PRÁCTICA 5: Sistemas de Ecuaciones Lineales. Métodos Directos

5.1 Resumen de Teoría

5.1.1 Introducción

La modelización de problemas de la Física, Mecánica, etc. conducen en muchas ocasiones, a la resolución de un sistema de ecuaciones lineales de dimensión finita. Además, otros problemas del Análisis Numérico como los de aproximación, interpolación, etc. recurren a la resolución de sistemas. Es por ello fundamental desarrollar métodos numéricos eficientes que obtengan la solución de sistemas de ecuaciones lineales, de lo que se ocupará esta práctica.

Estudiaremos la resolución de sistemas de *n* ecuaciones lineales con *n* incógnitas:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1j} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2j} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ a_{i1} & a_{i2} & a_{i3} & \dots & a_{ij} & \dots & a_{in} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nj} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{bmatrix}$$

los cuales representaremos en forma matricial como:

$$Ax = b$$

Donde:

$A = (a_{ij})_{n\times n}$ es la llamada matriz de coeficientes.
$x = (x_i)_{1\times n}$ es el llamado vector de incógnitas.
$b = (b_i)_{1\times n}$ es el llamado vector de términos independientes.

Supondremos que *A* es una matriz cuadrada, de orden *n*, no singular. Una primera idea que podemos aplicar para obtener la solución del sistema es calcularla como $x = A^{-1}b$, donde con A^{-1} representamos la inversa de la matriz *A*.

Otra manera de resolver este problema sería utilizar la regla de Cramer. Se puede demostrar que en ambos casos el número de operaciones a realizar sería del orden de n^2n!

En la práctica el tiempo de cálculo necesario es exagerado, incluso para un ordenador, y además, los errores de redondeo pueden llegar a ser tan grandes que invaliden el resultado final. Por todo lo anterior parece necesario construir otro tipo de métodos que sean más eficientes.

Los métodos que vamos a construir se agrupan en dos clases:

- **Métodos Directos**: persiguen la obtención de la solución exacta del sistema al cabo de un número finito de operaciones elementales. En dichos métodos no existe error de truncamiento, pero sí son sensibles a los errores de redondeo.
- **Métodos Iterativos**. Los métodos iterativos tienen como objetivo la construcción de una sucesión de vectores cuyo límite sea la solución del sistema.

Dentro de los métodos directos destacaremos el *método de Gauss*, así como aquellos que permiten factorizar la matriz *A* como producto de una matriz triangular inferior *L* y una matriz triangular superior *U*, entre los que destaca el método *LU* y el de *Cholesky*.

Respecto de los métodos iterativos, centraremos nuestro estudio en aquellos que resultan ser una generalización del proceso de iteración del punto fijo, como son el *método de Jacobi* y el de *Gauss-Seidel*. Estos métodos los estudiaremos en la Práctica 6.

En la Práctica 6 también veremos métodos numéricos que permitan aproximar la raíz de un sistema de ecuaciones no lineales. Dichos métodos son una generalización de los vistos para el caso de una variable. En particular, consideraremos el *método de Newton-Raphson*.

5.1.2 Método de Eliminación de Gauss

Comencemos repasando un par de definiciones previas. Una matriz *A* se dice *triangular superior* (respectivamente *triangular inferior*) si todos los elementos situados bajo (respectivamente sobre) la diagonal principal son nulos, es decir $a_{ij} = 0$ si $i > j$ (respectivamente $a_{ij} = 0$ si $i < j$). Un sistema de ecuaciones se dice triangular superior (respectivamente triangular inferior) si su matriz de coeficientes lo es.

Consideremos el siguiente sistema triangular superior:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1i} \dots & a_{1n} \\ 0 & a_{22} & a_{23} & \dots & a_{2i} \dots & a_{2n} \\ 0 & 0 & a_{33} & \dots & a_{3i} \dots & a_{3n} \\ \vdots & \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & & a_{ii} & a_{in} \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{pmatrix}$$

en el cual supondremos que: $|A| \neq 0,$ es decir: $a_{ii} \neq 0, \forall$i. Para resolver un sistema triangular superior se utiliza el procedimiento conocido como *sustitución regresiva*, que podemos expresar en la forma:

$$x_n = \frac{b_n}{a_{nn}}\,;\; x_i = \left(b_i - \sum_{j=i+1}^{n} a_{ij} x_j \right) / a_{ii} \;\;;\;\; i = n-1, n-2, \ldots, 1$$

Consideremos ahora el siguiente sistema triangular inferior**:**

$$\begin{pmatrix} a_{11} & 0 & 0 & \dots & 0 \\ a_{21} & a_{22} & 0 & \dots & 0 \\ a_{31} & a_{32} & a_{33} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{i1} & a_{i2} & \cdots & a_{ii} & \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{pmatrix}$$

La solución del sistema triangular inferior se obtiene por el procedimiento conocido como *sustitución progresiva*:

$$x_1 = \frac{b_1}{a_{11}};\ x_i = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j \right) / a_{ii}\ ;\ \ i = 2,...,n$$

Teniendo en cuenta que estos sistemas son de fácil solución y que el número de operaciones realizadas para calcular tales soluciones (n^2) es sensiblemente menor que el que resulta utilizando otras técnicas, vamos a intentar reducir el problema general a la resolución de un sistema triangular, siendo ésta la esencia de los métodos directos.

