Álgebra Lineal Numérica

Gabriel Pinochet Soto

23 de mayo de 2021

Problema 1. *Sea* $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$. *Calcular su determinante usando* `MATLAB`*. Explicar por qué el resultado no es un entero.*

Veamos, en primera instancia, que A es singular, pues sus vectores fila son linealmente dependientes. En efecto,

$$(-1)\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}^{\mathrm{T}} + 2\begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 7 \\ 8 \\ 9 \end{bmatrix}^{\mathrm{T}}. \tag{1}$$

De esto, y de la multilinealidad del determinante, tenemos $\det A = 0$.

Notar que en la documentación de `MATLAB` se menciona que se hace uso de la factorización LU para calcular el determinante de una matriz. Esto induce errores de aritmética de punto flotante. Para verificar esta última afirmación con nuestra matriz A, recordemos que `MATLAB` utiliza los pivotes de mayor valor. Hacemos énfasis en que nos enfocaremos en la matriz $U := U(A)$ de la factorización LU, pues es la que lleva diagonal significativa. Usaremos eliminación Gaussiana para repasar el error de arrastre. Si hacemos las (dos) operaciones filas necesarias para simplificar la primera columna, obtenemos

$$\begin{bmatrix} 0 & 6/7 & 12/7 \\ 0 & 3/7 & 6/7 \\ 7 & 8 & 9 \end{bmatrix}.$$

Notar que usando `format long` en `MATLAB` nos entrega las siguientes aproximaciones:

$$3/7 \approx 0{,}428571428571429, \qquad 6/7 \approx 0{,}857142857142857, \qquad 12/7 \approx 1{,}714285714285714.$$

Al colocar en la consola de `MATLAB` el siguiente cálculo `0.428571428571429*2 - 0.857142857142857` obtenemos por resultado algo no trivial, `9.992007221626409e-16`. En consecuencia, el bloque $\begin{bmatrix} 6/7 & 12/7 \\ 3/7 & 6/7 \end{bmatrix}$ es no singular *numéricamente*. De esta forma, U es no singular numéricamente, y el determinante es no trivial. El siguiente bloque de código computa el determinante de A.

```
format long
A = [1 2 3; 4 5 6; 7 8 9];
det(A)
```

Problema 2. *El objetivo es comparar los desempeños de los métodos LU y Cholesky.*

1. *Escribir función* `LUFacto` *que devuelva matrices* L *y* U *vía Algoritmo 6.1. Si el algoritmo no puede ser ejecutado, devolver un mensaje de error*
2. *Escribir función* `Cholesky` *que devuelva matriz* B *vía Algoritmo 6.2. Si el algoritmo no puede ser ejecutado, devolver un mensaje de error. Comparar con la funcion* `chol`*.*
3. *Definir* `A = PdSMat(n)` *matriz simétrica positiva definida, y un lado derecho* `b = ones(n,1)`*. Comparar el tiempo de ejecución de* `LUFacto` *y la resolución con sustitución directa e inversa, contra el tiempo de ejecución de* `Cholesky` *y la resolución con sustitución directa e inversa. Graficar y comentar.*

Es importante comentar que la factorización LU (y en consecuencia Cholesky) puede ser almacenada en una sola matriz, pues acoplamos la matriz triangular inferior (ignorando su diagonal de unos, que es conocida) con la matriz triangular superior. Más aún, gracias a la eliminación Gaussiana, podemos definir ámbas matrices sin requerir memoria adicional. Justamente esto es lo que hace el Algoritmo 6.1 (ver (1)). Así mismo, en base al algoritmo 6.2, tenemos (2).

Notar que las diferencias principales con los algoritmos presentados en el libro son los bloques que estudian la no nulidad de los pivotes, en función de no realizar una operación inválida. En dichos casos se retorna un mensaje de error y matrices `NaN`.

