Descomposición L-U

Marco teórico:

Existen distintas maneras de resolver un sistema de ecuaciones lineales de la forma

$$A.x = b \quad (1)$$

donde $A \in \mathbb{R}^{nxn}$, x $\in \mathbb{R}^n$ es el vector de incognitas y b $\in \mathbb{R}^n$ el producto de A.x. Una de las maneras de resolver éste problema es utilizando el método de la inversa, donde se halla la matriz inversa de A, A^{-1}, y se la multiplica por el vector b. De esta manera se halla la solución como:

$$A^{-1}.b = x$$

Uno de los mayores problemas de éste método es que el costo computacional del hallazgo de la inversa de A es alto, lo cual aumenta el tiempo de resolución del problema.
Existen otros métodos de resolución, entre ellos está el método de descomposición L-U que se basa en realizar operaciones elementales en forma de matrices sobre A de tal manera que el resultado sea una matriz triangular superior. Las matrices elementales utilizadas son de matrices triangular inferior unitaria donde en la parte de abajo solo tiene una columna no nula. Cada una de éstas matrices elementales recibiran el nombre L_i donde i indica el orden en el cual se le aplica esa operacion a la matriz A. Para facilitar la explicación tomemos el caso de una matriz de n=4. la cantidad de matrices L_i es siempre n-1, en este caso será 3. Y el proceso de obtención de la matriz U sería:

$$L_3.L_2.L_1.A = U \quad (2)$$

Una propiedad muy interesante de las matrices con la forma de L_i es que el proceso de obtención de inversa consta en cambiar el signo de los elementos de debajo de la diagonal. Y por otro lado tambien es cierto que el producto de matrices L_i donde las columnas llenas son distintases una matriz que posee las 2 columnas llenas con los coeficientes correspondiente a cada una de ellas. por ejemplo:

$$L_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ x & 1 & 0 & 0 \\ y & 0 & 1 & 0 \\ z & 0 & 0 & 1 \end{pmatrix};\ L_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & w & 1 & 0 \\ 0 & r & 0 & 1 \end{pmatrix};\ L_1 * L_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ x & 1 & 0 & 0 \\ y & w & 1 & 0 \\ z & r & 0 & 1 \end{pmatrix};$$

Aplicando estas dos propiedades en (2) llegamos a lo siguiente:

$$\left[L_3^{-1}.L_2^{-1}.L_3^{-1}\right].L_3.L_2.L_1.A = \underbrace{\left[L_3^{-1}.L_2^{-1}.L_3^{-1}.\right]}_{L} U$$

$$A = L.U \quad (3)$$

De esta manera se obtiene una matriz L triangular inferior unitaria y una matriz U triangular superior tal que el producto L.U de como resultado la matriz A. Reemplazando A por su equivalete en (1):

$$L.U.x = b$$

Si tomamos el producto U.x como un nuevo vector y obtendremos los siguientes sistema de ecuaciones:

$$L.y = b$$
$$U.x = y$$

Estos sistemas de ecuaciones son de simple resolucion, el segundo, es de solución casi trivial con un algoritmo de backward substitution. una vez hallados los valores del valor y la primer ecuación se puede resolver con un algoritmo de forward substitution.

Descomposición L-U con pivoteo

Si bien este método es más económico en cuanto a tiempo de proceso, pero presenta otro problema: es imposible realizar la descomposición si aparece un 0 en el elemto con el cual se debe reducir.
Para solucionar este problema se realiza un intercambio de filas con las filas siguientes. Para la elección de la fila se utiliza la que tenga el elemento con mayor valor absoluto en la columna en la cual se esté realizando la reducción. Para ello basta con agregar en el código original una función que realize este pivoteo.

Tiempo de resolución y órden.

Se considera que el algorítmo de resolución L-U es de órden n^3, es decir, el tiempo de resolución aumenta con el cubo de el órden de la matriz, esto indica que mientras mas grande es el orden de la matriz mucho mayor es el aumento de tiempo de resolucion.
Consideremos ahora los tiempos medidos para el algoritmo de descomposición L-U con pivoteo:

