CCI-22 2017
Lab3 Sistemas Lineares Parte 2

15 de março de 2018

1 Tarefas:

1.1 Implementação

Implementar as seguintes funções em MATLAB (cada uma em um arquivo .m separado). O arquivo de teste `testLab3LinSysP2Aluno.m` contem testes para estas funções:

1. [1 pt] `satisfaz = CriterioLinhas(A)`: verifica se a matriz $\mathbf{A}$ satisfaz o Critério das Linhas (condição suficiente para convergência do Método de Gauss-Jacobi) e retorna o resultado como a booleana `satisfaz`.

2. [2.5 pt] `[x, dr] = GaussJacobi(A, b, x0, epsilon, maxIteracoes)`: resolve o sistema $\mathbf{Ax} = \mathbf{b}$ através do Método de Gauss-Jacobi, usando `x0` como chute inicial, `epsilon` como tolerância para critério de parada por erro relativo e `maxIteracoes` como limite máximo de iterações. A solução $\mathbf{x} \in \mathbb{R}^{n\times 1}$ (`x`) é retornada, juntamente com o vetor `dr`, que contém os erros relativos de todas as iterações, de modo que `dr(k)` é o erro relativo calculado na iteração k . Considere $\mathbf{A} \in \mathbb{R}^{n\times n}$ e $\mathbf{b} \in \mathbb{R}^{n\times 1}$.

3. [1 pt] `[satisfaz, beta] = CriterioSassenfeld(A)`: verifica se a matriz $\mathbf{A}$ satisfaz o Critério de Sassenfeld (condição suficiente para convergência do Método de Gauss-Seidel) e retorna o resultado como a booleana satisfaz. Além disso, o β (`beta`) do Critério de Sassenfeld é retornado.

4. [2.5 pt] `[x, dr] = GaussSeidel(A, b, x0, epsilon, maxIteracoes)`: resolve o sistema $\mathbf{Ax} = \mathbf{b}$ através do Método de Gauss-Seidel, usando `x0` como chute inicial, `epsilon` como tolerância para critério de parada por erro relativo e `maxIteracoes` como limite máximo de iterações. A solução $\mathbf{x} \in \mathbb{R}^{n\times 1}$ (`x`) é retornada, juntamente com o vetor `dr`, que contém os erros relativos de todas as iterações, de modo que `dr(k)` é o erro relativo calculado na iteração k . Considere $\mathbf{A} \in \mathbb{R}^{n\times n}$ e $\mathbf{b} \in \mathbb{R}^{n\times 1}$.

Considere a seguinte definição de erro relativo na iteração k :

$$d_r^{(k)} = \frac{\max_{1\leq i\leq n} \left| x_i^{(k)} - x_i^{(k-1)} \right|}{\max_{1\leq i\leq n} \left| x_i^{(k)} \right|}$$

1.2 [2 pt] Análise 1

Analise o comportamento dos métodos de Gauss-Jacobi e Gauss-Seidel quando aplicados aos seguintes sistemas lineares de equações:

$$\begin{cases} x_1 + 3x_2 + x_3 = -2 \\ 5x_1 + 2x_2 + 2x_3 = 3 \\ 6x_2 + 8x_3 = -6 \end{cases} \tag{1}$$

$$\begin{cases} 5x_1 + 2x_2 + 2x_3 = 3 \\ x_1 + 3x_2 + x_3 = -2 \\ 6x_2 + 8x_3 = -6 \end{cases} \tag{2}$$

$$\begin{cases} x_1 + 3x_2 + 4x_3 = 8 \\ x_1 - 3x_2 + x_3 = -9 \\ x_1 + x_2 + 5x_3 = 1 \end{cases} \tag{3}$$

$$\begin{cases} x_1 + 2x_2 - 2x_3 = 3 \\ x_1 + x_2 + x_3 = 0 \\ 2x_1 + 2x_2 + x_3 = 1 \end{cases} \tag{4}$$

$$\begin{cases} 2x_1 + x_2 + x_3 = 4 \\ x_1 + 2x_2 + x_3 = 4 \\ x_1 + x_2 + 2x_3 = 4 \end{cases} \tag{5}$$

$$\begin{cases} 5x_1 - x_2 + x_3 = 10 \\ 2x_1 + 4x_2 - x_3 = 11 \\ -x_1 + x_2 + 3x_3 = 3 \end{cases} \tag{6}$$

Apresente em uma tabela os resultados dos critérios de convergência quando aplicados a estes sistemas, assim como o número de iterações necessárias para a convergência para cada método iterativo de solução. Caso ocorra divergência do método, deixe isto indicado na tabela.

Para padronizar a análise, utilize $x^{(0)} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$, $\varepsilon = 0,001$ e limite máximo de iterações de 100 para todos os casos.

Discuta os resultados obtidos, incluindo:

- Analise como atendimento aos critérios de convergência se relaciona com convergência efetiva dos métodos nestes casos. Note que, para ambos os métodos implementados, atendimento ao respectivo critério de convergência indica condição suficiente, mas não necessária, para convergência.

- Perceba que os sistemas 1 e 2 diferem apenas por uma troca entre as linhas 1 e 2, porém os comportamentos dos métodos quando submetidos a estes sistemas é muito diferente. Explique esta diferença à luz dos critérios de convergência.

- Nos casos em que ambos os métodos convergem, comente qual converge mais rápido (i.e. requer um menor número de iterações para atender ao critério de parada) e se isto é esperado.
- Plote o vetor de erros relativos `dr` para alguns casos para ajudar a embasar suas conclusões.

