Universidad de Concepción
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Matemática

Cálculo Numérico (521230)

Laboratorio 3: Métodos directos para la solución de sistemas de ecuaciones lineales.

En clases vimos que ni el cálculo de A^{-1} ni la regla de Cramer son variantes óptimas para la solución numérica de sistemas de ecuaciones lineales por su alto costo operacional.

Ejercicio 1: Escriba una función en matlab, llamémosla `comparar_tiempos`, que reciba como parámetro de entrada un vector de números naturales `nvec` y devuelva una matriz de 3 columnas y tantas filas como componentes tenga `nvec`. La fila i-ésima de esta matriz debe contener los tiempos necesarios para calcular la solución al sistema de ecuaciones $Ax = b$ de tamaño `n = nvec(i)` mediante eliminación gaussiana (`x = A\b`), cálculo de la inversa de A (`x = inv(A)*b`) y mediante el método de Cramer. La matriz A y el vector b deben ser los siguientes:

$$A = \begin{pmatrix} 2 & -1 & 0 & 0 & \cdots & 0 & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ldots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 2 & -1 \\ 0 & 0 & 0 & 0 & \cdots & -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}.$$

Para crearlos puede usar los comandos `diag` y `ones` de MATLAB .

1.1 Llame a la función `comparar_tiempos` con entrada `10:10:100` y grafique en una misma figura los tiempos necesarios para resolver cada uno de los sistemas de ecuaciones lineales con los tres métodos considerados. Para ello, una vez escrita la función `comparar_tiempos`, usted deberá escribir en la ventana de comandos de matlab

```
>> tiempos = comparar_tiempos(10:10:100)
>> plot(tiempos(:,1),'o')  % graficar tiempos de eliminación gaussiana
>> hold on
>> plot(tiempos(:,2),'x')  % graficar tiempos de cálculo de inversa
>> plot(tiempos(:,3),'*')  % graficar tiempos de método de Cramer
>> legend('Gauss','Inversa','Cramer')
```

La siguiente función puede servirle como base para la función `comparar_tiempos`.

```
function [tiempos] = comparar_tiempos(nvec)
% funcion para comparar tiempos necesarios para resolver Ax=b
% mediante eliminacion gaussiana, cálculo de inversa y método de Cramer

% convirtiendo vector de entrada en vector columna con componentes enteras
nvec = floor(nvec(:));
% chequear que nvec sólo contenga números naturales
if ~isnumeric(nvec) || any(nvec <= 0) || any(isinf(nvec))
        error('entrada errónea')
end
tiempos = zeros(length(nvec),3);
for i = 1:length(nvec)
        n = nvec(i);
        % escribir aquí comandos para crear A y b

        % eliminacion gaussiana
        tic
        x = A\b;
        tiempos(i,1) = toc;
        % escriba aquí cálculo de tiempo para obtener
        % x mediante x = inv(A)*b y guárdelo en tiempos(i,2)
        % escriba aquí cálculo de tiempo para obtener x
        % mediante Cramer y guárdelo en tiempos(i,3)
end
```

Ejercicio 2: En muchas aplicaciones es necesario resolver varios sistemas de ecuaciones $Ax_i = b_i$, con la misma matriz $A \in \mathbb{R}^{n\times n}$ y distintas partes derechas $b_i \in \mathbb{R}^n$, $i = 1, \ldots, m$. Para hacer esto en MATLAB resulta conveniente generar la matriz de partes derechas

$$B = \left[\begin{array}{c|c|c} b_1 & \cdots & b_m \end{array}\right] \in \mathbb{R}^{n\times m}$$

y resolver el sistema matricial $AX = B$, cuya solución

$$X = \left[\begin{array}{c|c|c} x_1 & \cdots & x_m \end{array}\right] \in \mathbb{R}^{n\times m}$$

es la matriz de vectores solución x_i, $i = 1, \ldots, m$, de los sistemas anteriores.
El siguiente programa MATLAB tiene por objeto verificar esto experimentalmente:

```
% rutero para observar re-uso de
% descomposición LU cuando se resuelven
% sistemas con misma matriz A y distintas partes derechas
A=rand(50);
% comprobar que A es invertible
while rank(A) ~= 50
  A = rand(50);
end
B=rand(50,100);

t0=cputime;
X=A\B;
t1=cputime-t0
% incializar matriz con soluciones de sistemas
Y = zeros(50,100);
t0=cputime;
for i=1:100
  Y(:,i)=A\B(:,i);
end
t2=cputime-t0

dif=norm(X-Y,inf)
```

