Métodos Numéricos

Resolución de Sistemas de Ecuaciones Lineales
Métodos Directos

Diego Passarella

Universidad Nacional de Quilmes

1^{er} Cuatrimestre de 2016

Sistemas de Ecuaciones Lineales

Se estudia la resolución de problemas del tipo

$$A\vec{x} = \vec{b}$$

con A una matriz cuadrada e inversible.

La resolución de x_i utilizando la regla de Cramer

$$x_i = \frac{\det(A_i)}{\det(A)}$$

implica resolver $n+1$ determinantes, cada uno a un costo aproximado a $2n!$ operaciones. Por lo tanto, utilizar este tipo de estrategia consume aproximadamente $2(n+1)!$ operaciones de máquina (sumas ó multiplicaciones).

Sistemas de Ecuaciones Lineales

Métodos para resolver S.E.L:

- Directos: Para sistemas "relativamente" pequeños y con matrices densamente pobladas. Se basan en operar sobre la matriz A o su expandida $[A|b]$ para llevarla a una forma que sea fácilmente resoluble (matriz diagonal o triangular). Se obtiene la solución (idealmente exacta) en un número finito de operaciones conocido *a priori*.
- Iterativos: Para sistemas grandes y estructuradamente ralos (muchos ceros dispuestos de forma conocida en la matriz A). Se genera una sucesión que aproxima a la solución por medio de la repetición de operaciones *matriz* $\times$ *vector* en cada iteración. Se llega a una aproximación de la solución con una dada tolerancia en una cantidad *a priori* desconocida de pasos.

Matriz triangular inferior

Una matriz triangular inferior es de la forma:

$$L = \begin{pmatrix} l_{1,1} & 0 & 0 & \cdots & 0 \\ l_{2,1} & l_{2,2} & 0 & \cdots & 0 \\ l_{3,1} & l_{3,2} & l_{3,3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n,1} & l_{n,2} & l_{n,3} & \cdots & l_{n,n} \end{pmatrix}$$

Algoritmo de descenso

Resolución de sistemas con matriz triangular inferior:

Algoritmo:

1) $x_1 = b_1/l_{1,1}$

2) para i=2 hasta n

$$x_i = \frac{b_i - \sum_{j=1}^{i-1} x_j l_{i,j}}{l_{i,i}}$$

Este agloritmo requiere n^2 operaciones para su resolución.

Matriz triangular superior

Una matriz triangular superior es de la forma:

$$U = \begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ 0 & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\ 0 & 0 & u_{3,3} & \cdots & u_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{n,n} \end{pmatrix}$$

Algoritmo de remonte

Resolución de sistemas con matriz triangular superior:

Algoritmo:

1) $x_n = b_n / u_{n,n}$

2) para i=n-1 hasta 1

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} x_j u_{i,j}}{u_{i,i}}$$

Este algoritmo requiere la misma cantidad de operaciones que el de descenso.

Eliminación de Gauss

Se opera con el sistema completo para hacer cero a los elementos debajo de la diagonal principal. Se trabaja con la matriz expandida $\widetilde{A} = [A|b]$.

La factorización de Gauss existe y es única si las submatrices principales A_i de orden $i = 1, \cdots, n-1$ son no singulares.

Esta factorización requiere del orden de $2n^3/3$ operaciones.

Eliminación de Gauss

Algoritmo:

1) para k=1 hasta n-1
(recorro las n-1 primeras columnas)

2) para i=k+1 hasta n
(recorro todas las filas debajo de la diagonal)

$$l_{i,k} = \widetilde{a}_{i,k}/\widetilde{a}_{k,k}$$

3) para j=k hasta n+1
(recorro toda la i-ésima fila)

$$\widetilde{a}_{i,j} = \widetilde{a}_{i,j} - l_{i,k}\widetilde{a}_{k,j}$$

Condición a verificar: $\widetilde{a}_{k,k} \neq 0 \quad \forall k.$

Eliminación de Gauss

El método de eliminación de Gauss se puede utilizar en su forma para:

- Matrices diagonalmente dominantes por filas
- Matrices diagonalmente dominantes por columnas (todos los multiplicadores resultan menores a 1)
- Matrices simétricas y definidas positivas

En otras condiciones, hay que verificar los multiplicadores resultantes y/o pivotar

Pivoteo

El método de factorización de Gauss puede generar grandes errores cuando el elemento $\widetilde{a}_{k,k}$ es pequeño (o directamente fallar cuando es cero). En esos casos, hay que sumar una etapa de pivoteo de filas en cada $k-$ésimo paso del algoritmo.

