Lista de exercícios de CM114

Abel Soares Siqueira

Última revisão: 12 de Ago de 2015

1 Preparatórios

Use os exercícios a seguir para se familiarizar com a linguagem Julia e com algumas ideias matemáticas. Em quase todos os exercícios **você deverá criar uma função**, e não apenas criar um arquivo com o código dentro.

Use discernimento e bom senso para decidir que funções prontas (já implementadas) você pode utilizar no exercício. Ex.: Se o exercício é para calcular o módulo de um número, você não deve usar a função `abs`. Se o exercício é para calcular a norma-1 de um vetor, você pode usar a função `abs`.

Além de resolver o exercício pedido, tente melhorar a sua solução tentando considerando o número de variáveis usadas e as alocações e cálculos desnecessários.

Em alguns exercícios, pede-se que você gere um erro. A maneira mais simples é usando a função `error` que recebe uma mensagem de texto e para completamente a execução do programa. Teste no terminal do Julia.

Quando for pedido que você faça uma comparação de tempo, utilize o pacote `TimeIt`. Para instalá-lo, no terminal do Julia digite `Pkg.add("TimeIt")`. Para usá-lo, antes do comando que você for usar, escreva `@timeit`. Além dessa comparação, você também pode comparar a memória utilizada, o número de iterações, de variáveis, etc..

1. Implemente as seguintes funções:

   - `modulo(x)` que retorna o módulo do número x.
   - `sinal(x)` que retorna 1 se x é positivo, 0 se x é zero, e -1 se x é negativo.
   - `ehpar(x)` que retorna `true` ou `false`, se x é par ou não, respectivamente. (Use `%`)
   - `ehprimo(x)` que retorna `true` ou `false`, se x é primo ou não, respectivamente.
   - `norma(x,p)` que calcula a norma-p de x, onde $1 \leq p < \infty$. A norma-p é definida por

   $$\|x\|_p = \left[\sum_{i=1}^{n} |x_i|^p \right]^{1/p}$$

   - `normainf(x)` que calcula a norma-∞ de x. A norma-∞ é definida por

   $$\|x\|_\infty = \max_{1 \leq i \leq n} |x_i|$$

2. Implemente as funções `mmc`, `mdc` que calculam o Mínimo Múltiplo Comum e o Máximo Divisor Comum. Você pode fazer como achar melhor, mas uma alternativa rápida é o Algoritmo de Euclides.

3. Implemente as funções `dec2bin`, `bin2dec` que fazer a conversão de decimal para binário e binário para decimal, respectivamente. `dec2bin` recebe um número e deve retornar uma String (cadeia de caracteres). O segundo recebe uma string e retorna um número. Os números são todos inteiros.

4. Implemente uma função que calcula o n-ésimo termo da progressão de Fibonacci. A progressão de Fibonacci segue a fórmula $F_1 = F_2 = 1$ e $F_n = F_{n-1} + F_{n-2}$. **Atenção,** existem duas maneiras de fazer essa função:

   - Usando for e vetor (Array);
   - Em uma linha usando recursão. É uma maneira pouco eficiente. Procure por operator ternário.

   Discuta porque recursão não é eficiente neste caso.

5. O número ε é dito **precisão da máquina** ou **precisão do tipo X**, onde X é um tipo de ponto flutuante se o cálculo de $1+\varepsilon$ resulta em 1 (este 1 sendo do tipo X). Faça um programa que calcula a precisão da máquina para o tipo `Float64` em Julia.

2 Problemas com um pouco (mais) de matemática

Os problemas anteriores servem como aquecimento. Estes já tem um nível de matemática maior envolvido.

1. Faça uma função que calcula a exponencial de um número x usando a expansão de Taylor em torno da origem:
$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!} + \dots$$
   Não calcule o fatorial explicitamente, e faça $e^x = 1/e^{-x}$ quando x for negativo. Sua implementação deve ir sequencialmente somando termos até que o termo seja muito pequeno (10^{-12}).

2. Faça uma função `bisseccao(f, a, b)` que procure por um zero da função f no intervalo $[a, b]$ usando o método da bisseccção. Teste com `f(x) = exp(x)*x - 1` e o intervalo $[0, 1]$. Seu algoritmo deve parar quando $|f(x)| < 10^{-12}$, ou se fizer mais de 5000 iterações. Se não existe garantia de que existe um zero de f no intervalo $[a, b]$, dê uma mensagem de erro (usando `error`).

3. Implemente as funções `solve_tri_inf(L,b)` e `solve_tri_sup(U,b)` que resolvem sistemas triangulares inferior e superior, respectivamente.

4. Implemente a função `elim_gauss(A,b)` que resolve o sistema linear $Ax = b$ através da Eliminação Gaussiana. Use a função anterior para resolver o sistema triangular final. Suponha que A é não singular.

5. Atualize a função `elim_gauss(A,b)` para o caso de A ser singular:
   - Se o sistema for compatível, retorne qualquer solução;
   - Se o sistema for impossível, gere um erro.
6. Crie uma função que calcula o determinante de uma matriz utilizando a regra de Laplace e uma função que calcula o determinante de uma matriz utilizando a eliminação Gaussiana. Compare as duas implementações.
7. Implemente a função `lu_dec(A)` que retorna as matrizes L e U da decomposição LU sem pivoteamento da matriz A.
8. Atualize o seu código de LU para retornar um vetor P de reordenação da linhas de A de modo que `L*U = A[P,:]`. Essa é a implementação tradicional de LU com pivoteamento.
9. Implemente a decomposição de Cholesky.
10. Implemente a decomposição LDL (LDLt).
11. Crie uma nova função `elim_gauss(A,B)` onde B é uma matriz e você quer resolver a equação $AX = B$ através da eliminação Gaussiana.
12. Crie uma função `inversa(A)` que calcula a inversa da matriz A. Use a função anterior.

3 Otimização em uma variável

Aqui consideramos o problema de minimizar uma função real de uma variável, possivelmente com algum restrição de intervalo. Algumas das funções daqui serão implementadas posteriormente.

1. Implemente o método de Newton para encontrar o mínimo de uma função. A sua função deve receber f, sua derivada, sua segunda derivada, um ponto inicial, e opcionalmente (procure Keyword Argument na documentação Julia), as tolerâncias para o critérios de parada $|f'(x)| < \varepsilon$ e o máximo de iterações.
2. Implemente o método da razão áurea (Golden Section Search).
3. Implemente a função `min_quad_box_1d(a, b, c, l, u)` que encontra o mínimo **global** de $f(x) = ax^2 + bx + c$ no intervalo $[\ell, u]$.