Estudiaremos en primer lugar el método de Gauss, basado en la transformación de la matriz *A* de partida en una matriz triangular superior a través de *n*-1 etapas, en cada una de las cuales se va consiguiendo dejar en forma triangular una fila más. Para ello hacemos ceros por debajo de la diagonal principal, usando como pivotes los elementos de dicha diagonal, por lo que, de ser alguno nulo, debe buscarse algún elemento no nulo por debajo del mismo y, una vez encontrado, proceder al intercambio de filas. En la primera etapa, que denotamos con superíndice 1:

$$\begin{array}{ccccccccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & a_{13}x_3 & + & \cdots & + & a_{1i}x_i & + & \cdots & + & a_{1n}x_n & = & b_1 \\ & & a_{22}^{(1)}x_2 & + & a_{23}^{(1)}x_3 & + & \cdots & + & a_{2i}^{(1)}x_i & + & \cdots & + & a_{2n}^{(1)}x_n & = & b_2^{(1)} \\ & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ & & a_{i2}^{(1)}x_2 & + & a_{i3}^{(1)}x_3 & + & \cdots & + & a_{ii}^{(1)}x_i & + & \cdots & + & a_{in}^{(1)}x_n & = & b_i^{(1)} \\ & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ & & a_{n2}^{(1)}x_2 & + & a_{n3}^{(1)}x_3 & + & \cdots & + & a_{ni}^{(1)}x_i & + & \cdots & + & a_{nn}^{(1)}x_n & = & b_n^{(1)} \end{array}$$

con: $a_{ij}^{(1)} = a_{ij} - \left(\frac{a_{i1}}{a_{11}} \right) a_{1j}$

Aplicando reiteradamente (*n*-1 etapas) este proceso llegamos a:

$$\begin{array}{ccccccccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & a_{13}x_3 & + & \cdots & + & a_{1i}x_i & + & \cdots & + & a_{1n}x_n & = & b_1 \\ & & a_{22}^{(1)}x_2 & + & a_{23}^{(1)}x_3 & + & \cdots & + & a_{2i}^{(1)}x_i & + & \cdots & + & a_{2n}^{(1)}x_n & = & b_2^{(1)} \\ & & & & & & & & \vdots & & \vdots & & \vdots & & \vdots \\ & & & & & & & & a_{ii}^{(i-1)}x_i & + & \cdots & + & a_{in}^{(i-1)}x_n & = & b_i^{(i-1)} \\ & & & & & & & & \vdots & & \vdots & & \vdots & & \vdots \\ & & & & & & & & & & & & a_{nn}^{(n-1)}x_n & = & b_n^{(n-1)} \end{array}$$

Donde:

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} \; ; \; 1 \leq i \leq k, \; 1 \leq j \leq n$$

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \left(\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \right) a_{kj}^{(k)} \; ; \; k+1 \leq i \leq n, \; k+1 \leq j \leq n$$

$$a_{ij}^{(k+1)} = 0 \; ; \; k+1 \leq i \leq n, \; 1 \leq j \leq k$$

En lo que refiere a los términos independientes, las componentes se obtendrán de la siguiente forma:

$$b_i^{(k+1)} = b_i^{(k)} \; ; \; 1 \leq i \leq k$$

$$b_i^{(k+1)} = b_i^{(k)} - \left(\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \right) b_k^{(k)} \; ; \; k+1 \leq i \leq n$$

Una vez realizada la eliminación Gaussiana, utilizamos el algoritmo del sistema triangular superior para solucionar completamente el sistema de ecuaciones.

Como vemos el algoritmo que hemos presentado costa de dos fases: la eliminación hacia adelante y la sustitución hacia atrás. Se le suele llamar *eliminación Gaussiana "simple"* porque como se mencionó antes, ocurren problemas obvios cuando un elemento pivote es cero, ya que el paso de normalización origina una división entre cero. Para obtener un algoritmo más fiable se requiere de algunas modificaciones que eviten dicha división entre cero. Dos son las técnicas más empleadas:

- *Pivoteo parcial*: Las filas se intercambian para que el elemento más grande sea el elemento pivote.
- *Pivoteo completo*: Se busca tanto en las columnas como en las filas el elemento más grande y luego se intercambian. Este se usa en muy raras ocasiones.

Además de evitar la división entre cero, el pivoteo también minimiza el error de redondeo. Otra técnica que también suele usarse es el *escalamiento*, en aquellos casos en los que algunas de las ecuaciones de un sistema tienen coeficientes mucho más grandes que otros. Esta aplicación, también minimiza los errores de redondeo. La técnica de Escalamiento no la consideraremos en esta práctica.