El sistema a resolver termina siendo:

$$PAx = Pb$$

Donde P es la matriz de permutaciones resultante del pivoteo.

Pivoteo - Ejemplo:

Factorizar el siguiente S.E.L. para $\varepsilon = 1$ y $\varepsilon = 0$

$$\begin{pmatrix} 1 & 1-\varepsilon & 3 \\ 2 & 2 & 2 \\ 3 & 6 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 5-\varepsilon \\ 6 \\ 13 \end{pmatrix}$$

¿Cuál es la matriz de pivoteo resultante?

Factorización LU

Idea: Transformar a la matriz A en el producto de dos matrices, una triangular superior y otra triangular inferior de manera que:

$$A = LU$$

$$Ax = b \Rightarrow LUx = b$$

con lo que se termina resolviendo dos sistemas de matriz triangular

$$Ly = b$$

$$Ux = y$$

La factorización LU no es única, por lo que se impone que la diagonal principal de L esté formada por 1's.

Factorización LU

Algoritmo:

Utilizar el mismo algoritmo que para la eliminación de Gauss, pero solamente a la matriz A (no a la expandida).

La matriz A resultante va a ser la matriz U y los coeficientes $l_{i,k}$ pasan a ser los elementos del triángulo inferior de L (agregar la diagonal de 1's).

Obviamente, su resolución requiere casi la misma cantidad de operaciones que la factorización de Gauss. No se opera con el vector de términos independientes, pero se deben resolver dos sistemas triangulares.

Factorización LU

Comentarios generales:

Para ahorrar memoria se pueden almacenar los coeficientes $l_{i,k}$ en las posiciones $a_{i,k}$ del triángulo inferior.

La factorización LU es conveniente cuando hay que resolver varios sistemas con la misma matriz A, pero con distintos vectores b.

En caso de ser necesario realizar pivotaje, los sistemas a resolver quedan:

$$PAx = Pb \Rightarrow PA = LU \therefore Ly = Pb \rightarrow Ux = y$$

Factorización LU - Cálculo de Determinantes

Recordar que:

$$det(AB) = det(A)det(B)$$

Por lo tanto, se puede verificar fácilmente que:

$$det(A) = det(LU) = det(L)det(U) = det(U) = \sum_{i=1}^{n} u_{i,i}$$

Factorización de Thomas

Resolución de sistemas tridiagonales utilizando la factorización LU.

$$\begin{pmatrix} a_1 & b_1 & 0 & \cdots & 0 \\ c_2 & a_2 & b_2 & \cdots & 0 \\ 0 & c_3 & a_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & c_n & a_n \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ \gamma_2 & 1 & 0 & \cdots & 0 \\ 0 & \gamma_3 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & \gamma_n & 1 \end{pmatrix} \begin{pmatrix} \alpha_1 & \beta_1 & 0 & \cdots & 0 \\ 0 & \alpha_2 & \beta_2 & \cdots & 0 \\ 0 & 0 & \alpha_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & \alpha_n \end{pmatrix}$$

Factorización:

$\alpha_1 = a_1, \quad \beta_i = b_i \, \forall \, i : 1, \cdots, n-1, \quad \gamma_i = \frac{c_i}{\alpha_{i-1}} \, \forall \, i : 2, \cdots, n$

$\alpha_i = a_i - \gamma_i \beta_{i-1} \, \forall \, i : 2, \cdots, n$

Resolución:

$Ly = b \rightarrow y_1 = b_1, \quad y_i = b_i - \gamma_i y_{i-1} \, \forall \, i : 1, \cdots, n$

$Ux = y \rightarrow x_n = y_n / \alpha_n, \quad x_i = (y_i - \beta_i x_{i+1}) / \alpha_i \, \forall \, i : n-1, \cdots, 1$

Este tipo de factorización permite resolver el sistema original con un costo computacional del orden de n operaciones.

