CCI-22 2018
LabExtra Bimestre 1

21 de fevereiro de 2018

Conta como um trabalho normal. Deve ser entregue até uma semana antes do prazo de fechamento de notas do 1o bimestre.

1 [20pt] Epsilon da máquina generalizado

Na primeira atividade, encontramos o chamado epsilon da máquina, que é definido como o menor número ε tal que $\varepsilon + 1 > 1$. Generalizando o conceito, para um número x, queremos encontrar o menor número $\varepsilon(x)$ tal que $\varepsilon(x) + x > x$. Implemente uma função matlab com cabeçalho `function epsilon = epsx(x)` que calcule $\varepsilon(x)$, e use a função para calcular $\varepsilon(x)$ para $x = 10^i$, para todos os valores inteiros de i entre -300 e +300.

- plote um gráfico com os valores calculados de $i \times \varepsilon(10^i)$. O eixo y deve ter uma escala logaritmica (use a função `semilogy`). Como já estamos usando os valores do expoente i no eixo x, o eixo x pode ser escalar.
- Comente: qual a curva encontrada, e qual a razão do formato específico desta curva?

2 Intersecção de Quádricas

Examine o código quaddemo. Quádricas no plano (círculos, elipses, parábolas, hipérboles) podem ser definidas em geral pela seguinte equação não-linear:

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

onde $A \quad B \quad C \quad D \quad E \quad F$ são as constantes que definem a quádrica, que é portanto representada por um vetor de 6 elementos $\mathbf{q} = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 & q_4 & q_6 \end{bmatrix} = \begin{bmatrix} A & B & C & D & E & F \end{bmatrix}$.

Já são fornecidas funções para:

1. plotar um countour plot para a quádrica, para conferir formato e gradiente.
2. Dados centro e raio de uma circunferência, encontrar o vetor $\mathbf{q}$ correspondente.
3. calcular $q(x, y) = Ax^2 + Bxy + Cy^2 + Dx + Ey + F$ dados $\mathbf{q}$, x e y.
4. calcular $\frac{\partial q(x,y)}{\partial x} = 2Ax + By + D$ dados $\mathbf{q}$, x e y.

5. calcular $\frac{\partial q(x,y)}{\partial y} = Bx + 2Cy + Ey$ dados $\mathbf{q}$, x e y.

6. Dadas 3 quádricas $\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3$, e valores para x e y, calcular um vetor de valores $\mathbf{F}(x,y) = \left[\begin{array}{ccc} q_1(x,y) & q_2(x,y) & q_3(x,y) \end{array}\right]^T$ e o valor do correspondente Jacobiano $\mathbf{J}(x,y)$, que é uma matriz 3×2. Esta é toda a informação necessária para aplicar o método de reolução de sistemas não-lineares e encontrar a intersecção das 3 quádricas.

2.1 [15pt] Autocorreção: [mysol] = mynonlinearsolver(F,x0)

O primeiro argumento F é um ponteiro de função, usado para calcular a função F(x,y) e o seu Jacobiano. O arquivo quadsolver.m e o próprio arquivo de testes já fornecem exemplos de como esta função deve ser chamada:

```
[mysol] = mynonlinearsolver(@(x) F(q1,q2,q3,x),x0);
```

Onde já existe uma função `F.m` parametrizada por q1,q2,q3, onde `x` é o parâmetro livre correspondendo às coordenadas desejadas. Ou seja, dentro de `mylinearsolver`, `F` é apenas uma função com um argumento `x`.

O arquivo `mynonlinearsolver.m` fornecido já contem exemplo de como utilizar a função `F.m`.

O segundo argumento x0 é a estimativa inicial da solução. Note que a estimativa inicial faz parte da definição do problema. Mudando as condições iniciais, as condições de convergência mudam: é possível, dependendo das quádricas, que ao iniciar em uma coordenada diferente, a convergência seja mais demorada ou mesmo que não haja convergência!

A saída deve ser uma matriz onde em cada linha aparece uma solução intermediária correspondendo a uma iteração. A primeira coluna é a coordenada x da solução, a segunda a coordenada y. A última linha deve corresponder a solução final correspodendo à última iteração.

2.2 [15 pt] Análise

Também é fornecida uma função `quadsolver` que, dadas 3 quádricas, plota-as, chama a função matlab `fsolve` para resolver o sistema não-linear, chama também a outra função resolvedora, `mynonlinearsolver.m`, que você deve implementar, para resolver o sistema, e plota os correspondentes resultados.

Portanto, você deve implementar a função resolvedora, conforme os parâmetros de entrada e formato de saída especificados no código, e utilizar `quadsolver` com:

- um exemplo de 3 quadricas que NÃO possuem um ponto comum de intersecção, mas que se intersectam em pares, e além disso deve existir um ponto não muito longe de todas as 3 quádricas, ou seja, elas quase se intersectam. A sua solução pode convergir ou não até o ponto de quase-intersecção.

- um exemplo de 3 quadricas que possuem um ponto comum de intersecção. A sua solução deve convergir até o ponto de quase-intersecção.