Modelo de tabela para resposta

sistema	Gauss-Jacobi		Gauss-Seidel	
	Converge	# Iterações	Converge	# Iterações
(1)	Sim ou Não	1 a 100	Sim ou Não	1 a 100
(2)	Sim ou Não	1 a 100	Sim ou Não	1 a 100
(3)	Sim ou Não	1 a 100	Sim ou Não	1 a 100
(4)	Sim ou Não	1 a 100	Sim ou Não	1 a 100
(5)	Sim ou Não	1 a 100	Sim ou Não	1 a 100
(6)	Sim ou Não	1 a 100	Sim ou Não	1 a 100

Note que "não converge" significa que o algoritmo parou no número máximo de iterações.

1.3 [3 pt] Análise 2: Três cenários

O arquivo de teste `testLab3LinSysP2BigAluno.m` testa uma solução iterativa com matrizes maiores. Para efeitos do exercício 1.1, este novo teste não é necessário, é suficiente apenas o teste anterior com matrizes menores. Mas para este exercício, se o seu código não passa no teste com matrizes maiores, a sua análise não é credível. Portanto é preciso passar no teste "Big" antes de responder a este exercício.

Forneço funções matlab para testar três cenários diferentes com as funções desenvolvidas nas partes 1 e 2. O objetivo é discriminar quando devem ser aplicados os métodos iterativos ou os métodos diretos.

Execute o script `runcompLinSys.m`. Ele define uma lista de ponteiros de funções com 4 funções que resolvem sistemas lineares, por Eliminação de Gauss, Decomposição LU, Gauss-Jacobi e GaussSeidel. As duas últimas são diretamente as que você implementou. As duas primeiras, são fornecidas, e apenas chamam as suas funções.

A seguir, o script chama a função `compLinSysTimes`, em dois cenários:

1. matrizes aleatorias de tamanho 50 (10 testes).
2. matrizes aleatórias esparsas[1] de tamanho 1000 (5 testes)

[1] o matlab representa matrizes esparsas de forma diferente. Mas para nós isso é transparente, pois muitas funções matlab aceitam ambas as representações. Na verdade a matriz deixa automaticamente de ser esparsa, e volta à representação usual, com determinadas operações que não aceitam a matriz esparsa, ou fazem com que a matriz deixe de ser esparsa. Estes fatos são parte da explicação na diferença de resultados. Mas, se um determinado método é capaz de aproveitar o fato da matriz ser esparsa para resolver mais rápido, isso é bom, certo? Isso não invalida os resultados.

Para construir estes dois cenários as funções geradoras de matrizes utilizam a sua função `CriterioSassenfeld`, para garantir que todas as matrizes podem ser resolvidas por todos os métodos, inclusive os iterativos.

Finalmente, o script chama a função `compLinSysTimesHilbert`, no terceiro cenário:

3. matrizes de hilbert de tamanho 6,7,8 (3 testes)

Tabele os seus resultados no relatório e baseado neles responda:

1. Quais métodos são melhores ou piores, tanto em relação ao tempo de execução quanto ao erro final (medido como o resíduo)? Principalmente, em quais cenários devemos aplicar métodos iterativos e em quais cenários devemos aplicar métodos diretos?

2. O que diferencia um cenário do outro e explica os resultados da pergunta anterior?

3. Como podemos discriminar entre os cenários acima (sem considerar as diferenças de tamanho da entrada), ou seja, dado que temos uma matriz de entrada A, e sabemos apenas:

   (a) A satisfaz o cenário de sassenfeld; AND

   (b) A é aleatória, ou A é aleatória esparsa, ou A é uma matriz muito mal condicionada, e apenas uma destas opções;

   Descreva os passos para resolver o sistema ou concluir que não podemos resolvê-lo com precisão ou em tempo hábil. Isto implica: decidir entre as opções de (b); então decidir qual método aplicar e avaliar se esperamos obter ou não uma solução rápida e precisa.

2 Instruções:

- Para responder às perguntas da parte de análise, vale pesquisar qualquer bibliografia.

- A primeira etapa do processo de correção consistirá em submeter as funções implementadas a vários casos de teste de forma automatizada. Assim, os cabeçalhos das funções devem ser seguidos **rigorosamente**. Arquivos .m com os nomes destas funções e os cabeçalhos já implementados foram fornecidos juntamente com este roteiro. Dê preferência a implementar seu laboratório a partir destes arquivos .m fornecidos para evitar erros.

- **Não** é permitido o uso de funções ou comandos prontos do MATLAB que realizem toda a funcionalidade atribuída a uma certa função. Entretanto, o uso destas funções para verificação das implementações realizadas é encorajado. Em caso de dúvida quanto à permissão de uso de alguma função ou comando, recomenda-se consultar o professor.

- Não é necessário se preocupar com verificação dos dados de entrada: assuma que as dimensões de **A** e **b** são compatíveis.
- Os arquivos .m implementados devem ser entregues juntamente com um relatório.
- No relatório, não é necessário demonstrar que as funções implementadas funcionam corretamente . Basta incluir resultados e conclusões da parte relativa a **Análise**.

3 Dicas:

- Submeta suas funções a vários casos de teste e compare com os resultados obtidos usando funções e comandos prontos do MATLAB. Devido a imprecisões numéricas, os resultados podem diferir um pouco, porém espera-se que as diferenças sejam bem pequenas. A função `cond(A)` do matlab verifica se a matriz A é bem condicionada. Valores altos indicam matrizes mal condicionadas (nos testes aceitamos matrizes onde $cond(A) < 1000$), o que pode ser verificado no código das funções que geram matrizes para teste.
- A função `hilb(n)` do matlab gera matrizes de Hilbert de tamanho `n`.
- Curiosidade: execute `cond(hilb(n))` para alguns diferentes valores de n, entre 3 e 20.