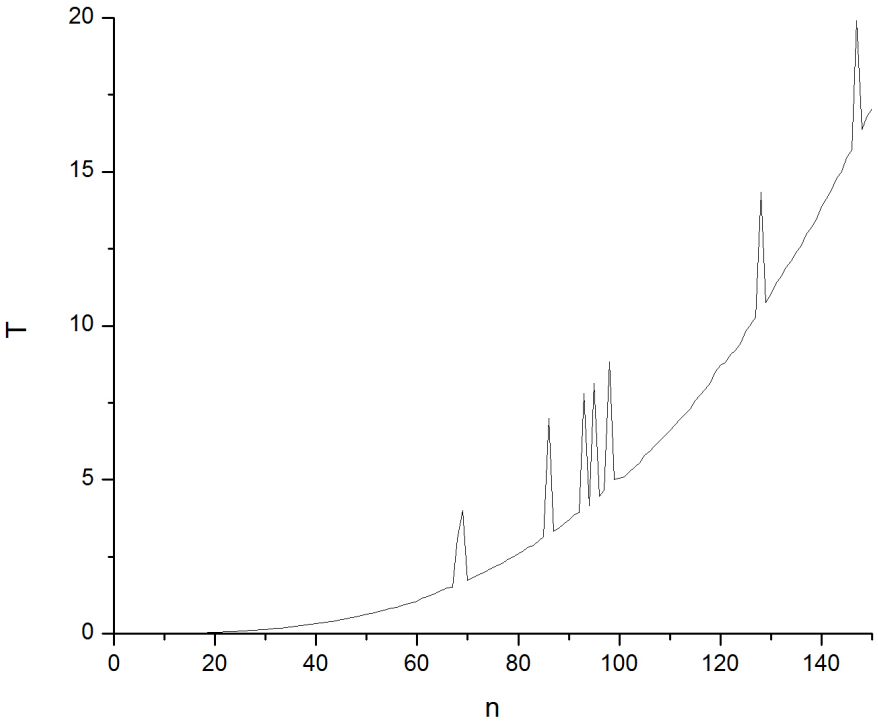

Figura 1: tiempo de resolucion en función de el órden n de la matriz.

Puede verse en la **Figura 1** que el tiempo de resolucion no aumenta linealmente con el orden, sino que lo hace aún mas rápido como n^3 una de las mejores formas de demostrar que el tiempo crece como el cubo de el orden de la matriz es aplicar logaritmo natural a n y a T. Con esto deberiamos obtener una sucesión de puntos a la cual podriamos ajustarle una recta, cuya pendiente debe ser el órden del algoritmo. Luego de la aplicación y el ajuste se obtiene:

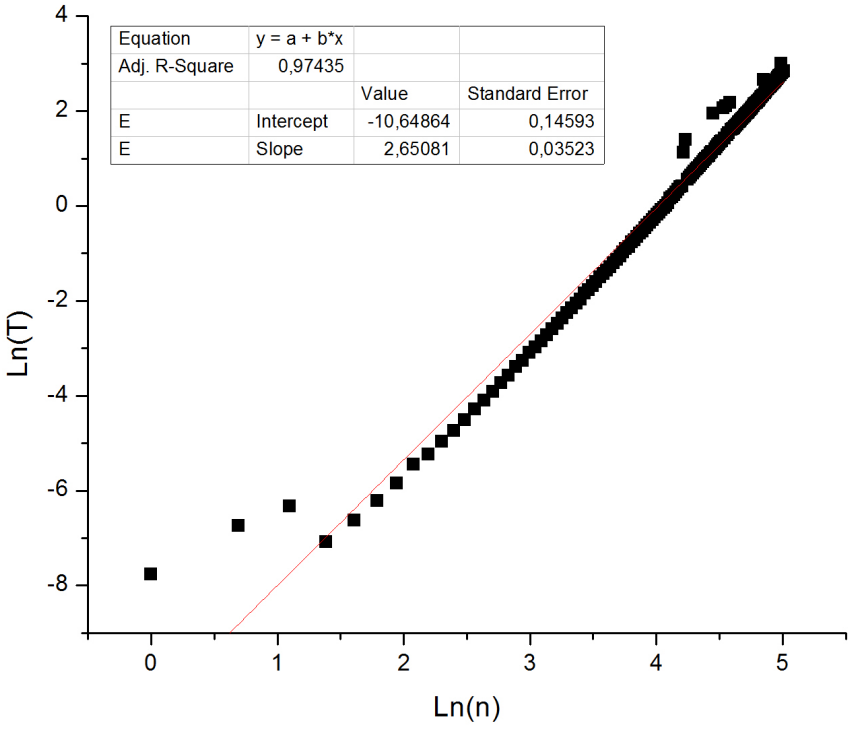

figura 2: logaritmo natural del tiempo de resolucion en funcion del logaritmo natural de n y la aplicación de un ajuste lineal con pendiente 2.65.

