UNIVERSIDAD DEL BÍO-BÍO
FACULTAD DE CIENCIAS
DEPARTAMENTO DE MATEMÁTICA

Métodos Numéricos 220138

Laboratorio 4

Sistemas de Ecuaciones Lineales

Sistema de ecuaciones lineales: Métodos directos

1. El objetivo de este laboratorio es aprender a utilizar eficientemente métodos directos para la solución de sistemas de ecuaciones lineales $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$.

   (1.1) Haga un programa *function* que genere una matriz tridiagonal de orden n de la forma

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

   Los datos de entrada deben ser los valores de n, a, b y c, y la salida la matriz $\mathbf{A}$.

   (1.2) Mediante el comando MATLAB *rank*, determine si la matriz tridiagonal y simétrica $\mathbf{A}$ correspondiente a $n = 10$, $a = 4$ y $b = c = 1$ es no singular.

   (1.3) Resuelva el sistema $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ con la matriz tridiagonal anterior y un segundo miembro $\boldsymbol{b}$ cualquiera, de los siguientes dos modos:

   1.3.1.
   ```
   >> x=A\b
   ```

   1.3.2.
   ```
   >> [L,U,P]=lu(A);
   >> x=U\(L\(P*b))
   ```

   Compare las soluciones obtenidas y calcule los residuos $\boldsymbol{r} = \boldsymbol{b} - \mathbf{A}\boldsymbol{x}$ respectivos.

   Observaciones: Note que el comando `\` de MATLAB resuelve el sistema de ecuaciones lineales y funciona de forma similar a la inversa de la matriz A. Por otro lado, la sintaxis `[L,U,P]=lu(A)` encuentra la factorización LU con pivoteo parcial de la matriz A. para más información, mira el `help` de MATLAB.

2. En muchas aplicaciones es necesario resolver varios sistemas de ecuaciones $\mathbf{A}\boldsymbol{x}_i = \boldsymbol{b}_i$, con la misma matriz $\mathbf{A} \in \mathbb{R}^{n\times n}$ y distintos segundos miembros $\boldsymbol{b}_i$, $i = 1, \ldots, m$. Para hacer esto en MATLAB resulta conveniente generar la matriz de segundos miembros

   $$\mathbf{B} = \left[\begin{array}{c|c|c} \boldsymbol{b}_1 & \cdots & \boldsymbol{b}_m \end{array}\right] \in \mathbb{R}^{n\times m}$$

   y resolver el sistema matricial $\mathbf{AX} = \mathbf{B}$, cuya solución

   $$\mathbf{X} = \left[\begin{array}{c|c|c} \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_m \end{array}\right] \in \mathbb{R}^{n\times m}$$

es la matriz de vectores solución $\boldsymbol{x}_i$, $i = 1, \ldots, m$, de los sistemas anteriores.

El siguiente programa MATLAB tiene por objeto verificar esto experimentalmente:

```
A=rand(50);
B=rand(50,100);

t0=cputime;
X=A\B;
t1=cputime-t0

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 $\mathbf{B}$ adecuada para calcular $\mathbf{A}^{-1}$. Compare este resultado con el resultado obtenido al ejecutar en MATLAB la sintaxis `X=inv(A)`.

3. El objeto de este ejercicio es demostrar que **no es conveniente** resolver un sistema de ecuaciones mediante la inversa de la matriz del mismo.

   Considere el siguiente programa MATLAB:

```
nn=[100:100:500];
t1=[];
t2=[];
for n=nn
   A=rand(n);
   y=ones(n,1);

   t0=cputime;
   x=inv(A)*y;
   t1=[t1 cputime-t0];

   t0=cputime;
   x=A\y;
   t2=[t2 cputime-t0];
end
plot(nn,t1,'*-',nn,t2,'o-')
```

(3.1) Analice lo que hace este programa, ejecútelo e indique qué alternativa es la más conveniente.

4. Sean

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

(4.1) Hacer un programa MATLAB que:

4.1.1. genere la matriz anterior para $n = 10$;

4.1.2. calcule una matriz $\mathbf{R}$ tal que $\mathbf{A} = \mathbf{R}^t\mathbf{R}$ y resuelva mediante el método de Cholesky el sistema $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$, para ello utilice la sintaxis de Matlab