5.1.3 Factorización *LU*

Supongamos ahora que la matriz A se puede escribir como el producto de LU, donde:

L : es una matriz triangular inferior de orden nxn
U : es una matriz triangular superior de orden nxn

Esto no siempre es posible. Cuando es posible factorizar $A = LU$ se dice que A tiene una descomposición LU, pero esto no ocurre de una única forma. Un criterio que nos permite asegurar cuando dicha descomposición es posible es el siguiente resultado:

Teorema. *Si la matriz A es regular, existe una matriz de permutación P, tal que PA = LU, siendo L una matriz triangular inferior, con todos los elementos de la diagonal principal iguales a 1, y U una matriz triangular superior, con* $u_{ii} \neq 0$ *, para todo valor de i.*

En otras palabras si *A* es no singular, existe una matriz de permutación *P* tal que *PA = LU*, donde *L* es una matriz triangular inferior con $l_{ii} = 1$, para todo *i* y *U* es una matriz triangular superior no singular. Recordemos que una matriz es regular, o no singular, si es invertible y que una matriz es invertible si y sólo si el determinante de *A* es distinto de cero.

Así, si se conocen *L*, *U* y *P*, el problema, se reduce a resolver dos sistemas triangulares.

Efectivamente, si en el sistema de partida *Ax = b*, multiplicamos por *P*, resulta:

$$PAx = P\,b \quad \rightarrow \quad LU\,x = b_1; \quad \text{con } b_1 = Pb$$

Y haciendo $U\,x = y$, el problema se reduce a resolver:

$$L\,y = b_1$$

que es un sistema triangular inferior el cual resolveremos por sustitución progresiva calculando el vector *y*. Sustituyendo éste en el segundo sistema:

$$U\,x = y$$

que es triangular superior, y utilizando sustitución regresiva podremos calcular el vector *x* que es la solución del sistema de partida.

De manera similar al caso de la eliminación de Gauss, la descomposición *LU* requiere de pivoteo para evitar la división entre cero. Sin embargo, para simplificar en esta práctica no abordaremos el tema del pivoteo.

Existe un conocido algoritmo que implementa la descomposición *LU* con eliminación de Gauss. En él la matriz *L* tiene números 1 en la diagonal. Formalmente, a esto se le denomina *descomposición* o factorización de *Doolittle*. Un método alternativo usa una matriz *U* con números 1 sobre la diagonal. Esto se conoce como *descomposición Crout*. El algoritmo de factorización de *Doolittle* viene dado por:

$$l_{kk} = 1 \; ; \; k = 1, 2, \ldots, n$$

$$u_{kj} = \left(a_{kj} - \sum_{s=1}^{k-1} l_{ks} u_{sj} \right) ; \quad j = k+1, k+2, \ldots, n; \; k = 1, 2, \ldots, n$$

$$l_{ik} = \left(a_{ik} - \sum_{s=1}^{k-1} l_{is} u_{sk} \right) / u_{kk} \; ; \quad i = k+1, k+2, \ldots, n; \; k = 1, 2, \ldots, n$$

Se puede demostrar que, si no hay que permutar filas, la matriz *U* es la que se obtiene por la triangulación de Gauss y la matriz *L* se obtiene poniendo 1 en la diagonal principal y con *L*[*i, j*] el coeficiente que se usó en el método de Gauss para eliminar ese elemento.

En la práctica nosotros obtendremos la factorización LU solamente mediante el uso de los comandos del Maxima, pues su programación excede los objetivos del curso.

5.1.4 Factorización de Cholesky

Si *A* es una matriz simétrica, definida positiva, es posible descomponer $A = LU$, con *L* triangular inferior y *U* triangular superior, de forma que *L* y *U* sean traspuestas una de la otra, con lo que la factorización será:

$$A = {}^tU\,U = L\,{}^tL$$

De las distintas formas de caracterizar las matrices definidas positivas, recordemos que:

(i) *A* definida positiva si y sólo si todos sus menores principales son positivos.
(ii) *A* es definida positiva si y sólo si todos los autovalores de *A* son positivos.

Se demuestra que el cálculo de los elementos de la matriz *L* viene dado por:

$$l_{11} = \sqrt{a_{11}}$$

$$l_{i1} = \frac{a_{i1}}{l_{11}}\ ,\ i = 2,3,....,n$$

$$l_{ii} = \left(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2 \right)^{1/2}\ ;\ \ i = 2,3,....,n$$

$$l_{ij} = \frac{1}{l_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right) \quad \begin{matrix} i = j+1,\ j+2,....,n \\ j = 2,3,.....,n \end{matrix}$$

$$l_{ij} = 0 \qquad i < j$$

Así, podemos factorizar $A = L\,{}^tL$, con *L* triangular inferior, con lo que $Ax = b$ pasa a ser:

$$L\,{}^tL\,x = b$$

y, como en el caso anterior, denotando ${}^tL\,x = y$, el problema se reduce a resolver el sistema triangular inferior:

$$L\,y = b$$

y, a continuación, el sistema triangular superior:

$${}^tL\,x = y$$

En la práctica nosotros obtendremos la factorización de *Cholesky* solamente mediante el uso de los comandos del Maxima, pues su programación excede los objetivos del curso.

