UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE CIENCIAS
FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MATEMÁTICA

Cálculo Numérico (521230) - Laboratorio 7

SISTEMAS DE ECUACIONES LINEALES

1. Sistema de ecuaciones

Un sistema de ecuaciones lineales:

$$\text{Hallar } x \in \mathbb{R}^n: \quad \sum_{j=1}^{n} a_{ij}x_j = b_i, \quad \forall i \in \{1, \cdots, n\},$$

se puede resumir en su forma matricial como:

$$\text{Hallar } x \in \mathbb{R}^n: \quad Ax = b, \quad A \in \mathcal{M}_{n\times n}(\mathbb{R}), b \in \mathbb{R}^n,$$

donde A es llamada matriz de coeficientes del sistema y b es llamado vector de datos o del lado derecho. Por ejemplo el sistema

$$\left.\begin{array}{rl} 2x + 3y + 4z & = 7 \\ 6x + 8y + 9z & = 1 \\ 8x + 11y + 13z & = 9 \end{array}\right|,$$

se representa como

$$\begin{bmatrix} 2 & 3 & 4 \\ 6 & 8 & 9 \\ 8 & 11 & 13 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 7 \\ 1 \\ 9 \end{bmatrix},$$

especial cuidado deben tener los órdenes (número de filas y columnas) en la notación matricial, puesto estos deben coincidir con la estructura necesaria para que la operación **producto matricial** tenga sentido.

La búsqueda de tales soluciones nos ha presentado varios algoritmos numéricos para este tipo de problemas, categorizables en **directos** e **iterativos**

`size(A)`	`[n,m]=size(A)` retorna el orden $n \times m$ de una matriz A
`eye(n,m)`	Retorna una matriz de orden $n \times m$ que en su diagonal principal tiene sólo el número 1.
`diag(A,M)`	Retorna una matriz cuya diagonal es la $m-$ésima columna de A.
`ones(n,m)`	Retorna una matriz de puros unos de orden $n \times m$.
`cross(A,B)`	Producto cruz a lo largo de los vectores componentes de las matrices A,B
`dot(A,B)`	Producto punto a lo largo de los vectores componentes de las matrices A,B
`tril(A)`	Matriz triángulo inferior de la matriz A
`triu(A)`	Matriz triángulo superior de la matriz A
`transpose(A)`	Matriz transpuesta de A igual que la instrucción `A'`
`rand(n,m)`	Genera una matriz con entradas aleatorias de orden $n \times m$

1.1. Operatoria Matricial en Matlab

Existen muchas instrucciones para cargar matrices en Matlab,

```
 >> A=pascal(4)

A =

     1     1     1     1
     1     2     3     4
     1     3     6    10
     1     4    10    20

>> B=magic(3)

B =

     8     1     6
     3     5     7
     4     9     2

>> C=eye(4)

C =

     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1

>> D=diag(1:0.1:1.3)

D =

    1.0000         0         0         0
         0    1.1000         0         0
         0         0    1.2000         0
         0         0         0    1.3000
```

Observe lo que sucede cuando se ejecuta la función `diag` sobre un vector.

1.1.1. Adición y sustracción de matrices

Los operadores `+` y `-` realizan la operación de suma y resta matricial

```
>> E=diag(1:5)+ones(5)

E =

     2     1     1     1     1
     1     3     1     1     1
     1     1     4     1     1
     1     1     1     5     1
     1     1     1     1     6

>> C=E+E

C =

     4     2     2     2     2
     2     6     2     2     2
     2     2     8     2     2
     2     2     2    10     2
     2     2     2     2    12
```

Esta operación debe hacerse entre sumandos del mismo órden

```
>> F=ones(1,3)+diag(1:4)
Error using  +
Matrix dimensions must agree.

>> diag(eye(3))+[1:3]
Error using  +
Matrix dimensions must agree.
```

1.1.2. Productos matriciales en Octave

El operador `*` representa el producto matricial, el operador `.*` representa el producto componente a componente. Por ejemplo, en el producto punto

```
>> a=[1:3];b=[2:4];
>> ( a*b'==dot(a,b) ) && (b*a'==sum(a.*b))

ans =

     1
```

Análogamente a la suma, la operación producto debe ser ejecutada entre operandos válidos

```
>> A=rand(3,4)*rand(1,4)
Error using  *
Inner matrix dimensions must agree.
```

```
>> A=[1:3].*ones(3,1)
Error using  .*
Matrix dimensions must agree.
```

```
>> A=ones(3,2)*eye(3)
Error using  *
Inner matrix dimensions must agree.
```

```
>> A=diag(1:4).*ones(4,1)
Error using  .*
Matrix dimensions must agree.
```