```
>> R=chol(A);
>> x=R\(R'\b)
```

(4.2) ¿Es la matriz $\mathbf{A}$ definida positiva? Justifique su respuesta.

(4.3) Como ejercicio adicional, ejecute la sintaxis anterior ingresando una matriz que no sea definida positiva. Observe qué sucede.

5. Las matrices de *Hilbert*

$$\mathbf{H}_n = \left(h_{ij}^n\right) \in \mathbb{R}^{n\times n}, \qquad \text{con } h_{ij}^n = \frac{1}{i+j-1}, \quad i,j = 1,\ldots,n,$$

son matrices muy mal condicionadas. Estas matrices se generan en MATLAB con el comando *hilb*.

(5.1) Tabule los números de condición en norma euclideana (comando *cond*) y las estimaciones de los números de condición en norma 1 (comando *condest*) de estas matrices, para $n = 2,\ldots,10$.

(5.2) Sean

$$\boldsymbol{b}_0 = \begin{pmatrix} 0.7487192 \\ 0.4407175 \\ 0.3206968 \\ 0.2543113 \\ 0.2115308 \\ 0.1814429 \end{pmatrix} \quad \text{y} \quad \boldsymbol{b}_1 = \boldsymbol{b}_0 + \delta\boldsymbol{b},$$

con $\delta\boldsymbol{b}$ un vector de perturbaciones aleatorias de valor absoluto menor o igual a 10^{-6} (Note que la sentencia MATLAB `(2*rand(n,1)-1)*a` genera un vector columna aleatorio de dimensión n, uniformemente distribuido en el intervalo $[-a, a]$).

Resuelva los sistemas $\mathbf{H}_6\boldsymbol{x}_0 = \boldsymbol{b}_0$ y $\mathbf{H}_6\boldsymbol{x}_1 = \boldsymbol{b}_1$. Compare la diferencia de las soluciones $\delta\boldsymbol{x} = \boldsymbol{x}_1 - \boldsymbol{x}_0$ con la de los segundos miembros $\delta\boldsymbol{b}$. Describa lo qué se observa.

(5.3) Verifique que se satisface la relación

$$\frac{\|\delta\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \le \text{cond}(\mathbf{H}_6)\frac{\|\delta\boldsymbol{b}\|}{\|\boldsymbol{b}\|}.$$

Sistema de ecuaciones lineales: Métodos iterativos

Un método iterativo para resolver el sistema de ecuaciones $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$, con la matriz $\mathbf{A} \in \mathbb{R}^{n\times n}$ y el vector columna $\boldsymbol{b} \in \mathbb{R}^n$ dados, comienza con una aproximación inicial $\boldsymbol{x}^{(0)}$ de la solución $\boldsymbol{x}$ y genera una sucesión de vectores $\{\boldsymbol{x}^{(k)}\}_{k=0}^{\infty}$ que converge a $\boldsymbol{x}$. Los métodos iterativos traen consigo un proceso que convierte el sistema $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ en otro equivalente de la forma $\boldsymbol{x} = \mathbf{T}\boldsymbol{x} + \boldsymbol{c}$ para alguna matriz fija $\mathbf{T}$ y un vector $\boldsymbol{c}$.

Luego de seleccionar el vector inicial $\boldsymbol{x}^{(0)}$, la sucesión de los vectores de la solución aproximada se genera calculando

$$\boldsymbol{x}^{(k)} = \mathbf{T}\boldsymbol{x}^{(k-1)} + \boldsymbol{c}\,,$$

para cada $k = 1, 2, 3, \ldots$. Para detener el proceso, una forma sencilla consiste en dar una tolerancia deseada ε y parar el ciclo o bucle cuando $\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\| < \varepsilon$ en caso de convergencia, o bien finalizar cuando se ha realizado un número máximo de iteraciones, por si llegase haber divergencia del método.

1. Hacer un programa `function` en MATLAB que genere el proceso iterativo de **Jacobi** para aproximar la solución del sistema $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$. La `function` debe tener los parámetros: El número de incógnitas `n` del sistema, la matriz `A`, el vector `b`, la tolerancia para el error `tol` y el numéro máximo de iteraciones `maxiter`. Debe entregar como respuesta el vector de solución `sol` con el numéro de iteraciones `iter` realizadas en caso que el esquema converga antes que se cumpla en número máximo de iteraciones. En caso contrario, debe entregar por respuesta un mensaje que no se obtuvo la convergencia deseada, además debe dejar en memoria (sin mostrar en pantalla) el vector solución `sol` y el error `err` que se lleva hasta ese momento.

2. Hacer exactamente lo mismo pero para el método de **Gauss-Seidel**.

Otro objetivo de este laboratorio es aprender a utilizar eficientemente el correcto tratamiento de matrices dispersas (*sparse*).

1. (1.1) Haga un programa *function* que genere una matriz de la forma

$$\mathbf{B} = \begin{pmatrix} 2\,\mathbf{I} & -\mathbf{I} \\ -\mathbf{I} & 2\,\mathbf{I} \end{pmatrix} \in \mathbb{R}^{2n\times 2n}$$

para $n = 100$. Compruebe que $\mathbf{B}$ es una matriz dispersa mediante el comando `nnz` (*Number of Non Zeros*) que da la cantidad total de entradas no nulas de la matriz.

(1.2) Utilice la siguiente sentencia para almacenar en forma *sparse* otra matriz $\mathbf{A}$ igual a $\mathbf{B}$:

