UNIVERSIDADE FEDERAL DO PARÁ
FACULDADE DE GEOFÍSICA

PROGRAMAÇÃO COMPUTACIONAL
Prof. Cícero Régis

1 Ordenamento de vetores

O arquivo `coordenadas.dat` contém três colunas contendo as coordenadas (x, y) de 500 pontos do gráfico de uma função, no seguinte padrão:

```
500
 1    5.9853573      -2.9344454
 2    3.4009480      -2.5645757
 3    5.7954922      -4.6858921
 4    4.4875660      -9.7483358
 5    2.2043452       8.0593157
 6    2.8838215       2.5492599
 7    0.4425255       4.2822309
 .      .               .
 .      .               .
 .      .               .
```

Os valores na primeira coluna são índices inteiros, na segunda e na terceira estão as coordenadas x e y, respectivamente. Acontece que os pontos não estão dispostos sequencialmente no arquivo. Eles foram embaralhados de forma aleatória.

Escreva um programa para realizar as seguintes tarefas:

1. Ler os dados do arquivo `coordenadas.dat`;
2. Criar um segundo arquivo contendo as mesmas coordenadas, agora ordenadas de acordo com os valores da coluna dos x.
3. Construa o gráfico da função $y = f(x)$.

2 Conversão entre radianos e graus

Escreva um programa que receba valores de ângulos expressos em radianos e os converta em graus, minutos e segundos. Escreva os resultados em um formato semelhante ao do exemplo:

```
Valor em radianos?
1.
1.0000000 rad = 57 graus, 17 min e 44.8223877 seg.
```

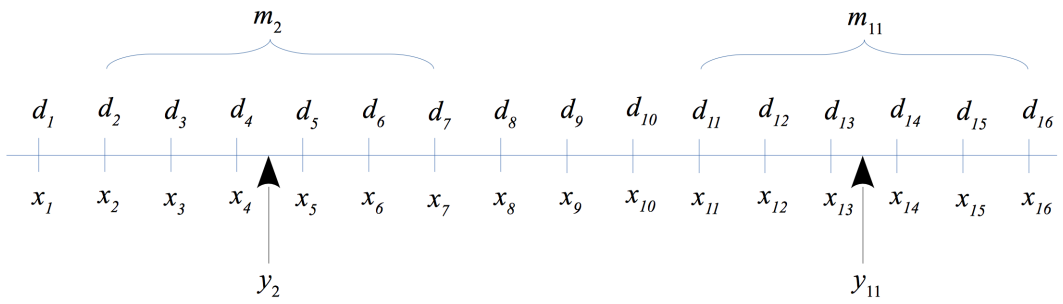
Repare que os valores em graus e minutos são números inteiros. Apenas os valores em segundos são expressos com variáveis reais.

3 Média móvel

Considere um vetor de dados d de tamanho total N, no qual os dados são contaminados com ruído aleatório de média zero. Cada valor deste vetor está associado a uma coordenada x. O processo de suavização conhecido como "média móvel" é uma espécie de filtragem dos dados pelo cálculo dos valores médios de uma sequência de sub-conjuntos do vetor. O processo consiste nos seguintes passsos:

1. definir o tamanho de uma janela w, ou seja, o número de elementos do sub-conjunto do vetor;
2. calcular a média m_1 dos valores abarcados pela janela começando da primeira posição do vetor original;
3. atribuir o valor da média calculado no passo 2 à coordenada (y_1) do centro do intervalo e salvar estes dois valores: o da média e o da coordenada;
4. repetir os passos 2 e 3 para o intervalo começando na segunda posição do vetor original, depois começando da terceira posição, e assim por diante até que a janela móvel atinja o fim do vetor;

Por exemplo, na figura estão ilustradas duas posições de uma janela w de largura 6, a primeira posição gera o segundo valor (m_2) em uma sequência de médias móveis, o qual é atribuído à coordenada y_2. A segunda posição da janela de filtragem mostrada na figura gera o décimo-primeiro valor de média móvel, gerando o par (y_{11}, m_{11}).



Nesta tarefa você irá programar a suavização pela média móvel e aplicar aos dados do arquivo `dados_ruido.dat`. Este arquivo contém duas colunas de valores. Na primeira estão as coordenadas x e na segunda estão os valores dos dados d contaminados com ruído, num total de 1257 pontos igualmente espaçados.

Para realizar a tarefa, escreva uma function para calcular a média de um dado vetor de dados reais. Esta function deve ter como entrada apenas o vetor de dados e o número de elementos do vetor.