Os seus exemplos devem usar pelo menos 2 quádricas diferentes entre elipses, parábolas e hiperboles. (círculos são elipses)

- as funções que desenham gráficos usam um domínio de $[-2\pi, 2\pi]$. Utilize quádricas que se intersectam dentro deste domínio, e de preferência relativamente longe das bordas para não prejudicar o desenho.
- A sua solução deve retornar um vetor com soluções intermediárias, em ordem, de forma que o último elemento seja a última solução e o primeiro seja o chute inicial. Cada elemento é um ponto 2D com coordenadas (x, y).
- Não vale copiar os exemplos dos testes. A idéia é entender como usar as quádricas.

2.2.1 O que entregar

- Inclua os gráficos no pdf do relatorio, mostrando as quadricas e as iterações da solução, desde o chute inicial, passando pelas soluções intermediárias a cada iteração, e a solução final, para os dois exemplos.
- Submeta script que define as quadricas e chama as funções necessárias para gerar os gráficos.

3 Inversa

3.1 [10pt] Autocorreção: `Ai = myinv(A)`

O teste inclui apenas o caso bem condicionado, com os tamanhos indicados em 3.2.

3.2 [15pt] Análise

Implemente um dos algoritmos indicados na aula de resolução de sistemas lineares para encontrar a matriz inversa. Apresente um script que realize os testes indicados abaixo:

- Inverta matrizes bem condicionadas de tamanho 4, 8 e 16 (utilize a função `generateWellCond` do lab3)
- Inverta as matrizes de hilbert de tamanho 4, 8 e 16. (use a função `hilb()` do matlab)
- Verifique os seus resultados fazendo $max(max(abs(\mathbf{I} - \mathbf{A}\mathbf{A}^{-1})))$, onde $\mathbf{A}^{-1}$ é o resultado do seu programa e $\mathbf{I}$ é a matriz identidade de tamanho apropriado que pode ser obtida com o comando `eye(<tamanho>)`. Compare com $max(max(abs(\mathbf{I} - \mathbf{A} * inv(\mathbf{A}))))$ onde $inv(\mathbf{A})$ indica utilizar a função $inv()$ do matlab.

Comente os resultados.

4 Base de Chebyshev

Além de propor os nós de Chebyshev, a mesma pessoa, Pafnuty Chebyshev, propôs uma base para o espaço vetorial formado pelo conjunto de todos os polinômios de grau n. Esta base é chamada de *Polinômios de Chebyshev*. Os nós de Chebyshev também são os extremos dos Polinômios de Chebyshev no intervalo $[-1, 1]$, mas não confunda os dois conceitos!

Os primeiros polinômios de Chebyshev são:

$$\begin{array}{lll} T_0(x) = 1 & T_1(x) = x & T_2(x) = 2x^2 - 1 \\ T_3(x) = 4x^3 - 3x & T_4(x) = 8x^4 - 8x^2 + 1 & \quad etc. \end{array}$$

que pode ser generalizado com a seguinte fórmula recursiva dados $T_0(x) = 1$ e $T_1(x) = x$:

$$T_j(x) = 2xT_{j-1}(x) - T_{j-2}(x)$$

Como os polinômios de chebyshev são uma base do conjunto de polinômios, todo polinômio $p(x)$ pode ser escrito como uma combinação linear dos polinômios de chebyshev:

$$p(x) = \sum_{j=0}^{n} c_j T_j(x) \tag{1}$$

onde n é o grau do polinômio. Note que $T_i(x)$ tem grau i, da mesma forma que, na base canônica de monômios x^i tem grau i. Portanto, podemos encontrar o polinomio interpolador usando exatamente o mesmo algoritmo da matriz de vandermonde, mas com uma matriz de coeficientes diferente.

Com a base canônica de monômios $\{x^0, x^1, \ldots, x^n\}$, montamos o sistema linear como:

$$\begin{bmatrix} x_0^0 & x_0^1 & \cdots & x_0^n \\ x_1^0 & x_1^1 & \cdots & x_1^n \\ \vdots & \vdots & \ddots & \vdots \\ x_n^0 & x_n^1 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

onde os pares (x_i, y_i) são os nós de interpolação, e encontramos o vetor com os elementos a_i que serão os coeficientes do polinômio interpolador. Agora, usando a base de Chebyshev:

$$\begin{bmatrix} T_0(x_0) & T_1(x_0) & \cdots & T_n(x_0) \\ T_0(x_1) & T_1(x_1) & \cdots & T_1(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ T_0(x_n) & T_1(x_n) & \cdots & T_n(x_n) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

Note que as únicas incógnitas são os coeficientes c_i. Ao resolver este sistema linear, não obtemos mais diretamente os coeficentes a_i do polinômio interpolador, mas os coeficientes c_i da equação 1. Portanto, para encontrar o polinômio interpolador, agora temos que somar os polinômios de chebyshev considerando os coeficientes c_i da equação 1.

A vantagem de utilizar os polinômios de Chebyshev como base, é que, no intervalo $[-1, 1]$, eles são muito mais diferentes entre si do que os monômios da base canônica!