Para comparar usamos un vector `times = zeros(2, 100)` para almacenar los tiempos de ejecución de las distintas factorizaciones. Suponiendo que el tiempo de ejecución es de la forma

$$\text{time}(n) \approx p_2 \cdot n^{p_1},$$

usamos `polyfit(log(x),log(times(i,:)),1)` para calzar la mejor línea en un grafico `loglog`, para los distintos

```
function [L U] = LUFacto(A)
    n = size(A,1);

    for k=1:(n-1)
        if (A(k,k)==0)
            disp('Error! Pivot must be nonzero')
            L = NaN;
            U = NaN;
            return
        else
            for i=(k+1):n
                A(i,k) = A(i,k)/A(k,k);
                for j=(k+1):n
                    A(i,j) = A(i,j) - A(i,k)*A(k,j);
                end
            end
        end
    end
    L = eye(n) + tril(A,-1);
    U = triu(A);
end
```

Código 1: Fatorización LU, vía Algoritmo 6.1

```
function B = Cholesky(A)
    if (A~=A')
        disp('Input must be symmetric!')
        B = NaN;
        return
    else
        B = tril(A);
        n = size(A,1);
        for j=1:n
            for k=1:(j-1)
                B(j,j) = B(j,j) - (B(j,k))^2;
            end
            if (B(j,j)<=0)
                disp('Error! Pivot must be positive')
                B = NaN;
                return
            else
                B(j,j) = sqrt(B(j,j));
                for i=(j+1):n
                    for k=1:(j-1)
                        B(i,j) = B(i,j) - B(j,k)*B(i,k);
                    end
                    B(i,j) = B(i,j)/B(j,j);
                end
            end
        end
    end
end
```

Código 2: Fatorización de Cholesky, vía Algoritmo 6.2

métodos. Ver (3). Obtenemos

$$\text{time}_{\text{Cholesky}}(n) \approx \exp(-19{,}2129)n^{2{,}6530}$$
$$\text{time}_{\text{chol}}(n) \approx \exp(-17{,}6267)n^{1{,}7777}.$$

Para el último ítem definimos las sustituciones directa (4) e inversa (5). Para la construcción de la matriz simétrica y positiva definida requerimos definir `SymmetricMat` (ver (6)) que entrega una matriz simétrica. El código `PdSMat` en (7), en base a perturbar los valores propios de una matriz simétrica, construye una matriz positiva definida.

Comparamos los valores bajo una premisa similar a la anterior. Ver (8). Tenemos

$$\text{time}_{\text{LU}-\text{Solver}}(n) \approx \exp(-18{,}4525)n^{2{,}7004}$$
$$\text{time}_{\text{C}-\text{Solver}}(n) \approx \exp(-18{,}6139)n^{2{,}5385}.$$

```
times = zeros(2,length(1:100));
for i = 1:100
    n = i*10;
    A = PdSMat(n);

    tic
    B = Cholesky(A);
    times(1,i) = toc;

    tic
    B = chol(A);
    times(2,i) = toc;
end
x = 10:10:1000;
p1 = polyfit(log(x),log(times(1,:)),1);
p2 = polyfit(log(x),log(times(2,:)),1);

figure
loglog(x,times(1,:),'*')
hold on
loglog(x,exp(p1(2)).*x.^p1(1),'--')
loglog(x,times(2,:),'o')
loglog(x,exp(p2(2)).*x.^p2(1),'--')
hold off
xlabel('Size of the matrix')
ylabel('Time of computation')
legend('Cholesky','Best line for Cholesky','chol','Best line for chol')
```

Código 3: Comparación de tiempos de computación para la factorización de Cholesky

```
function x = ForwSub(L,y)
    n = size(L,1);
    x = zeros(n,1);
    for i = 1:n
        s = 0;
        for j = 1:(i-1)
            s = s + L(i,j)*x(j);
        end
        x(i) = (y(i)-s)/L(i,i);
    end
end
```

Código 4: Sustitución directa

Problema 3. *El objetico es evaluar la influencia de la permutación de filas en la eliminación Gaussiana.*
Poner `e=1.E-16; A=[e 1 1;1 -1 1; 1 0 1]; b=[2 0 1]'`.