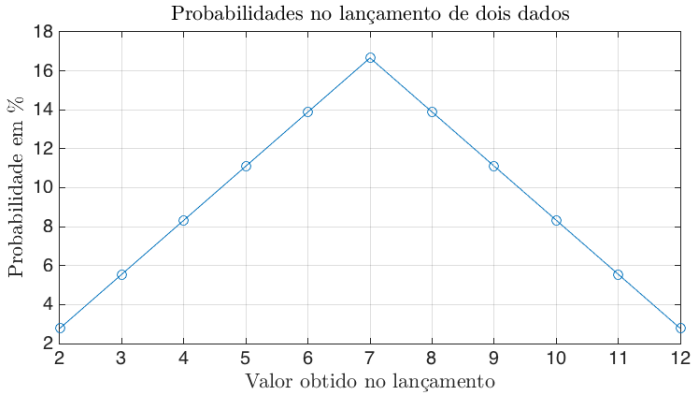
Então, escreva um programa para realizar os seguintes passos:

1. Ler do arquivo o vetor de dados de tamanho N;
2. Pedir do usuário um tamanho de janela w;
3. Executar um laço para calcular as médias m usando a function que você já construiu. Note que você passará para a function apenas a parte do vetor do tamanho da janela w, usando a notação vetorial `(valor_inicial : valor_final)`. Quantas voltas serão necessárias para este laço?
4. No mesmo laço calcule o valor da coordenada y do centro da janela;
5. Ainda no mesmo laço, escreva os valores das coordenadas y e das médias m num arquivo de saída.

Para observar o resultado, construa um gráfico mostrando os dados suavizados juntamente com os dados originais. Experimente com vários tamanhos de janelas e observe os resultados conseguidos. Como fica o gráfico se a janela for pequena demais e como fica se ela for demasiadamente grande?

4 Lançamentos de dados seguem uma distribuição gaussiana?

Se você lançar um dado muitas vezes, observará uma sequência de números aleatórios entre 1 e 6 com distribuição uniforme, ou seja, todo número tem igual probabilidade (16,667 %) de aparecer em cada lançamento. Já no lançamento de dois dados, os números possíveis são de 2 a 12, agora com probabilidades diferentes para cada número, como ilustrado na figura:

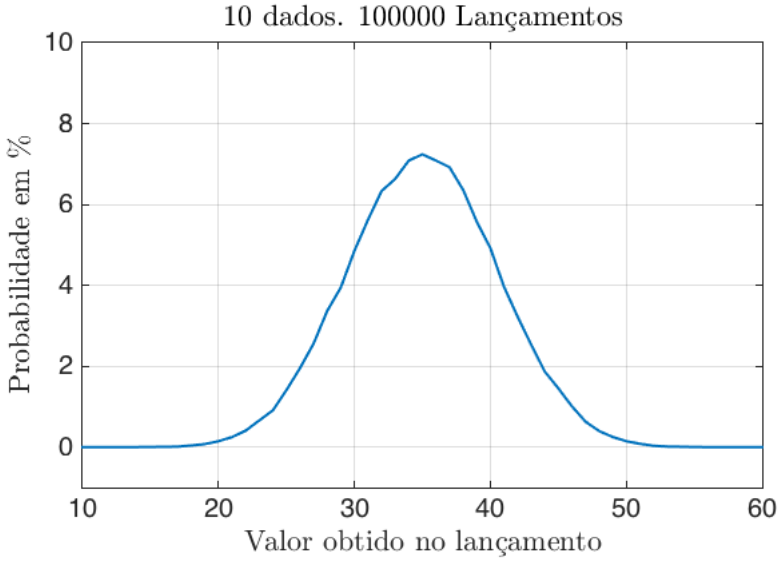


Com mais dados sendo lançados, as probabilidades passam a variar de maneira não linear, de modo que quanto maior o número de dados em um lançamento, mais a curva das probabilidades se assemelha a uma gaussiana.

Sua tarefa nesta questão é escrever um programa para verificar este comportamento:

1. Escreva uma subrotina que simule o lançamento de N dados. Para dados de 6 faces, os resultados possíveis são os números de N até $6N$.
2. Execute um grande número de lançamentos, registrando o número de vezes que cada número foi obtido, e ao final calcule a probabilidade atingida por cada valor.
3. Salve os resultados num arquivo para visualizar o gráfico.