2.1 Escriba y ejecute el programa anterior.
2.2 Analice lo que hace este programa. (Note que el comando `cputime` entrega el tiempo de CPU utilizado desde que se inició MATLAB.)
2.3 Justifique la diferencia de tiempos de ejecución que se observa.
2.4 Utilice la sentencia MATLAB `X=A\B` con una matriz B adecuada para calcular A^{-1}. Compare el resultado obtenido con el del comando MATLAB `inv`.

Note que si en este ejemplo, en lugar de `cputime`, usamos `tic, toc` para medir los tiempos transcurridos, no obtendremos exactamente los mismos resultados (éstos serán similares, pero no iguales). MATLAB recomienda el uso de `tic, toc` teniendo en cuenta que, para una mejor medición del tiempo transcurrido, ninguna otra aplicación debería estar corriendo en el computador.
Ejercicio 3: Comprobemos que `A\b` es equivalente a calcular la descomposición LU de A con pivoteo parcial. Si A es estrictamente diagonal dominante no será necesario realizar permutaciones de filas al calcular su descomposición LU con pivoteo parcial, es decir, $P = I$.

3.1 Descomposición LU de matriz estrictamente diagonal dominante.

3.1.1 Haga un programa *function* que genere una matriz tridiagonal de orden n de la forma

$$A = \begin{pmatrix} a & b & & 0 \\ c & \ddots & \ddots & \\ & \ddots & \ddots & b \\ 0 & & c & a \end{pmatrix}.$$

Los datos de entrada deben ser los valores de n, a, b y c, y la salida la matriz A.

3.1.2 Mediante el comando MATLAB `rank` (devuelve el rango de una matriz), determine si la matriz tridiagonal y simétrica A correspondiente a $n = 10$, $a = 4$ y $b = c = 1$ es invertible. Note que con estos valores de a, b, c la matriz A es estrictamente diagonal dominante.

3.1.3 Resuelva el sistema $Ax = b$ con la matriz tridiagonal anterior y un segundo miembro b cualquiera, de los modos siguientes:

1) mediante `x = A\b`,
2) mediante

```
>> [L,U,P] = lu(A)
>> x = U\(L\b)
```

Compruebe que $P = I$ y $A = LU$.

3.1.4 Compare las soluciones obtenidas y calcule los residuos $r = b - Ax$ respectivos.

3.2 Descomposición LU de matriz A que no es diagonal dominante.

3.2.1 Con el comando `rand` genere una matriz A de tamaño 10 y un vector $b \in \mathbb{R}^{10}$.

3.2.2 Mediante el comando MATLAB `rank` (devuelve el rango de una matriz), determine si A es invertible. Si no lo es, genere una nueva matriz con `rand` hasta obtener una matriz invertible.

3.2.3 Resuelva el sistema $Ax = b$ con la matriz A y el vector b generados de los modos siguientes:

1) mediante `x = A\b`,
2) mediante

```
>> [L,U,P] = lu(A)
>> x = U\(L\(P*b))
```

Compruebe que $PA = LU$.

3.2.4 Compare las soluciones obtenidas y calcule los residuos $r = b - Ax$ respectivos.

Ejercicio 4: Sean

$$A = \begin{bmatrix} n+1 & 1 & \cdots & 1 \\ 1 & \ddots & \ddots & 1 \\ \vdots & \ddots & \ddots & 1 \\ 1 & \cdots & 1 & n+1 \end{bmatrix} \in \mathbb{R}^{n\times n} \qquad \text{y} \qquad b = \begin{bmatrix} 1 \\ \vdots \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^n.$$

Haga un programa MATLAB que genere la matriz anterior para $n = 10$.

4.1 Compruebe que A es simétrica y definida positiva (puede calcular los valores propios de A con el comando `eig` de MATLAB y comprobar que ellos son mayores que cero o puede demostrar que para todo $x \in \mathbb{R}^{10}$ se cumple $x^{\mathrm{t}}Ax > 0$).

4.2 Encuentre mediante el comando `chol` la descomposición de Cholesky de A.

4.3 Resuelva, con ayuda de esta descomposición, el sistema $Ax = b$.