1. *Calcular las matrices* `[L U]` *usando* `LUFacto` *(ver* (1)*).*
2. *Definir dos matrices* `[l u]` *con* `[l u] = LUFacto(p*A)`*, con* `p` *matriz de permutación definita en* `[~ ~ p] = lu(A)`*. ¿Qué se observa?*
3. *Determinar la solución del sistema* $Ax = b$ *mediante* `BackSub(U,ForwSub(L,b))` *y* `BackSub(u,ForwSub(l,p*b))`*. Comparar con* $x = [0, 1, 1]^{\mathrm{T}}$.

Vemos que, usando (9) obtenemos una matriz `U` singular, con fila final nula. Esto se debe nuevamente a los errores de la aritmética de punto flotante. Procedemos como en el primer ejercicio. Hacemos eliminación Gaussiana *en la primera fila,* pues (1) no permuta las filas. En tal caso nos queda la matriz

$$\begin{bmatrix} 10^{-16} & 1 & 1 \\ 0 & -10^{16}-1 & -10^{16}+1 \\ 0 & -10^{16} & -10^{16}+1 \end{bmatrix}.$$

Vemos que la contribución de ± 1 en la submatriz $\begin{bmatrix} -10^{16}-1 & -10^{16}+1 \\ -10^{16} & -10^{16}+1 \end{bmatrix}$ es irrelevante, pues es de orden mucho menor que 10^{16}. Luego, al despreciarlo nos queda un bloque singular. En términos más abstractos, dados tres vectores u, v, w en un espacio vectorial normado, con u, v linealmente intependientes, podemos hacer que $u + \lambda w$ y $v + \lambda w$ se "parezcan" si consideramos λ suficientemente grande: podríamos despreciar u y v en dichas combinaciones lineales.

En contraste, en (10), si permutamos vía `[~ ~ p]=lu(A)` evitamos el fenómeno descrito anteriormente. Si permutamos la primera y la segunda fila de A, obtenemos $\begin{bmatrix} 1 & -1 & 1 \\ 10^{-16} & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$. Haciendo eliminación Gaussiana tenemos

$$\begin{bmatrix} 1 & -1 & 1 \\ 0 & 1+10^{-16} & 1-10^{-16} \\ 0 & 1 & 0 \end{bmatrix},$$

que después de despreciar la contribución de $\pm 10^{-16}$ sigue siendo no singular. Lo anterior se refleja en el resultado de (11): `x1` tiene componentes infinitas y `NaN`.

```
function x = BackSub(U,y)
    n = size(U,1);
    x = zeros(n,1);
    for i = n:-1:1
        s = 0;
        for j = n:-1:(i+1)
            s = s + U(i,j)*x(j);
        end
        x(i) = (y(i)-s)/U(i,i);
    end
end
```

Código 5: Sustitución inversa

```
function S = SymmetricMat(n)
    S = rand(n);
    S = S + S';
end
```

Código 6: Matriz simétrica aleatoria

```
function S = PdSMat(n)
    S = SymmetricMat(n);
    [P, D] = eig(S);
    D = abs(D);
    D = D + norm(D)*eye(size(D));
    S = P*D*inv(P);
end
```

Código 7: Matriz simétrica positiva definida aleatoria

```
times = zeros(2,length(1:50));

for i = 1:100
    n = i*10;
    A = PdSMat(n);
    b = ones(n,1);

    tic;
    [L U] = LUFacto(A);
    x1 = BackSub(U,ForwSub(L,b));
    times(1,i) = toc;

    tic;
    B = Cholesky(A);
    x2 = BackSub(B',ForwSub(B,b));
    times(2,i) = toc;
end

x = 10:10:1000;
p1 = polyfit(log(x),log(times(1,:)),1);
p2 = polyfit(log(x),log(times(2,:)),1);

figure
loglog(x,times(1,:),'*')
hold on
loglog(x,exp(p1(2)).*x.^p1(1),'--')
loglog(x,times(2,:),'o')
loglog(x,exp(p2(2)).*x.^p2(1),'--')
hold off
xlabel('Size of the matrix')
ylabel('Time of computation')
legend('LU-based solver','Best line for LU-solver','Cholesky-based solver','Best line
↪  for C-solver')
```