Por exemplo, para 100 mil lançamentos de 10 dados, o resultado é o seguinte:



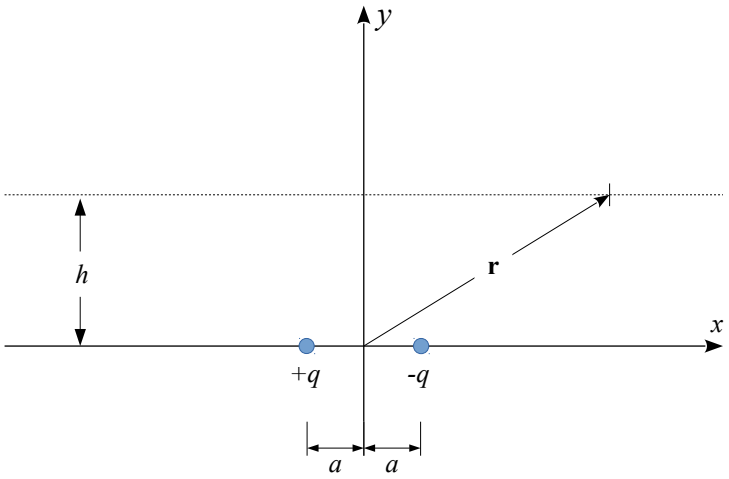
Como você faria para determinar se a distribuição é uma gaussiana verdadeira ou apenas aproximadamente?

Repare que neste problema implementamos o cálculo das probabilidades a partir dos resultados dos lançamentos, mas na verdade é possível calcular as probabilidades teóricas para o lançamento de qualquer número de dados.

5 Campo potencial de um dipolo elétrico

Neste problema você irá construir um programa para calcular o potencial elétrico U_B de uma distribuição de duas cargas elétricas puntiformes de sinais contrários, às vezes chamada de *bipolo*, e comparar com o potencial U_D de um dipolo elétrico pontual.

Considere um sistema formado por duas cargas $+q$ e $-q$, separadas pela distância $2a$, como mostrado na figura. Neste caso, o potencial elétrico em um ponto dado pelo vetor posição $\mathbf{r}$ é calculado como a soma dos dois potenciais devidos às duas cargas.



Agora, considere um dipolo pontual localizado na origem do sistema de coordenadas com o vetor momento de dipolo $\mathbf{p}$, horizontal, apontando para a esquerda e de igual valor ao das duas cargas. Neste caso, a expressão para o potencial do dipolo em uma posição dada pelo vetor $\mathbf{r}$ é

$$U_D = \frac{1}{4\pi\epsilon_0}\frac{\mathbf{p}\cdot\mathbf{r}}{r^3}, \qquad \mathbf{p} = -2aq\hat{\boldsymbol{\imath}}$$

Escreva um programa para calcular os valores de potencial elétrico gerados por estas destas duas fontes (dipolo e duas cargas opostas) em pontos sobre a reta horizontal na altura h indicada na figura.

Para visualizar o resultado, proceda da seguinte maneira: Mantendo fixo o valor de a ($a = 0.5$), execute seu programa para vários valores diferentes de h, por exemplo: $h = 10, h = 5, h = 1$, e construa os gráficos dos dois potenciais para cada valor de h. Faça os gráficos com um intervalo em x adequado para cada caso.

Se tudo der certo você irá observar que para valores grandes de h, as curvas dos dois potenciais coincidem, mas quando a linha de medidas se aproxima das fontes, o potencial do dipolo já não é mais igual àquele das duas cargas.

Tente determinar para qual valor de h, em termos da distância $d = 2a$ entre as cargas, a máxima diferença antes os dois potenciais é menor do que 1 %. Ou seja, determine quantas vezes o h precisa ser maior do que d para atingir esta aproximação. Refaça as medidas para diferentes valores de separação entre as cargas e verifique se a relação se mantém.

6 Aceleração da convergência na integração com trapézios

Neste problema você irá implementar um processo para acelerar a convergência da rotina de integração numérica usando trapézios, que você já programou.

O processo é baseado no fato de que o termo principal do erro cometido ao estimar a integral com N trapézios é uma função de $1/N^2$ e de termos subsequentes com expoentes pares de N. Isto sugere a seguinte análise: Suponha que avaliamos a integral com N intervalos, obtendo o resultado S_N, e novamente com $2N$ intervalos, obtendo o resultado S_{2N}. O termo principal do erro na segunda avaliação terá $1/4$ do valor daquele na primeira avaliação. Então a combinação

$$S = \frac{1}{3}\left(4S_{2N} - S_N\right)$$

irá cancelar o termo principal do erro. Como não há termo de erro de ordem $1/N^3$, o erro restante será de ordem $1/N^4$.

Altere sua subrotina de integração, aplicando a fórmula nos valores de duas avaliações seguidas e testando a convergência no resultado S. Verifique que agora são necessários menos intervalos para atingir a convergência desejada.

A análise que leva à formula acima pode ser encontrada em livros sobre métodos numéricos. Em particular ela está no excelente "Numerical Recipes: The Art of Scientific Programming", cujas edições antigas podem ser baixadas livremente na página

`http://numerical.recipes/com/storefront.html`

7 Função Erro

Para uma variável real x, a **função erro** muitas vezes representada por **erf** é definida como

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt.$$

Esta função tem várias aplicações. A mais comum é na estatística, onde (para $x > 0$) ela representa a probabilidade de uma variável aleatória com distribuição normal, média zero e variância $1/2$, pertencer ao intervalo $[-x, x]$.