Factorización de Cholesky

Cuando la matriz A es simétrica ($A = A^t$) y definida positiva ($x^t Ax > 0 \quad \forall x \neq \overline{0}$), se puede realizar la siguiente factorización:

$$A = LL^t$$

Se descompone a A en el producto de dos matrices triangulares, donde una es la traspuesta de la otra.

Esta factorización requiere menos operaciones que la factorización LU ($n^3/3$) y permite ahorrar memoria, dado que solamente se almacenan los coeficientes de L.

Factorización de Cholesky

Algoritmo:

1) $l_{1,1} = \sqrt{a_{1,1}}$

2) para i=2 hasta n (recorro toda la primera columna)
   $l_{i,1} = a_{i,1}/l_{1,1}$

3) para j=2 hasta n-1 (recorro las columnas interiores)
   $l_{j,j} = \sqrt{a_{j,j} - \sum_{k=1}^{j-1} l_{j,k}^2}$
   para i=j+1 hasta n (recorro los elem. inferiores de la col.)
   $l_{i,j} = \left(a_{i,j} - \sum_{k=1}^{j-1} l_{i,k} l_{j,k}\right) / l_{j,j}$

4) $l_{n,n} = \sqrt{a_{n,n} - \sum_{k=1}^{n-1} l_{n,k}^2}$

Importante: Se debe verificar que los argumentos de las raíces sean positivos.

Condicionamiento

Sensibilidad a perturbaciones:

Es interesante analizar la sensibilidad del resultado de x ante perturbaciones en el vector de términos independientes b.

$$Ax = b \rightarrow A(x + \delta x) = (b + \delta b)$$

Los errores resultantes están acotados por:

$$\frac{||\delta x||}{||x||} \leq K(A) \frac{||\delta b||}{||b||}$$

Número de condición de una matriz

El número de condición se define como:

$$K_p(A) := ||A||_p \cdot ||A^{-1}||_p$$

Esta forma de calcularlo no suele ser práctica, por lo que hay que usar definiciones más simples que son válidas para cierto tipo de matrices.

Para matrices simétricas y definidas positivas, el número de condición $K_2(A)$ esta dado por:

$$K_2(A) = \frac{\lambda_{max}}{\lambda_{min}}$$

Número de condición de una matriz

Algunas características de $K(A)$:

- El número de condición de una matriz A mide la sensibilidad de la solución de un S.E.L. respecto a variaciones en b.
- Otorga una cota máxima de cuánto se puede amplificar el error en el cálculo de x, al cometer un error en la determinación de b (o tener incertidumbre en su medida).
- El sistema se dice bien condicionado si $K(A)$ es pequeño ($K(A) \geq 1$ siempre).
- Se puede mejorar el condicionamiento de un sistema multiplicando filas por coeficientes adecuados (precondicionamiento).

Ejemplo de condicionamiento

Analizar el resultado x del siguiente S.E.L ante pequeñas perturbaciones de b:

$$\begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix}$$

Comandos útiles de Matlab

- `det`: para el cálculo de determinantes de matrices.
- `norm`: para el cálculo de normas de matrices y vectores.
- `cond`: para el cálculo de números de condición de matrices.
- `\`: para resolver sistemas lineales con un método directo.
- `rand`: generador de escalares, vectores o matrices de números random $\in (0, 1)$.
- `eye/ones/NaN`: generador de vectores o matrices que contienen 1's en la diagonal, todos 1's o `NaN`'s.

recuerden usar el `help ó doc (+ comando)` en la ventana de comandos de Matlab para más información.