```
A=sparse(B);
```

Utilice el comando `whos` para determinar la cantidad de memoria que requiere el almacenamiento de cada una de esas matrices. Indique cuál resulta más conveniente.

(1.3) Genere un vector aleatorio $\mathbf{b} \in \mathbb{R}^{2n}$ y compare los tiempos necesarios para resolver los sistemas $\mathbf{B}\boldsymbol{x} = \mathbf{b}$ y $\mathbf{A}\boldsymbol{x} = \mathbf{b}$. Indique cuál resulta más conveniente.

(1.4) Hay varios comandos que tienen su contraparte para matrices dispersas. Por ejemplo, `speye` es semejante a `eye`, pero la matriz identidad que genera se almacena como *sparse*.
Al realizar operaciones con matrices *sparse* (por ejemplo, suma, producto, traspuesta, concatenación, etc.), MATLAB almacena los resultados en nuevas matrices *sparse*.
Así mismo, muchos comandos, cuando se aplican a matrices *sparse*, generan matrices *sparse*. Por ejemplo, `diag`, `tril` y `triu`.
Utilice estos comandos para obtener las matrices $\mathbf{D}$, $\mathbf{L}$ y $\mathbf{U}$ de la descomposición $\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{U}$. Calcule las matrices de iteración de los métodos de **Jacobi** $\mathbf{T}_J = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})$ y de **Gauss-Seidel** $\mathbf{T}_G = (\mathbf{D} - \mathbf{L})^{-1}\mathbf{U}$. Compruebe que las matrices $\mathbf{D}$, $\mathbf{L}$, $\mathbf{U}$, $\mathbf{T}_J$ y $\mathbf{T}_G$ también se almacenan como matrices *sparse*.

(1.5) El comando `spy` permite "espiar" la estructura de una matriz mediante un gráfico con la distribución de sus entradas no nulas. "Espíe" la estructura de las matrices $\mathbf{A}$, $\mathbf{D}$, $\mathbf{L}$, $\mathbf{U}$, $\mathbf{T}_J$ y $\mathbf{T}_G$

(1.6) Si bien el almacenamiento de la matriz $\mathbf{A}$ requiere mucha menos memoria que el de la matriz $\mathbf{B}$, para generar $\mathbf{A}$ como se ha descrito resulta necesario haber almacenado previamente $\mathbf{B}$.
A veces, la memoria del computador no alcanza para almacenar $\mathbf{B}$ en forma llena. En tal caso, es necesario generar directamente la matriz dispersa $\mathbf{A}$ sin pasar nunca por la matriz llena $\mathbf{B}$. Indique alguna forma de hacer esto y compruébelo.

(1.7) Para algunas aplicaciones es necesario dar la forma llena de una matriz dispersa. El comando inverso de `sparse` que almacena una matriz dispersa en forma de matriz llena es `full`. Compruébelo.