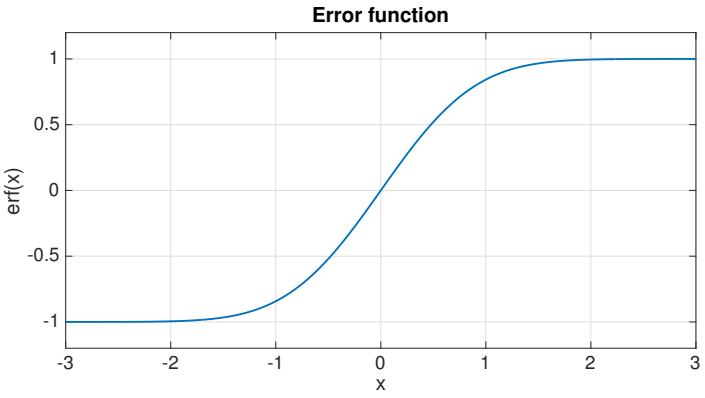
Sua tarefa é gerar valores da função $\text{erf}(x)$ e construir seu gráfico. Acontece que esta integral de aparência tão inocente na verdade não tem solução analítica em termos de funções elementares. Então, você terá que usar sua rotina de integração numérica para avaliar o valor da função para cada x.

Proceda de maneira semelhante a que usou para resolver a questão 7 da segunda lista:

- Crie uma `function` definida como o integrando;
- Construa um laço para variar os valores de x de -3 a 3, incluindo uma quantidade de pontos suficiente para o gráfico parecer uma curva suave;
- Para cada valor de x, calcule $\text{erf}(x)$ usando sua rotina de integração por trapézios, de preferência na versão com aceleração da convergência;
- Salve o resultado em um arquivo para poder construir o gráfico.

Use precisão dupla em todas as variáveis e constantes reais que façam parte tanto do programa principal quanto dos subprogramas.

Para verificar seu código, construa o gráfico dos valores calculados e compare com o da função `erf`, que faz parte tanto do Matlab quanto do Octave. O resultado deve ser assim:



8 De novo, integração numérica

A função $F(x)$ é definida através da seguinte integral:

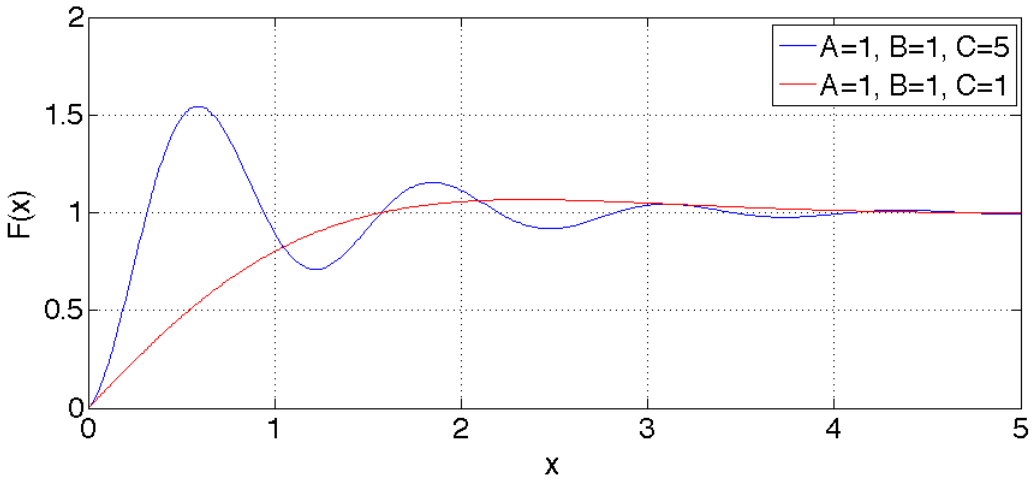
$$F(x) = \int_0^x A\, e^{-B\lambda} \left[B\cos(C\lambda) + C\sin(C\lambda)\right] d\lambda.$$

Sua tarefa nesta questão é construir gráficos da função F com diferentes valores dos parâmetros A, B e C, que estão listados no arquivo `parametros_ABC.dat`. Para isto, você deve construir um programa seguindo as seguintes instruções:

1. Escreva uma function com o integrando acima e tendo como único argumento a variável λ.
2. O programa principal lê os valores dos três parâmetros na primeira linha do arquivo.
3. Use sua rotina de integração numérica (na versão com a convergência mais rápida) para gerar valores da função $F(x)$ com os parâmetros lidos, com x variando de 0 a 5.
4. Repita os passos 2 e 3 para cada um dos conjuntos de parâmetros nas demais linhas do arquivo.
5. Ao final salve os resultados em um único arquivo organizado da seguinte maneira: na primeira coluna os valores da variável x, da segunda coluna em diante os valores de $F(x)$ calculados para cada conjunto de parâmetros.