Puede verse, tanto en la **Figura 1** como en la **Figura 2** que existen puntos que parecieran no pertener a las curvas, estos pueden deberse a las inestabilidades del sistema mientras se corre el programa. podemos ver que la pendiente de la regresión es menor a 3 lo cual se debe a que la función se comporta mas parecida a n^3 para n grandes. Veamos una regresión aplicada a valores de n entre 100 y 150:

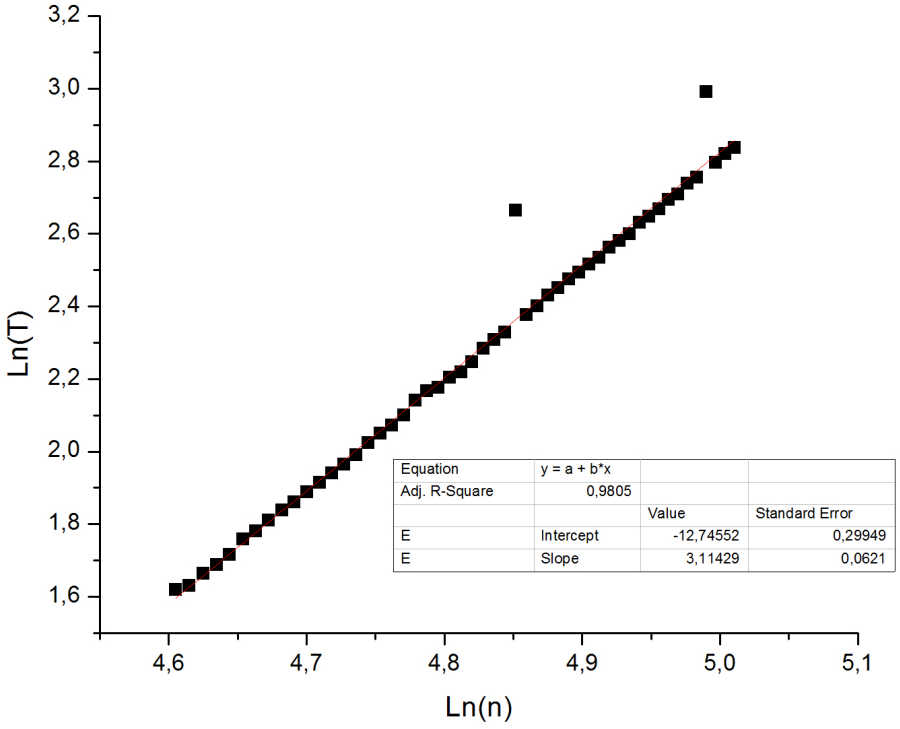

Figura 3: Grafico de Ln(T) en función del Ln(n) para valores de n entre 100 y 150. Con un ajuste lineal aplicado con pendiente 3.11.

Puede verse como en este caso la pendiente del ajuste es mas próxima a 3, aunque por arriba de este numero, lo cual es entendible considerando que existen puntos alejados de los esperados.

Codigo del programa:

En la siguiente se adjunta el código del programa de resolución de sistemas de ecuaciones de orden n por el método de descomposición L-U con pivoteo parcial.

```
# Descomposicion L U
for n=1:150
a= rand(n,n);
x1= rand(n,1);                          #Solucion real, para calcular el error
b=a*x1;
function [a,b]= pivoteo(a,b, k,n)       # funcion de pivoteo
        [x,ix]=max(abs(a(k:n,k)));      # k+1:n va desde k+1 hasta n
                piv1=eye(n);
                ix=ix+k-1;              # correccion de la posicion del max
                piv1(k,k)=piv1(ix,ix)=0;
                piv1(ix,k)=1;
                piv1(k,ix)=1;
                a=piv1*a;               # pivoteo de a
                b=piv1*b;               # pivoteo de b
endfunction
```

```
t1=time;
for k=1:(n-1)
        [a,b]=pivoteo(a,b,k,n);            # fin pivoteo
        for l=(k+1):n
                a(l,k)=(a(l,k))/(a(k,k));          # llenado de la parte inferior de L en a
                for m=(k+1):n
                        a(l,m)=a(l,m) -a(l,k)*a(k,m);   # llenado de la parte de U en a
                endfor
        endfor
endfor
U=zeros(n);                                        # armado de las matrices L y U
L=eye(n);
for k=1:n
        for l=1:n
                if(l<=k)
                        U(l,k)=a(l,k);
                elseif(k<l)
                        L(l,k)=a(l,k);
                end
        endfor
endfor                                              # Fin del armado
y=zeros(n,1);
x=zeros(n,1);
for k=1:n                                          # solucion de L*y=b
        suma=0;
        for g=1:k-1
                suma+=(a(k,g)*y(g));
        endfor
        y(k)=(b(k)-suma)/(L(k,k));
endfor
for k=n:-1:1                                       # solucion de U*x=y
        suma=0;
        for g=k+1:n
                suma+=(a(k,g)*x(g));
        endfor
        x(k)=(y(k)-suma)/(U(k,k));
endfor
T(n)=time-t1;
error(n)=norm(x-x1);
endfor
```