5.2 Comandos Maxima para la resolución de sistemas lineales

Como ya vimos, la resolución de sistemas lineales, se puede hacer con **solve**, como puede verse en el ejemplo que sigue:

```
(%i1)    kill(all)$
(%i1)    solve([x-y=3,x+y=1],[x,y]);
(%o1)    [[x=2,y=-1]]
```

Pero se dispone de un comando más eficiente, que funciona como **solve**, se trata del comando:

linsolve([ecuaciones],[variables])

que resuelve el sistema lineal [ecuaciones] respecto a las variables [variables], como se ve en el ejemplo:

```
(%i2)    linsolve([x-y=3,x+y=1],[x,y]);
(%o2)    [x=2,y=-1]
```

También se puede definir el sistema de ecuaciones separadamente.

```
(%i3)    p:[3*x+5*y-4*z=1,x+2*y+3*z=-2,-4*x+y-3*z=4];
(p)      [-4 z+5 y+3 x=1,3 z+2 y+x=-2,-3 z+y-4 x=4]
(%i4)    linsolve(p,[x,y,z]);
(%o4)    [x=-61/108,y=11/108,z=-59/108]
```

No hay problema cuando hay infinitas soluciones.

```
(%i5)    p:[x+y+3*z=1, 3*x+5*y-z=2, -x-3*y+7*z=0]$
(%i6)    linsolve(p,[x,y,z]);
solve: dependent equations eliminated: (1)
(%o6)    [x=-(16 %r1-3)/2,y=(10 %r1-1)/2,z=%r1]
```

En este caso sólo 2 ecuaciones son linealmente independientes y habría infinitas soluciones. Maxima llama %*r*1 al parámetro arbitrario.

Cuando no hay soluciones, Maxima lo indica como sigue:

```
(%i7)    p:[x+y=3,2*x+2*y=5]$
(%i8)    linsolve(p,[x,y]);
(%o8)    []
```

En la Práctica 1 vimos como introducir matrices con las que después trabajar. También vimos una serie de comandos que ahora vamos a repasar, junto con otros nuevos que nos serán de utilidad para la resolución de los sistemas lineales.

```
(%i1)   kill(all)$
(%i1)   A:matrix([1,0,-2], [2,3,2], [-2,0,1]);
```

$$(A)\quad \begin{bmatrix} 1 & 0 & -2 \\ 2 & 3 & 2 \\ -2 & 0 & 1 \end{bmatrix}$$

Para definir una matriz, además del comando **matrix**, recordemos que wxMaxima nos ofrece una interfaz muy atractiva, a la que se puede acceder a través del menú **Algebra-Introducir matriz**.

Para acceder a los elementos de una matriz, podemos utilizar los corchetes.

```
(%i2)   A[1,3];
(%o2)   -2
```

Utilizando **submatrix** se obtienen submatrices, para lo cual introducimos las filas a eliminar (si existe alguna), a continuación el nombre de la matriz y por último las columnas a eliminar (en su caso).

```
(%i3)   submatrix(1,A,1);
```

$$(\%o3)\quad \begin{bmatrix} 3 & 2 \\ 0 & 1 \end{bmatrix}$$

Las funciones **row** y **col** nos permiten acceder, respectivamente, a filas y columnas que deseemos. Se pueden añadir filas y columnas nuevas con **addrow** y **addcol**, todo lo cual muestran los siguientes ejemplos:

```
(%i4)   row(A,1);
(%o4)   [1  0  -2]
(%i5)   col(A,3);
```

$$(\%o5)\quad \begin{bmatrix} -2 \\ 2 \\ 1 \end{bmatrix}$$

```
(%i6)   addcol(A,[10,11,12]);
```

$$(\%o6)\quad \begin{bmatrix} 1 & 0 & -2 & 10 \\ 2 & 3 & 2 & 11 \\ -2 & 0 & 1 & 12 \end{bmatrix}$$

Recordemos otras funciones aplicables sobre matrices:

- **transpose** = da la matriz traspuesta
- **invert** = da la matriz inversa
- **determinant** = calcula el determinante de una matriz cuadrada

- **rank** = calcula el rango
- **mat_trace** = devuelve la traza
- **triangularize** = devuelve una matriz triangular superior resultante de aplicar el método de Gauss a una matriz dada
- **eigenvalues** = devuelve dos listas, la primera formada por los autovalores de una matriz y la segunda por sus multiplicidades
- **matrix_size** = devuelve una lista con el número de filas y columnas, respectivamente
- **rowswap(M, i, j)** = si M es una matriz, intercambia las filas *i* y *j*
- **columnswap(M, i, j)** = si M es una matriz, intercambia las columnas *i* y *j*