Todas as variáveis reais devem ser de dupla precisão. A integral deve ser calculada com precisão de 10^{-6}.

Para verificar seus resultados, compare com os seguintes gráficos, obtidos usando dois dos conjuntos de parâmetros do arquivo:



9 Montagem de matrizes a partir de vetores

Uma imagem como esta

pode ser construída a partir da distribuição de intensidades de três cores básicas: vermelho (red), verde (green) e azul (blue). Um programa com ferramentas para exibir gráficos (MATLAB ou OCTAVE) pode ler a informação sobre as cores em três matrizes e compor a imagem. Neste problema você irá trabalhar com estas informações.

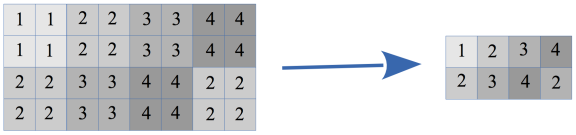
No arquivo `fontes.dat` está toda a informação para construir a imagem da foto. Na primeira linha do arquivo estão dois números inteiros (Nx, Ny) que correspondem às quantidades de pixels que formam a imagem, na horizontal e na vertical, nesta ordem. Em seguida vem uma única coluna de dados com 3*Nx*Ny números. Estes números correspondem aos elementos de três matrizes de tamanhos iguais, com as informações das três cores, na sequência red, green, blue. Os valores estão organizados por coluna, de maneira que os primeiros Ny números formam a primeira coluna da matriz Red, os próximos Ny valores formam a segunda coluna da mesma matriz e assim por diante. Na sequência vem as colunas da matriz Green e depois as da Blue. Seu programa deve dimensionar as matrizes a partir da leitura dos valores de Nx e Ny.

Você deve construir um programa para ler a informação contida no arquivo `fontes.dat` e a partir dela montar três matrizes para serem importadas para o MATLAB ou OCTAVE para exibir a imagem. Salve as matrizes em três arquivos chamados `Red.dat, Green.dat e Blue.dat`. Depois, para visualizar a imagem final, simplesmente use o script `monta_imagem.m` dentro da mesma pasta onde salvou os arquivos.

10 Compressão de arquivo de imagem com perdas

No arquivo `Beni.dat` está toda a informação para construir a imagem abaixo, organizada no arquivo exatamente da mesma maneira que foi descrita na questão 9.

A imagem tem 2.359.296 pixels dispostos em 1.152 linhas e 2.048 colunas, o que resulta num arquivo ASCII de 127,4 MB. O objetivo desta tarefa é reduzir o número de pixels da imagem, de maneira a gerar matrizes menores. Uma maneira simples de criar uma imagem com um quarto do número de pixels (perdendo bastante informação!) é construir matrizes das intensidades de cores de modo que cada pixel seja a média de 4 pixels da imagem original. Por exemplo, se a imagem tivesse apenas 4 por 8 pixels, cada matriz reduzida seria 2 por 4, sendo que cada elemento desta seria a média de 4 elementos da matriz original:



Para gerar as matrizes reduzidas, você deve construir um programa para realizar os seguintes passos:

1. Ler a informação contida no arquivo `Beni.dat` e a partir dela montar três matrizes, R, G e B, com o tamanho original;
2. Dimensionar outras três matrizes (R2, G2, B2) com metade do número de linhas e metade do número de colunas das matrizes originais;
3. Calcular o valor de cada elemento das matrizes reduzidas como a média dos valores de 4 elementos adjacentes das matrizes originais;
4. Salvar as matrizes em seis arquivos chamados `Red.dat, Green.dat, Blue.dat,` com os valores originais, e `Red2.dat, Green2.dat, Blue2.dat,` com os valores das matrizes reduzidas.

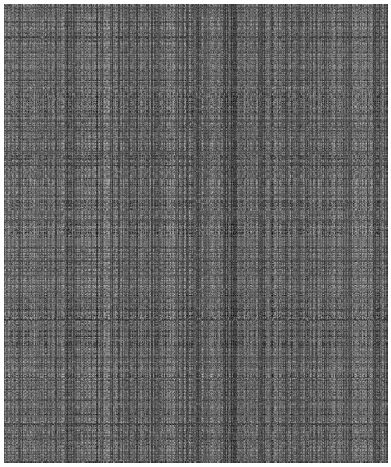
Você pode considerar que os números de linhas e de colunas são sempre pares.

Ao final, para visualizar e comparar as imagens use o script `monta_imagem_Beni.m` na mesma pasta onde estão salvas os arquivos com as matrizes. Verifique as diferenças nas imagens quando você dá (*zoom*) nas duas. Verifique também as diferenças nos tamanhos dos arquivos com as matrizes originais e as reduzidas.