De la misma forma existen los operadores `^` y `.^`, para representar la potenciación de matrices,

```
>> A=rand(2,2);
>> A*A==A^2

ans =

     1     1
     1     1
```

```
>> B=diag(1:3);
>> B==B.^2

ans =

     1     1     1
     1     0     1
     1     1     0
```

1.2. Métodos de factorización de matrices en Octave

Algunos de los métodos de factorización de matrices en Octave son

`chol(A)`	Factorización de Cholesky de A, produce una matriz triangular superior `R` tal que `R'*R=A`, `A` debe ser definida positiva.
`lu(A)`	Factorización LU de una matriz A utilizando eliminacin Gaussiana con pivoteo parcial mediante la instrucción `[L,U,P]=lu(A)`
`qr(A)`	Factorización QR de una matriz A, `[Q,R] = qr(A)`, donde A es de $m \times n$,produce una matriz `Q` triangular superior de $m \times n$ y `R` una matriz unitaria de orden m tal que `A = Q*R`.

1.3. Métodos de resolución de sistemas de ecuaciones lineales en Octave

Las siguientes funciones vienen incluídas en Octave

`rank(A)`	Entra una estima del número de filas o columnas de A que son linealmente independientes.
`cond(A)`	Número de condición utilizando la matriz inversa
`condest(A)`	Número de condición utilizando la norma 1
`inv(A)`	Matriz inversa de A

`mldivide(A,b)`	Resuelve el sistema de ecuaciones lineales $Ax = b$, si el sistema es sobredeterminado o subdeterminado busca soluciones en el sentido de mínimos cuadrados. Se entrega una alerta si A es mal condicionada. Se puede ejecutar también como `A\b`

1.3.1. Ejemplos de uso

1. Cálculo del número de condición de una matriz

```
>> cond(eye(10))
ans =  1
>> cond([eye(10),zeros(10,1);...
zeros(1,11)])
ans = Inf
```

2. Resuelve un sistema de ecuaciones lineales simple

```
A = magic(3);
B = [15; 15; 15];
x = A\B
```

3. Resuelve un sistema de ecuaciones con una matriz singular

```
>> A=[1,2;2,4];
>> x=A\[1;3]
warning: matrix singular to machine precision
x =

   0.28000
   0.56000
```

4. Matrices singulares con problemas de precisión

```
>> A = [1 0; 0 0];
>> b = [1; 1];
>> x = A\b
warning: matrix singular to machine precision
x =

   1
   0
```

5. Cálculo de la inversa de una matriz

```
>> inversa=inv(diag(1:4))
inversa =

Diagonal Matrix

   1.00000         0         0         0
         0   0.50000         0         0
         0         0   0.33333         0
         0         0         0   0.25000

>> inversa*diag(1:4)
ans =

Diagonal Matrix

   1   0   0   0
   0   1   0   0
   0   0   1   0
   0   0   0   1
```

6. Solución de un sistema usando factorización QR

```
>> A=[1,2,3;0,1,1;0,0,1];
>> [Q,R]=qr(A)
Q =

   1   0   0
   0   1   0
   0   0   1

R =

   1   2   3
   0   1   1
   0   0   1

>> x=Q\R\[1;1;1]
x =

  -2
   0
   1
```

7. Solución de un sistema usando método de Cholesky

```
>> A=gallery('moler',5)
A =

   1  -1  -1  -1  -1
  -1   2   0   0   0
  -1   0   3   1   1
  -1   0   1   4   2
  -1   0   1   2   5

>> C=chol(A)
C =

   1  -1  -1  -1  -1
   0   1  -1  -1  -1
   0   0   1  -1  -1
   0   0   0   1  -1
   0   0   0   0   1


>> isequal(C'*C,A)
ans =  1
```

8. Descomposición LU de una matriz

```
>> [L,U]=lu(vander(1:4))
L =

   0.01562   0.33333   1.00000   0.00000
   0.12500   0.88889   0.66667   1.00000
   0.42188   1.00000   0.00000   0.00000
   1.00000   0.00000   0.00000   0.00000

U =

  64.00000  16.00000   4.00000   1.00000
   0.00000   2.25000   1.31250   0.57812
   0.00000   0.00000   0.50000   0.79167
   0.00000   0.00000   0.00000  -0.16667

>> L*U
ans =

    1    1    1    1
    8    4    2    1
   27    9    3    1
   64   16    4    1
```