Algunos ejemplos:

```
(%i1)   kill(all)$
(%i1)   A:matrix([0,-1,2],[2,-3,3],[1,1,1]);
```

$$(A)\quad \begin{bmatrix} 0 & -1 & 2 \\ 2 & -3 & 3 \\ 1 & 1 & 1 \end{bmatrix}$$

```
(%i2)   transpose(A);
```

$$(\%o2)\quad \begin{bmatrix} 0 & 2 & 1 \\ -1 & -3 & 1 \\ 2 & 3 & 1 \end{bmatrix}$$

```
(%i3)   invert(A);
```

$$(\%o3)\quad \begin{bmatrix} -\frac{2}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{9} & -\frac{2}{9} & \frac{4}{9} \\ \frac{5}{9} & -\frac{1}{9} & \frac{2}{9} \end{bmatrix}$$

```
(%i5)   matrix_size(A);
(%o5)   [3,3]

(%i6)   columnswap(A,1,2);
```

$$(\%o6)\quad \begin{bmatrix} -1 & 0 & 2 \\ -3 & 2 & 3 \\ 1 & 1 & 1 \end{bmatrix}$$

```
(%i7)   rowswap(A,1,2);
```

$$(\%o7)\quad \begin{bmatrix} 2 & -3 & 3 \\ 0 & -1 & 2 \\ 1 & 1 & 1 \end{bmatrix}$$

5.3 El Método de Eliminación de Gauss Simple

Si una matriz tiene todos los elementos situados por debajo de la diagonal nulos decimos que es triangular superior y si tiene todos los elementos situados por encima de la diagonal nulos decimos que es triangular inferior.

Resolver un sistema de *n* ecuaciones con *n* incógnitas cuya matriz sea triangular es muy sencillo porque basta (para el caso de triangular superior) empezar por la última ecuación para hallar la última incógnita, sustituir en la penúltima ecuación para hallar la penúltima incógnita, sustituir las incógnitas halladas en la antepenúltima ecuación para hallar la antepenúltima incógnita, etc. En definitiva resolver el sistema empezando por abajo (*sustitución regresiva*). Observa que, haciéndolo de esta forma, cada ecuación sólo tiene una incógnita al resolverla. Vamos a comenzar por programar el método de *sustitución regresiva*. Sin más que aplicar las fórmulas que vimos en la introducción tenemos el siguiente bloque.

```
(%i1)    kill(all)$
(%i1)    /* Función sustregre(A,b) */
         /* Resolución de Ax=b,
            cuando A es triangular superior,
            mediante sustitución regresiva */
         /* ARGUMENTOS DE ENTRADA: */
         /* A ... Matriz de coeficientes */
         /* b ... Matriz columna de términos independientes */
         /* ARGUMENTOS DE SALIDA: */
         /* sol ... Vector de incógnitas x[n] */
         sustregre(A,b):=block(
         [m,n]:matrix_size(A),
         if m#n then
             (disp("ERROR: Matriz NO Cuadrada"),return()),
         if m#length(b) then
             (disp("ERROR: Sistema NO Coherente"),return()),
         if determinant(A)=0 then
             (disp("ERROR: Matriz NO Regular"),return()),
         x:zeromatrix(n,1),
         x[n]:b[n]/A[n,n],
         for i:n-1 thru 1 step -1 do(
         x[i]:(b[i]-sum(A[i,j].x[j],j,i+1,n))/A[i,i]),
         return(print("La solución es:",x,""))
             )$;
```

Hemos incluido tres comprobaciones sobre la validez del método, que devuelven un mensaje error en caso de incumplirse. Veamos un ejemplo.

```
(%i2)    A:matrix([2,-3,3],[0,-2,4],[0,0,-9]);
```

$$(A)\quad \begin{bmatrix} 2 & -3 & 3 \\ 0 & -2 & 4 \\ 0 & 0 & -9 \end{bmatrix}$$

```
(%i3)    b:matrix([-8],[-20],[-2]);
```

$$(b)\quad \begin{bmatrix} -8 \\ -20 \\ -2 \end{bmatrix}$$

```
(%i4)    sustregre(A,b);
```

$$\textit{La solución es:}\quad \begin{bmatrix} \frac{34}{3} \\ \frac{94}{9} \\ \frac{2}{9} \end{bmatrix}$$

(%o4)

Comprobemos la solución con el comando **linsolve**.

```
(%i6)    sis:[2*x1-3*x2+3*x3=-8,-2*x2+4*x3=-20,-9*x3=-2]$
         linsolve(sis,[x1,x2,x3]);
```

$$(\%o6)\quad [x1=\frac{34}{3}, x2=\frac{94}{9}, x3=\frac{2}{9}]$$

Una vez implementado el método de sustitución regresiva, veamos cómo crear un bloque para el método de eliminación llamado el *método de triangulación de Gauss* que se basa en pasar el sistema a uno equivalente cuya matriz de coeficientes sea triangular superior. Vamos a comenzar por el caso más simple: *sin pivoteo*.

```
(%i7)    /* Función gausssimple(A,b) */
         /* Dado Ax=b,
            convierte el sistema en triangular superior,
            sin pivoteo */
         /* ARGUMENTOS DE ENTRADA: */
         /* A ... Matriz de coeficientes */
         /* b ... Matriz columna de términos independientes */
         /* ARGUMENTOS DE SALIDA: */
         /* MA ... Matriz ampliada triangular superior */
         gausssimple(A,b):=block(
         [m,n]:matrix_size(A),
         MA:addcol(A,b),
              for k:1 thru n-1 do (
                 for i:k+1 thru n do (
                  coe:MA[i,k]/MA[k,k],
                     for j:k thru n+1 do (
                  MA[i,j]:MA[i,j]-coe*MA[k,j]
                 )
                 )
                 ),MA
            )$;
```

Trabajamos con la matriz ampliada, que llamaremos *MA*, para de esa forma operar también con los términos independientes. Observemos que utilizamos tres bucles. Hemos denotando por:

k = incógnita que se está eliminado.
i = ecuación (fila) de la cual la estamos eliminando.
j = coeficiente (columna) de la ecuación (fila) *i* - ésima que estamos transformando.