- Diversão extra:
Este problema pode ser resolvido de forma mais econômica, sem alocar memória para as matrizes completas. Tente construir uma segunda versão de seu programa que utilize o mínimo possível de memória.

11 Mais uma de montar imagem

O arquivo `Lara_Croft.dat` contem uma matriz de números inteiros com 787 linhas e 658 colunas que deveria formar uma imagem em tons de cinza. Entretanto, o arquivo foi alterado de modo que se você usar o script `Mostra_Imagem_LC.m` verá apenas isto:



Para a imagem ficar assim, as colunas e as linhas da matriz original foram embaralhadas aleatoriamente. Felizmente, a informação sobre a ordem correta das colunas e linhas está salva nos arquivos criativamente chamados `Linhas.dat` e `Colunas.dat`.

Sua tarefa é escrever um programa para ler os arquivos e colocar cada elemento da matriz em sua posição correta, de modo a formar a imagem original. Seu programa deve gerar um novo arquivo com a matriz corrigida para ser carregado para o MATLAB ou o OCTAVE.

Perceba que há duas interpretações possíveis para a informação contida nestes arquivos. Ou o elemento na posição `(i,j)` da matriz embaralhada deve ir para a posição indicada pelos números `(Linhas(i),Colunas(j))` na matriz corrigida; ou o elemento na posição `(Linhas(i),Colunas(j))` da matriz embaralhada deve ir para a posição `(i,j)` da matriz corrigida. Como não sabemos qual das duas possibilidades é a correta, você deve testar as duas e verificar.

Use o script `Mostra_Imagem_LC.m`, substituindo o nome do arquivo no comando `load` pelo arquivo que você criou. Se tudo der certo você verá uma bela imagem da Lara Croft.

12 Equação de Laplace com discretização retangulares

Você já construiu o código para encontrar a solução numérica da equação de Laplace 2D pelo método de relaxação (Gauss-Seidel) para uma discretização com subintervalos iguais nas direções dos eixos x e y. Porém, quando o domínio bidimensional é um retângulo, nem sempre desejamos trabalhar com passos iguais nas duas direções. Neste caso, vamos fazer o cálculo de maneira um pouco diferente.

Sendo Δx e Δy os passos nas direções x e y, as aproximações de diferenças finitas para as segundas derivadas da função $\phi(x, y)$ no nó (i, j) são

$$\frac{\partial^2 \phi_{(i,j)}}{\partial x^2} \approx \frac{\phi_{(i,j-1)} - 2\phi_{(i,j)} + \phi_{(i,j+1)}}{\Delta x^2}, \qquad \frac{\partial^2 \phi_{(i,j)}}{\partial y^2} \approx \frac{\phi_{(i-1,j)} - 2\phi_{(i,j)} + \phi_{(i+1,j)}}{\Delta y^2}.$$

Então, aplicando na equação de Laplace,

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0,$$

$$\frac{\phi_{(i,j-1)} - 2\phi_{(i,j)} + \phi_{(i,j+1)}}{\Delta x^2} + \frac{\phi_{(i-1,j)} - 2\phi_{(i,j)} + \phi_{(i+1,j)}}{\Delta y^2} = 0.$$

A *razão de aspecto* β, para domínios retangulares, é

$$\beta = \frac{\Delta x}{\Delta y}.$$

Então, a iteração fica

$$\phi_{(i,j)} = \frac{\phi_{(i,j-1)} + \phi_{(i,j+1)} + \beta^2 \left[\phi_{i-1,j} + \phi_{(i+1,j)}\right]}{2(1 + \beta^2)}.$$

Quando o domínio é quadrado, $\beta = 1$ e a expressão se reduz à que você já programou.

Agora, faça uma nova versão de seu programa na qual o tamanho dos lados do retângulo (ou as coordenadas dos vértices) sejam dados de entrada e implemente as iterações usando a razão de aspecto de acordo com a equação.

Os resultados mais acurados são obtidos com $\beta = 1$. Por isso, tome cuidado para que os incrementos nas coordenadas (Δx e Δy) não sejam muito diferentes entre si. Por exemplo, se o domínio retangular tem um lado de comprimento igual ao dobro do outro, é possível discretizar o retângulo com incrementos iguais $\Delta x = \Delta y$, o que é preferível.

Para verificar a diferença nas soluções, defina o domínio do problema como um retângulo de lados $L_x = 2, L_y = 1$. Neste caso, se você fizer o mesmo número de células nas duas direções, terá $\Delta x = 2\Delta y$, ou $\beta = 2$. Se, por outro lado, discretizar o lado L_x com o dobro de células do L_y, terá $\beta = 1$. Rode o programa com estas duas discretizações e compare as soluções observando se há diferença nas respostas. Procure no software gráfico quais os comandos para mostrar as linhas de contorno das duas na mesma figura, uma com linhas contínuas e a outra com linhas tracejadas, por exemplo.

