_real,
             double *p_imag, double *d_real, double *d_imag);
```

Se p é o polinômio especificado no arquivo `polinomio.txt`, essa função recebe um número complexo $x = x\_real + x\_imag \cdot i$ e coloca em `*p_real` e `*p_imag` as partes real e imaginária de $p(x)$ e em `*d_real` e `*d_imag` as partes real e imaginária da derivada $p'(x)$.

Por exemplo, para o polinômio $(3+i)x^5 - 1$, o conteúdo do arquivo `polinomio.c` após a execução do seu programa poderia ser:

```
  void calcula(double x_real, double x_imag, double
*p_real,
```

```
                double *p_imag, double *d_real, double
*d_imag)

  {

    double a, b;



    *p_real = *p_imag = 0;

    *d_real = *d_imag = 0;



    potencia(&a, &b, x_real, x_imag, 5);

    multiplica(&a, &b, a, b, 3, 1);

    *p_real += a;

    *p_imag += b;

    potencia(&a, &b, x_real, x_imag, 4);

    multiplica(&a, &b, a, b, 15, 5);

    *d_real += a;

    *d_imag += b;

    potencia(&a, &b, x_real, x_imag, 0);

    multiplica(&a, &b, a, b, -1, 0);

    *p_real += a;

    *p_imag += b;

  }
```

Note que na função acima usamos duas outras funções, `potencia` e `multiplica`. Essas e outras funções devem estar definidas no programa `newton.c`. A função `potencia`, por exemplo, tem protótipo:

```
void potencia(double *ret_real, double *ret_imag, double x_real,
              double x_imag, int k);
```

Ela coloca em `*ret_real` e `*ret_imag` as partes real e imaginária da potência

$(x\_real+x\_imag \cdot i)^k$.

A função `multiplica` faz a multiplicação de dois números complexos. Seu protótipo é:

```
void multiplica(double *ret_real, double *ret_imag, double x_real,
                double x_imag, double y_real, double y_imag);
```

Ela coloca em `*ret_real` e `*ret_imag` as partes real e imaginária do produto

$(x\_real+x\_imag \cdot i)(y\_real+y\_imag \cdot i)$.

É claro que você pode usar outras estratégias na geração da função `calcula`. Você pode usar outras funções definidas em `newton.c`, por exemplo, ou fazer todas as contas diretamente na função `calcula`. Uma boa estratégia é, em vez de calcular as potências como acima, usar o **método de Horner**.

4. Orientações para a Implementação

4.1. Manipulação de arquivos

Você pode manipular arquivos usando funções parecidas com `printf` e `scanf`. Primeiro, entretanto, é preciso abrir o arquivo para leitura ou gravação. O programa abaixo deve ser suficiente para você pegar o jeito da coisa. Ele lê uma sequência de números reais positivos terminada por um número negativo de um arquivo chamado `entrada.txt` e grava o dobro de cada número no arquivo chamado `saida.txt`.

```
#include <stdio.h>



int main()

{

  double num;

  FILE *in, *out;



  in = fopen("entrada.txt", "r");  /* r = abrir para
leitura */
```

```
    if (!in) {

      printf("Nao consegui abrir o arquivo para
leitura!\n");

      return 0;

    }



    out = fopen("saida.txt", "w");  /* w = abrir para
gravacao */



    if (!out) {

      printf("Nao consegui abrir o arquivo para
gravacao!\n");

      return 0;

    }



    fscanf(in, "%lf", &num);

    while (num > 0) {

      fprintf(out, "O dobro e' %.10g\n", 2 * num);

      fscanf(in, "%lf", &num);