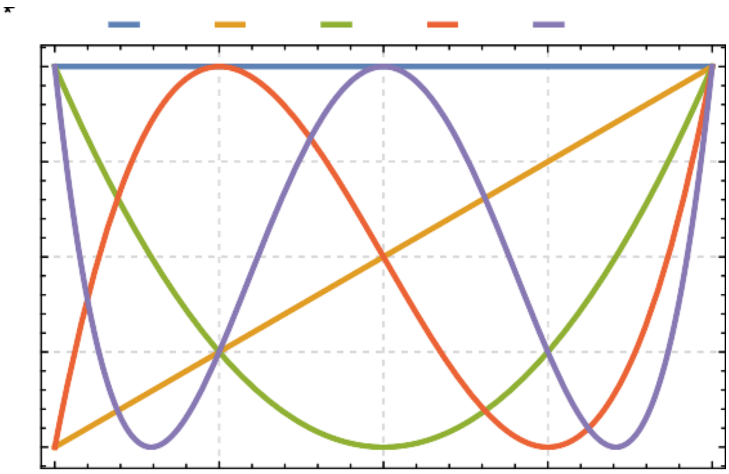

Figura 1: Polinômios de Chebyshev

A figura 1 mostra os polinômios de Chebyshev no intervalo $[-1, 1]$. Portanto, a matriz de coficientes do sistema linear tende a ser mais bem condicionada! Com as seguintes considerações:

- O teorema que indica que sempre existe uma função que faz a interpolação divergir ainda é válido. Mas com uma base melhor, é mais difícil de divergir. Ou pelo menos, precisamos de mais pontos de interpolação para observar a divergência (note que com tamanho maior, a matriz tende a se tornar pior condicionada nos dois casos)
- A interpolação deve ser aplicada apenas no intervalo $[-1, 1]$. Caso os pontos de interpolação estejam fora do intervalo, eles devem ser reescalados para este intervalo, e depois uma transformação de escala inversa deve ser aplicada no polinômio encontrado para se chegar ao polinômio final.

Mas não nos preocuparemos com os detalhes acima neste exercício.

4.1 [25pt] Tarefa: Compare com as outras soluções

Implemente a interpolação polinomial utilizando a base de Chebyshev, e estenda o exercício correspondente no lab4 para também comparar com a interpolação usando a base de chebyshev. Introduza 2 novas linhas em cada tabela, que agora se tornam:

função de Runge	nós	$\min E(n)$	n para $\min E(n)$	converge para
PolinomioInterpolador	regular			
InterpolacaoNewton	regular			
SplineCubica	regular			
InterpolacaoChebyshev	regular			
PolinomioInterpolador	Chebyshev			
InterpolacaoNewton	Chebyshev			
SplineCubica	Chebyshev			
InterpolacaoChebyshev	Chebyshev			

função g(x)	nós	$\min E(n)$	n para $\min E(n)$	converge para
PolinomioInterpolador	regular			
InterpolacaoNewton	regular			
SplineCubica	regular			
InterpolacaoChebyshev	regular			
PolinomioInterpolador	Chebyshev			
InterpolacaoNewton	Chebyshev			
SplineCubica	Chebyshev			
InterpolacaoChebyshev	Chebyshev			

responda novamente todas as perguntas, estendendo as suas respostas para também considerar o novo algoritmo. Pode ajudar a refletir e/ou explicar alguma diferença, se, nos algoritmos baseados em resolução de sistema linear, retornar ou imprimir `cond(A)`, onde A é a matriz de coeficientes, um valor que indica o condicionamento da matriz.

Inclua os gráficos correspondentes à função de Runge com este novo método.

4.2 Dicas:

funções já fornecidas pelo professor:

- `getChebyshevPolyCoef(i)` retorna os coeficientes de T_i no formato de `polyval`.
  - Esta função é lenta, pois utiliza o symbolic toolbox. Demora na ordem de dezenas de segundos para completar a função `Runge` até $n = 50$ se chamar esta função a cada avaliação ou soma de polinômio. Se desejar acelerar o algoritmo, implemente a fórmula recursiva e calcule você mesmo. Ou, mais rápido ainda, pré-calcule os coeficientes até T_{50}, salve-os em um arquivo .mat, e use os coeficientes armazenados ao invés de recalculá-los.
  - Um bom formato para armazenar os coeficientes é um cell array: p{1} = [1 2 3 4]; p{2} = [5 6 7]; p{3} = 2; Veja que é diferente de uma matriz, pois cada elemento do cell array pode ter tamanhos diferentes.
  - Para salvar e carregar dados, veja os comandos `save` e `load`. É possivel salvar todas as variáveis do ambiente corrente, ou apenas algumas variáveis.

- `sumpoly(p1,p2)` soma dois polinomios no formato de `polyval`. Já leva em conta tamanhos de vetor diferentes correspondendo a graus diferentes.

5 Instruções:

- A questão 4 é obviamente dependente da questão correspondente no lab de interpolação. Se trata apenas de um passo a mais. Mas se não fez o lab de interpolação, não precisa completar a tabela apenas para esta atividade.