13 Aceleração de convergência para a solução da equação de Laplace

A taxa de convergência do método de relaxação pode ser aumentada através do método de sobre-relaxação SOR (Successive-Over-Relaxation). Funciona assim: ao invés de simplesmente substituir o valor em cada nó pela expressão dada na equação[1]

$$\phi_{(i,j)} = \frac{\phi_{(i,j-1)} + \phi_{(i,j+1)} + \beta^2 \left[\phi_{i-1,j} + \phi_{(i+1,j)}\right]}{2(1+\beta^2)},$$

defina o termo $\Delta\phi_{(i,j)}$:

$$\Delta\phi_{(i,j)} = \frac{\phi_{(i,j-1)} + \phi_{(i,j+1)} + \beta^2 \left[\phi_{i-1,j} + \phi_{(i+1,j)}\right]}{2(1+\beta^2)} - \phi_{(i,j)}.$$

Então, a iteração tradicional do método de relaxação que você já programou é

$$\phi_{(i,j)} = \phi_{(i,j)} + \Delta\phi_{(i,j)}.$$

A convergência é acelerada se você alterar a iteração através do fator de sobre-relaxação ω:

$$\phi_{(i,j)} = \phi_{(i,j)} + \omega\Delta\phi_{(i,j)}.$$

A máxima taxa de convergência é obtida para um valor ótimo ω_{opt}, entre 1.0 e 2.0.

Em alguns casos especiais, o fator de sobre-relaxação ótimo pode ser determinado teoricamente. Para uma região retangular ω_{opt} pode ser estimado assim:

$$\omega_{\text{opt}} = 2\left(\frac{1-\sqrt{1-\epsilon}}{\epsilon}\right),$$

em que

$$\epsilon = \left[\frac{\cos(\pi/N_x) + \beta^2\cos(\pi/N_y)}{1+\beta^2}\right]^2,$$

e N_x e N_y são os números de células em cada direção.

Crie uma nova versão de seu programa, implementando a sobre-relaxação com o parâmetro ω calculado a partir dos dados de entrada. Compare a solução com a anterior, verificando o número de iterações realizadas.

Todos os detalhes sobre as soluções numéricas para as equações de Laplace e Poisson podem ser encontrados em vários livros sobre métodos numéricos, como no já citado "Numerical Recipes", ou por exemplo, o livro de Joe D. Hoffman, "Numerical Methods for Engineers and Scientists". O que nós vimos nesta matéria foi só uma parte do assunto, que tem vários outros aspectos para ser estudados, incluindo muitos detalhes importantes para as aplicações de Diferenças Finitas.

[1] Repare que a notação que eu usei nestes problemas não está incluindo um índice para indicar em qual iteração está cada número. A rigor, na equação da iteração existem dois valores de ϕ calculados na iteração k e dois calculados na $k+1$. Como está, a notação é mais próxima da linguagem de programação do que da matemática mesmo. Tome cuidado!

14 Crivo de Eratóstenes.

Este problema trata de um método para encontrar números primos. Funciona da seguinte maneira: crie um vetor de números inteiros com todos os elementos iniciados com 1. Elementos cujas posições no vetor são números primos permanecerão valendo 1. Todos os demais elementos serão convertidos para zero.

Iniciando na segunda posição, toda vez que um elemento for encontrado cujo valor seja 1, percorra o vetor até o final e mude para zero todos os valores cujas posições sejam múltiplos daquela do elemento de valor 1. Quando esse processo estiver completo, os elementos do vetor que ainda valem 1 indicam que suas posições são números primos. Então, imprima essas posições.

Por exemplo, o vetor no qual foram zerados todos os valores de índices pares maiores que 2 é
$[0, 1, 1, 0, \ldots]$.
Depois da segunda posição, o próximo elemento que tem valor 1 é o terceiro, então devemos zerar todos os valores seguintes cujos índices sejam múltiplos de 3:
$[0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, \ldots]$.
Depois da posição 3, o próximo valor não nulo está na posição 5, então devemos zerar todos os elementos seguintes cujas posições são múltiplas de 5:
$[0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, \ldots]$.
Repare que os índices múltiplos de 5 que também eram de 2 ou de 3 já estavam zerados.

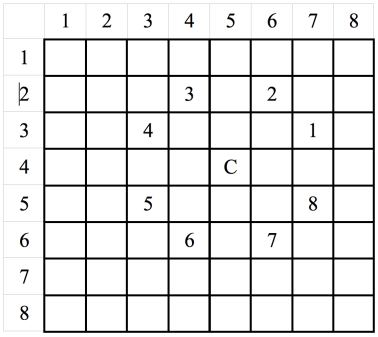
Teste seu programa para encontrar os números primos até 1000. Otimize ao máximo o programa e tente determinar quantos números primos existem até 1 milhão.