En el más interno (con variable *j*) restamos de cada ecuación que está por debajo la ecuación donde está el pivote por el factor adecuado para que se haga cero el elemento que está por debajo del pivote. El segundo bucle (con variable *i*) recorre las ecuaciones que están por debajo del pivote y el primero (con variable *k*) recorre las columnas eligiendo los pivotes.

Veamos un ejemplo.

```
(%i9)  A:matrix([1,-1,2],[2,-3,3],[1,1,1]);
       b:matrix([-8],[-20],[-2]);
```

$$(A)\quad \begin{bmatrix} 1 & -1 & 2 \\ 2 & -3 & 3 \\ 1 & 1 & 1 \end{bmatrix}$$

$$(b)\quad \begin{bmatrix} -8 \\ -20 \\ -2 \end{bmatrix}$$

```
(%i10) gausssimple(A,b);
```

$$(\%o10)\quad \begin{bmatrix} 1 & -1 & 2 & -8 \\ 0 & -1 & -1 & -4 \\ 0 & 0 & -3 & -2 \end{bmatrix}$$

Extraemos ahora la nueva matriz triangular superior, el nuevo vector de términos independientes y les aplicamos el algoritmo de sustitución regresiva anterior.

```
(%i12) TA:submatrix(MA,4);
       Tb:col(MA,4);
```

$$(TA)\quad \begin{bmatrix} 1 & -1 & 2 \\ 0 & -1 & -1 \\ 0 & 0 & -3 \end{bmatrix}$$

$$(Tb)\quad \begin{bmatrix} -8 \\ -4 \\ -2 \end{bmatrix}$$

```
(%i13)  sustregre(TA,Tb);
```

La solución es:

$$\begin{bmatrix} -6 \\ \frac{10}{3} \\ \frac{2}{3} \end{bmatrix}$$

(%o13)

Comprobemos la solución con el comando **linsolve**.

```
(%i15)  sis:[x1-x2+2*x3=-8,2*x1-3*x2+3*x3=-20,x1+x2+x3=-2]$
        linsolve(sis,[x1,x2,x3]);
```

(%o15) $[x1=-6, x2=\frac{10}{3}, x3=\frac{2}{3}]$

Hemos preferido no juntar los dos bloques para apreciar mejor las dos fases del método de Gauss, pero recomendamos al lector que pruebe a hacerlo por su cuenta.

Recordemos también que Maxima devuelve la forma triangular superior de una matriz rectangular con la instrucción **triangularize**. Advertimos que el resultado devuelto por dicho comando no necesariamente va a coincidir con el obtenido por nosotros, si bien la solución del sistema va a ser obviamente la misma.

```
(%i16)  C:addcol(A,b);
```

(C)
$$\begin{bmatrix} 1 & -1 & 2 & -8 \\ 2 & -3 & 3 & -20 \\ 1 & 1 & 1 & -2 \end{bmatrix}$$

```
(%i17)  triangularize(C);
```

(%o17)
$$\begin{bmatrix} 1 & -1 & 2 & -8 \\ 0 & -1 & -1 & -4 \\ 0 & 0 & 3 & 2 \end{bmatrix}$$

```
(%i18)  MA;
```

(%o18)
$$\begin{bmatrix} 1 & -1 & 2 & -8 \\ 0 & -1 & -1 & -4 \\ 0 & 0 & -3 & -2 \end{bmatrix}$$

```
(%i19)  is (triangularize(C)=MA);
(%o19)  false
```

5.4 El Método de Eliminación de Gauss con Pivoteo

Como vimos, el algoritmo anterior falla si alguno de los pivotes es cero. Además es conveniente elegir como pivote el elemento de la columna de mayor valor absoluto, para hacer esto permutamos las filas con el comando **rowswap**. El algoritmo puede ser:

```
(%i20)   /* Función gausspivoteo(A,b) */
         /* Dado Ax=b,
            convierte el sistema en triangular superior,
            con pivoteo parcial */
         /* ARGUMENTOS DE ENTRADA: */
         /* A ... Matriz de coeficientes */
         /* b ... Matriz columna de términos independientes */
         /* ARGUMENTOS DE SALIDA: */
         /* MA ... Matriz ampliada triangular superior */
         gausspivoteo(A,b):=block(
         [m,n]:matrix_size(A),
         MA:addcol(A,b),
              for k:1 thru n-1 do (
                 /* elección de pivote*/
                 for i:k+1 thru n do (
                 if abs(MA[k,k])<abs(MA[i,k]) then
                 MA:rowswap(MA,k,i)),
                 /* fin eleccion de pivote */
                 for i:k+1 thru n do (
                  coe:MA[i,k]/MA[k,k],
                     for j:k thru n+1 do (
                   MA[i,j]:MA[i,j]-coe*MA[k,j]
                 )
                 )
                 ),MA
            )$;
```