Código 8: Comparación de los tiempos de resolución para las distintas factorizaciones

Problema 4. *Sea A matriz de orden n con medio ancho de banda p. Para $n >> p >> 1$, calcular el número de operaciones $N_{\text{op}}(n,p)$ requeridas para la factorización LU. (Ver* (1)*.)*

Enfatizamos que el algoritmo (1) es una forma compacta de describir la eliminación Gaussiana. En la eliminación Gaussiana, requeriremos trabajar a lo más p operaciones fila de a lo más $p+1$ elementos, pues es la cantidad de elementos potencialmente no nulos a partir de la diagonal, en ambas direcciones. Debido a que sabemos que ocurre bajo el $k-$ésimo término de la diagonal, basta considerar los elementos que le suceden.

Es decir, para el paso $k-$ésimo del algoritmo, requerimos hacer operaciones fila desde la fila $i = k+1$ hasta la fila $i = \text{mín}(k+p, n)$. Dichas operaciones fila reqieren calcular el factor $a_{i,k}/a_{k,k}$, y tomar la diferencia puntual del $i-$ésimo vector fila contra $a_{i,k}/a_{k,k}$ veces el $k-$ésimo vector fila, que está soportado entre $j = k+1$ hasta $j = \text{mín}(k+p, n)$.

Tenemos que el número de operaciones es descrito por

$$N_{\text{op}}(n,p) = \sum_{k=1}^{n-1} \sum_{i=k+1}^{\text{mín}\{n,k+p\}} \left(1 + \sum_{j=k+1}^{\text{mín}\{n,k+p\}} 1 \right).$$

Vemos que $k+p \leq n$ si y solo si $k \leq n-p$. Esto nos permite separar la sumatoria. Tenemos

$$\begin{aligned}
N_{\text{op}}(n,p) &= \textstyle\sum_{k=1}^{n-1} (\text{mín}\{n, k+p\} - k)(\text{mín}\{n, k+p\} - k - 1) \\
&= \textstyle\sum_{k=1}^{n-p} p(p-1) + \sum_{k=n-p+1}^{n-1} (n-k)(n-k-1) \\
&= p(p-1)(n-p) + \textstyle\sum_{k=0}^{p-2} k(k+1) \\
&= p(p-1)(n-p) + \frac{1}{3} p(p-1)(p-2) \\
&= np^2 - p^3 + \frac{1}{3} p^3 + O(np + p^2) \\
&= np^2 - \frac{2}{3} p^3 + O(np),
\end{aligned}$$

pues $p^2 \leq np$ para $n >> p$.

```
e=1.E-16;
A=[e 1 1;1 -1 1; 1 0 1];
b=[2 0 1]';
[L, U] = LUFacto(A);
```

Código 9: Factorización LU mal comportada

```
[~ ~ p]=lu(A);
[l u] = LUFacto(p*A)
```

Código 10: Factorización LU con precondicionador

```
x1 = BackSub(U,ForwSub(L,b))
x2 = BackSub(u,ForwSub(l,p*b))
```

Código 11: Comparación soluciones de los métodos LU

```
for k=1:(n-1)
    for i=(k+1):min([n k+p])
        A(i,k) = A(i,k)/A(k,k);
        for j=(k+1):min([n k+p])
            A(i,j) = A(i,j) - A(i,k)*A(k,j);
        end
    end
end
```

Código 12: Fatorización LU para matriz de medio ancho de banda p