9. Descomposición LU de una matriz singular

```
>> A=[1,2,3;4,5,6;5,7,9];

>> det(A)
ans =    2.6645e-15

>> [L,U]=lu(A)
L =

   0.20000  -1.00000   1.00000
   0.80000   1.00000   0.00000
   1.00000   0.00000   0.00000

U =

   5.00000   7.00000   9.00000
   0.00000  -0.60000  -1.20000
   0.00000   0.00000   0.00000
```

2. Métodos iterativos para la resolución de sistemas de ecuaciones lineales

En la teoría sabemos que es posible usar un esquema iterativo que mejore una solución de un sistema hasta que se obtenga una convergencia deseada. A continuación se muestran los códigos en Matlab de los dos métodos estudiados en clases.

2.1. Método de Jacobi

El sistema con diagonal sin ceros y dominante

$$Ax = b,$$

se descompone en

$$(L + D + U)x = b,$$

y se propone el esquema iterativo, dado $x^0 \in \mathbb{R}^n$

$$x^{k+1} = D^{-1}(-(L+U)x^k + b)$$

```
function x=jacobi(A,b,x0,maxit)

x=x0;
for k=1:maxit
    x=diag(diag(A))\(b-(triu(A,1)+tril(A,1))*x);
    if norm(A*x-b,2)<tol
        return
    end
end
```

2.2. Método de Gauss-Seidel

Las filas con elemento diagonal no nulo del sistema $Ax = b$ se dividen por su elemento diagonal transformando el sistema en

$$(L + I + U)x = b,$$

y se propone el esquema iterativo, dado $x^0 \in \mathbb{R}^n$

$$x^{k+1} = -(L+U)x^k + b$$

se puede probar que cuando el método de Jacobi converge, el método de Gauss-Seidel lo hace más rápido.

```
function x=gauss_seidel(A,b,x0,tol,maxit)
DE=tril(A);
F=A-DE
x=x0;
for k=1:maxit
    x = DE\(b-F*x);
    if norm(A*x-b,2)<tol
        return
    endif
endfor
```

3. Ejercicios

1. Alicia compra tres manzanas, una docena de plátanos y una chirimoya en $3700. Jacinta compra una docena de manzanas y dos chirimoyas en $8000, Carolina compra dos plátanos y tres chirimoyas en $3200. ¿Cuanto cuesta cada fruta?.

2. Usando los algoritmos iterativos de Jacobi y Gauss-Seidel, dibuje el conjunto de las primeras 10 aproximaciones de la solución de cada método y la solución exacta del sistema

$$\begin{aligned} 3x + 4y + 6z &= 1 \\ 4x + 9y + z &= 2 \\ 2x + y &= 0 \end{aligned}$$

   Hint: reordene el sistema para que la matriz de coeficientes sea diagonal dominante.

3. Para $n = 100$ resuelva el sistema de ecuaciones

$$\begin{array}{ll} 2x_1 - x_2 & = 1 \\ -x_j + 2x_j - x_{j+1} & = j, \quad \forall j \in \{2, \cdots, n-1\} \\ -x_{n-1} + 2x_n & = 1 \end{array}$$

   a) Use el comando `diag()` tres veces para generar la matriz de coeficientes del sistema.

   b) Utilize factorización **LU** para solucionar el problema.

   c) Utilize el operador \ para resolver el sistema.

   d) Utilize los métodos de Jacobi y Gauss-Seidel para resolver el sistema.

   e) Utilize `condest()` para estimar el número de coeficiente de la matriz del problema.

   f) Modifique su rutero para solucionar el problema en función del número de incógnitas n y grafique el número de condición en función del número de incógnitas.

4. ¿Qué crees que debe ser siempre verdad sobre los valores `rank(A)` y `rank(A')` para cualquier matriz `A`?.

5. Sabemos que la transformación lineal que transforma $\hat{i}$ en $\hat{i} + \hat{j}$ y $\hat{j}$ en $\hat{i}$ tiene por matriz que la representa a

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}.$$

   Cree un rutero que permite graficar la imagen del conjunto $\{(x, y) : x^2 + y^2 < 1\}$ a través de esta transformación lineal.

Revisado a Semestre 2018–2