Como vemos, sobre el algoritmo anterior, tan sólo hemos incluido un proceso de intercambio de filas previo, buscando la fila debajo con mayor valor absoluto en esa columna. Veamos un ejemplo.

```
(%i22)   A:matrix([1,-1,2],[2,-2,3],[1,1,1]);
         b:matrix([-8],[-20],[-2]);
```

$$(A)\quad \begin{bmatrix} 1 & -1 & 2 \\ 2 & -2 & 3 \\ 1 & 1 & 1 \end{bmatrix}$$

$$(b)\quad \begin{bmatrix} -8 \\ -20 \\ -2 \end{bmatrix}$$

```
(%i23)  gausspivoteo(A,b);
0 errors, 0 warnings
```

$$(\%o23)\quad \begin{bmatrix} 2 & -2 & 3 & -20 \\ 0 & 2 & -\frac{1}{2} & 8 \\ 0 & 0 & \frac{1}{2} & 2 \end{bmatrix}$$

Extrayendo sobre la matriz ampliada *MA* los valores que necesitamos, aplicamos de nuevo sustitución regresiva.

```
(%i25)  TA:submatrix(MA,4);
        Tb:col(MA,4);
```

$$(TA)\quad \begin{bmatrix} 2 & -2 & 3 \\ 0 & 2 & -\frac{1}{2} \\ 0 & 0 & \frac{1}{2} \end{bmatrix}$$

$$(Tb)\quad \begin{bmatrix} -20 \\ 8 \\ 2 \end{bmatrix}$$

```
(%i26)  sustregre(TA,Tb)$
```

La solución es: $\begin{bmatrix} -11 \\ 5 \\ 4 \end{bmatrix}$

Y comprobamos la solución.

```
(%i28)  sis:[x1-x2+2*x3=-8,2*x1-2*x2+3*x3=-20,x1+x2+x3=-2]$
        linsolve(sis,[x1,x2,x3]);
(%o28)  [x1=-11,x2=5,x3=4]
```

Es interesante observar que, si no efectuamos el pivoteo, obtenemos un error en el algoritmo simple.

```
(%i29)  gausssimple(A,b);
expt: undefined: 0 to a negative exponent.
 #0: gausssimple(a=matrix([1,-1,2],[2,-2,3],[1,1,1]),b=matrix([-8],[-20],[-2])
  -- an error. To debug this try: debugmode(true);
```

5.5 Métodos de factorización

Habida cuenta de la sencillez y eficiencia computacional de la resolución de sistemas lineales cuya matriz de coeficientes sea triangular, tanto con matriz triangular inferior como superior (en cualquier caso conlleva un total de n^2 operaciones la resolución de uno de tales sistemas de n ecuaciones lineales), estamos interesados en los métodos de factorización de la matriz de coeficientes como producto de matrices triangulares, que conduciría a la resolución sencilla de dos sistemas lineales con matriz triangular. Veamos algunos comandos de Maxima para realizar factorizaciones de matrices.

5.5.1 Factorización *LU*

Dada una matriz cuadrada *A* la factorización *LU* consiste en calcular tres matrices, una matriz permutación *P*, y otras dos matrices *L* y *U*, con *L* triangular inferior con unos en la diagonal, *U* triangular superior y tales que *PA* = *LU*. Maxima calcula dicha factorización directamente con el uso de comandos de la siguiente forma:

- **get_lu_factors (lu_factor (A))** = da una lista [P, L, U], donde P es la matriz de permutación, L es la triangular inferior y U es la triangular superior, con P A = L U
- **lu_backsub (lu_factor (A), b)** = resuelve el sistema lineal A x = b.

Veamos un ejemplo.

```
(%i1)  kill(all)$
(%i1)  A:matrix([1,0,2],[2,7,3],[2,1,0])$
(%i2)  get_lu_factors(lu_factor(A));
```

$$(\%o2)\quad \left[\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix},\begin{bmatrix}1&0&0\\2&1&0\\2&\frac{1}{7}&1\end{bmatrix},\begin{bmatrix}1&0&2\\0&7&-1\\0&0&-\frac{27}{7}\end{bmatrix}\right]$$