    }



    /* Fecha os arquivos */

    fclose(in);

    fclose(out);



    return 0;
```

```
}
```

É importante notar que temos que testar se os arquivos foram abertos com sucesso, como acima. Se ocorreu um problema, o programa deve ser encerrado. Também é preciso fechar os arquivos chamando a função `fclose` assim que eles não precisem mais ser usados. Fora isso, as funções `fscanf` e `fprintf` funcionam como `scanf` e `printf`, exceto que recebem também os arquivos nos quais devem trabalhar.

Para gravar números do tipo `double` neste EP, use o formato `"%.10g"` em vez de `"%lf"`. O formato `"%.10g"` decide automaticamente como gravar o número de modo a mostrar 10 dígitos significativos.

4.2. Método de Horner

Uma forma simples e eficiente de calcular um polinômio num dado ponto é através do *método de Horner*. Considere o polinômio $p(x)=x^5+2x^4+3x^3+4x^2+5x+6$. Podemos reescrevê-lo como:

$$p(x)=6+x(5+x(4+x(3+x(2+x)))).$$

Assim, uma forma de calculá-lo é a seguinte:

```
double x = 5;  /* Vamos calcular o polinomio em x */

double p = 0;  /* p guarda o resultado do calculo */



p = 1 + x * p;

p = 2 + x * p;

p = 3 + x * p;

p = 4 + x * p;

p = 5 + x * p;

p = 6 + x * p;
```

Essa fatoração e a forma de calcular o polinômio apresentada acima sugere um algoritmo, chamado de método de Horner. Você pode usá-lo na implementação deste EP se quiser.

5. Geração de Imagens

Depois que o seu programa `newton.c` gerou o arquivo `saida.txt`, você vai querer visualizá-lo como uma imagem. Para fazer isso, use o programa **`fazimagem.c`**. Você pode baixar o código fonte e compilá-lo, ou simplesmente baixar um dos executáveis dele disponíveis **aqui**.

Coloque o programa no mesmo diretório dos seus programas. Quando ele é executado, ele lê o arquivo `saida.txt` (que deve existir) e gera um arquivo chamado `imagem.ppm`. Este arquivo contém a imagem correspondente. Para ver arquivos do formato PPM você pode usar o visualizador padrão no Linux, ou usar o programa de manipulação de imagens **GIMP**, disponível para Linux, Windows e Mac OS X. O editor de texto Emacs também abre arquivos PPM e os exibe como imagens.

O programa `fazimagem` não é especialmente complicado. Ele lê a saída gerada pelo seu programa e tenta identificar as raízes diferentes que aparecem. A cada raiz diferente o programa atribui uma cor e depois gera uma imagem. Se o polinômio possui raízes muito próximas, o programa `fazimagem` pode se confundir e considerá-las idênticas. Também pode ocorrer o oposto: raízes que seriam idênticas são consideradas diferentes pois a precisão do método não foi grande o bastante. Esse tipo de erro de precisão é um problema sempre presente quando lidamos com números de ponto-flutuante.

Por isso, você pode controlar a precisão usada pelo programa para considerar se dois números `double` são iguais ou não. O programa considera que dois números reais x e y são iguais se $|x-y|<\varepsilon$. Por padrão temos $\varepsilon=10^{-7}$. Se você chamar o programa da linha de comando, pode especificar como primeiro argumento o valor de ε, como nos exemplos a seguir:

```
fazimagem 1e-4

fazimagem 1e-5
```

6. Observações

- Leia as **INFORMAÇÕES SOBRE ENTREGA DE EPs** antes de entregar o seu EP.
- **ATENÇÃO:** Neste EP, você deve entregar no portal Graúna dois arquivos com extensão ".c": um para o programa `newton` (`newton.c`) e outro para o programa `gerador` (`gerador.c`). Não se esqueça de incluir um cabecalho com o seu nome, número USP e turma nos dois arquivos a serem entregues.
- Certifique-se de que os dois arquivos ".c" com os códigos-fonte de seus programas foram realmente depositados no site verificando se você recebeu um e-mail com a confirmação da entrega.
- Executáveis deste exercício-programa estão disponíveis para donwload **aqui**. Caso você tenha dúvidas sobre qual deve ser o comportamento do seu programa em alguma situação, veja como se comporta o executável.