Verifique se há diferença de desempenho entre usar laços DO ou operações vetoriais para zerar os valores.

15 Passeio do cavalo

Um interessante quebra-cabeças para aficcionados do jogo de xadrez é o problema chamado de passeio do cavalo. A questão é esta: pode o cavalo se mover em um tabuleiro vazio e tocar cada uma das 64 casas uma única vez? Para analisar este problema, vamos escrever um programa para realizar os movimentos do cavalo no tabuleiro.

O cavalo faz movimentos em forma de L (andando duas casas em uma direção e então uma casa em uma direção perpendicular). Então, de uma casa no meio de um tabuleiro vazio, o cavalo pode fazer oito movimentos diferentes como mostrado na figura:



Em seu programa, o tabuleiro é representado por uma matriz 8 por 8, do tipo `integer`, chamada `tab` que deve ser iniciada com zeros.

Descreva cada um dos 8 movimentos possíveis em termos de suas componentes horizontal e vertical. Por exemplo, um movimento do tipo 1 como mostrado na figura consiste em mover duas casas horizontalmente para a direita e uma casa verticalmente para cima. Já o movimento 6 consiste em mover uma casa horizontalmente para a esquerda e duas casas verticalmente para baixo. Os oito movimentos podem ser descritos por dois vetores, h e v, de números inteiros, com os movimentos para a direita e para baixo representados por números positivos e aqueles para esquerda e para cima, por números negativos, da seguinte maneira:

h(1) = 2	v(1) = -1
h(2) = 1	v(2) = -2
h(3) = -1	v(3) = -2
h(4) = -2	v(4) = -1
h(5) = -2	v(5) = 1
h(6) = -1	v(6) = 2
h(7) = 1	v(7) = 2
h(8) = 2	v(8) = 1

Crie duas variáveis, `L` e `C`, para indicar a posição do cavalo na matriz em cada momento. A partir da posição `(L,C)`, para fazer um movimento do tipo N, em que N é um número entre 1 e 8, seu programa pode usar os comandos

```
L = L + v(N)
C = C + h(N)
```

Iniciando em uma posição `(L,C)` dada pelo usuário, escolha aleatoriamente os próximos passos do cavalo. Crie um contador que varia de 1 a 64. Registre na matriz `tab` o valor corrente do contador em cada casa em que o cavalo chegar. Em cada passo, ao gerar uma nova posição é preciso verificar se o cavalo já passou por aquela casa, e, claro, para se certificar de que ele não vai cair fora do tabuleiro. Caso encontre alguma destas situações, o programa deve realizar um movimento diferente. Caso nenhum dos oito movimentos seja possível, o programa para e exibe na tela a matriz com os passos realizados.

Inicie o movimento a partir de casas diferentes e veja quantos passos são dados em cada vez. Você consegue algum passeio completo, ou seja, o cavalo tocar em cada uma das 64 casas sem repetir nenhuma?

Após analisar vários passeios do cavalo com seu programa, você pode perceber que as casas nas bordas são mais problemáticas do que as casas próximas ao centro do tabuleiro. De fato, as mais problemáticas, ou de difícil acesso, são os 4 vértices.

Tente, então, mover o cavalo para as casas mais complicadas primeiro e deixar aquelas de acesso mais fácil para o final, quando o tabuleiro vai ficando mais congestionado.

Escreva uma segunda versão do programa, na qual você vai classificar cada uma das casas de acordo com a facilidade de acesso e mover o cavalo sempre para a casa de menor acessibilidade. Crie uma matriz 8×8 chamada `acesso`, na qual cada elemento indica o número de casas a partir das quais cada casa pode ser atingida. Em um tabuleiro vazio, a matriz `acesso` terá os valores:

2	3	4	4	4	4	3	2
3	4	6	6	6	6	4	3
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
3	4	6	6	6	6	4	3
2	3	4	4	4	4	3	2

Agora, em seu programa, ao invés de andar aleatoriamente, o cavalo deve sempre se mover para a casa com o menor número de acessibilidade. Em caso de empate, o cavalo pode mover-se para qualquer uma das casas empatadas. Enquanto o cavalo se movimenta, seu programa deve diminuir os números de acessibilidade, conforme mais e mais casas ficam ocupadas. Dessa forma, em qualquer instante, o número de acessibilidade para cada casa vai sempre ser exatamente o número de casas a partir das quais ela pode ser atingida.

Execute seu programa 64 vezes, iniciando em cada uma das casas. Quantos passeios completos você obteve?