Las tres matrices que aparecen son respectivamente *P*, *L* y *U*. Comprobamos que la factorización es correcta:

```
(%i5)  P:%o2[1]$
       L:%o2[2]$
       U:%o2[3]$
(%i7)  P.A;
       L.U;
```

$$(\%o6)\quad \begin{bmatrix}1&0&2\\2&7&3\\2&1&0\end{bmatrix}$$

$$(\%o7)\quad \begin{bmatrix}1&0&2\\2&7&3\\2&1&0\end{bmatrix}$$

Ha hecho la factorización *PA=LU*, pero sólo cambiando filas en caso de división por cero. Si queremos que haga pivoteo parcial tendremos que pasarle el argumento opcional **floatfield**. Notar que en este caso calcula la factorización de manera numérica.

```
(%i8)   get_lu_factors(lu_factor(A,floatfield));
(%o8)   [matrix([0,0,1],[1,0,0],[0,1,0]),matrix([1,0,0],[1.0,1,0],[0.5,0.5833333333333334,1]),matrix([2.0,7.0,3.0],[0,-6.0,-3.0],[0,0,2.25])]
```

Para resolver un sistema *Ax=b* utilizando la descomposición *LU*, hacemos:

```
(%i9)   b:matrix([-1],[-1],[2])$
(%i10)  lu_backsub(lu_factor(A),b);
(%o10)  matrix([1],[0],[-1])
(%i12)  sis:[x1+2*x3=-1,2*x1+7*x2+3*x3=-1,2*x1+x2=2]$
        linsolve(sis,[x1,x2,x3]);
(%o12)  [x1=1,x2=0,x3=-1]
```

5.5.2 Factorización de Cholesky

La factorización de Cholesky es una factorización parecida a la anterior, Maxima la realiza para matrices reales simétricas y en este caso la matriz *A* se descompone en la forma $A = LL^t$ con *L* triangular inferior. Dicha factorización está asegurada para matrices reales simétricas y definidas positivas, siendo en este caso los elementos diagonales de *L* positivos. Maxima calcula dicha factorización directamente con el uso del comando:

- **cholesky** = da la factorización Cholesky

```
(%i13)  A:matrix([2,-1,-2],[-1,14,7],[-2,7,5]);
(A)     matrix([2,-1,-2],[-1,14,7],[-2,7,5])
(%i14)  B:cholesky(A);
(B)     matrix([sqrt(2),0,0],[-1/sqrt(2),3^(3/2)/sqrt(2),0],[-sqrt(2),2^(3/2)/sqrt(3),1/sqrt(3)])
(%i15)  is (B.transpose(B)= A);
(%o15)  true
```

5.6 Ejercicios Propuestos

Ejercicio 1. Adaptar el bloque del algoritmo de sustitución regresiva para crear un bloque que resuelva el algoritmo de sustitución progresiva, para matrices triangulares inferiores. Aplicarlo a:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 3 & 0 \\ 3 & 4 & 5 \end{pmatrix}; b = \begin{pmatrix} -1 \\ -1 \\ 2 \end{pmatrix}$$

Comprobar la solución.

Ejercicio 2. Convierte a forma triangular por el método de Gauss y resuelve después los sistemas siguientes:

$$(a)\ \left.\begin{array}{rl} 2x - y & = 1 \\ -x + 2y - z & = 0 \\ -y + 2z & = 2 \end{array}\right\};\quad (b)\ \left.\begin{array}{rl} 4x + y + z + t & = 1 \\ x + 3y - z & = 2 \\ x - y + 2z & = 3 \\ x + y + 2t & = 5 \end{array}\right\};\quad (c)\ \left.\begin{array}{rl} 6x + 2y + z - t & = 0 \\ x + 5y + z - t & = 2 \\ x + y + 4z - t & = 0 \\ -x - z + 3t & = 10 \end{array}\right\}$$

Comprueba la solución.

Ejercicio 3. Convierte a forma triangular por el método de Gauss usando pivotes y resuelve después los sistemas siguientes:

$$(a)\ \left.\begin{array}{rl} -y + 2z & = 2 \\ -x + 2y - z & = 0 \\ 2x - y & = 1 \end{array}\right\};\quad (b)\ \left.\begin{array}{rl} x + y + 2t & = 5 \\ x - y + 2z & = 3 \\ x + 3y - z & = 2 \\ 4x + y + z + t & = 1 \end{array}\right\};\quad (c)\ \left.\begin{array}{rl} -x - z + 3t & = 10 \\ x + y + 4z - t & = 0 \\ x + 5y + z - t & = 2 \\ 6x + 2y + z - t & = 0 \end{array}\right\}$$

Comprueba la solución.

Ejercicio 4. Realizar una factorización LU sin pivoteo de la matriz de coeficientes del sistema:

$$\left.\begin{array}{c} x + 2y - z = -3 \\ -x - 4y + z = 5 \\ 2x + 10y + 2z = -4 \end{array}\right\}$$

Comprobar que dicha factorización está bien hecha. Además resolver usando la descomposición LU el sistema de ecuaciones lineales. Tendréis que resolver dos sistemas triangulares por sustitución progresiva y regresiva respectivamente.

Ejercicio 5. Realizar una factorización de Cholesky de la matriz:

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 2 & 8 & -2 \\ -1 & -2 & 17 \end{pmatrix}$$

Comprobar que dicha factorización está bien hecha. Utilizar la factorización para resolver el sistema de ecuaciones *Ax = b*, siendo *b* el vector columna (0, 0, 16). Tendréis que resolver dos sistemas triangulares por sustitución progresiva y regresiva respectivamente.