

Trabajo Práctico N°2 – Cuatrimestre 1 de 2020

Análisis estático de una estructura

Grupo N° 23	Mauricio Fernandez	94363
	Ricardo Brandan	96970
	Iñaki Ustariz	102252

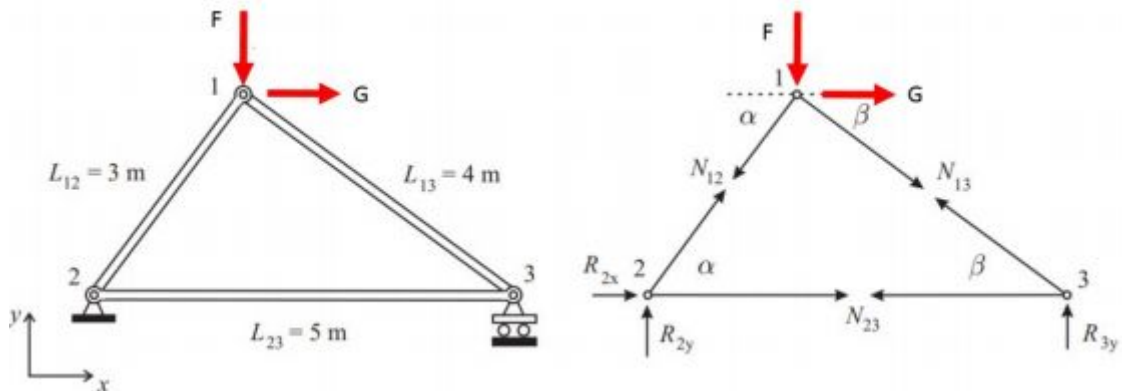
Fecha	Correcciones / Observaciones	Docente
11/06	1) Buen TP 2) Incluir resultados del punto c) 3) Ajustar detalles de los gráficos 4) Expresar los resultados correctamente redondeados del punto d)	Pablo García

Calificación Final	Docente	Fecha

Introducción	3
Desarrollo	5
Algoritmo de Eliminación Gaussiana	5
Función Diagonal Dominante	5
Función Pivoteo Parcial	5
Función Triangular	7
Función Sustitución Inversa	7
Función Sustitución Directa	9
Descomposición LU	9
Diferencias entre Crout, Doolittle y Cholesky	10
Costo computacional de cada algoritmo	10
Múltiples combinaciones de carga	11
Perturbaciones experimentales	15
Conclusión	18
Referencias	19
Anexo: Código de programación	20
Matriz es diagonal dominante	20
Pivoteo parcial en una matriz	21
Triangulador de matrices	22
Sustitución inversa de matriz triangular superior	22
Realiza una sustitución directa	23
Creador matriz de permutación	23
Descomposición LU	24
Graficador de fuerzas N12, N23, N13	25
Código de resolución de punto C	25
Código de resolución de punto D	26
Código de resolución de punto E	27

1. Introducción

En el siguiente informe, se analizará la estructura de la cubierta del Aeropuerto de Hamburgo en Alemania. Este sistema estructural puede ser modelizado mediante una serie de enlaces inextensibles unidos en nodos, con un extremo fijo y otro libre para realizar desplazamientos horizontales. A los efectos del problema que se intenta resolver, se supone que esta estructura está sometida a dos fuerzas aplicadas en uno de sus nodos.



Aplicando la Ley de Newton de acción y reacción en los nodos, se pueden calcular las tensiones a las que están sometidos los enlaces de la estructura cuando la misma está en equilibrio, sabiendo que al estar en equilibrio la sumatoria de las fuerzas tanto verticales como horizontales resultan ser igual a cero. La estructura cuenta con 3 nodos. Se arma un sistema de ecuaciones con N_{12} , N_{13} y N_{23} (que son los esfuerzos internos en cada una de las barras) y R_{2x} , R_{2y} , R_{3y} (las fuerzas de reacción), obteniéndose el siguiente sistema de ecuaciones lineales ($\alpha=53^\circ$ y $\beta=37^\circ$).

$$\begin{bmatrix} -\cos \alpha & 0 & \cos \beta & 0 & 0 & 0 \\ -\sin \alpha & 0 & -\sin \beta & 0 & 0 & 0 \\ \cos \alpha & 1 & 0 & 1 & 0 & 0 \\ \sin \alpha & 0 & 0 & 0 & 1 & 0 \\ 0 & -1 & -\cos \beta & 0 & 0 & 0 \\ 0 & 0 & \sin \beta & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} N_{12} \\ N_{23} \\ N_{13} \\ R_{2x} \\ R_{2y} \\ R_{3y} \end{bmatrix} = \begin{bmatrix} -G \\ F \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

El programa utilizado, al igual que en el trabajo anterior, fue Octave. Se generó un código con distintas operaciones básicas para ir sacando conclusiones de los problemas que se tenían. La mecánica del trabajo fue la siguiente:

- En primer lugar, se busca codificar un algoritmo que resuelva un Sistema de Ecuaciones Lineales mediante el método de Eliminación Gaussiana.

- Luego, se trató de codificar un algoritmo que resuelva un Sistema de Ecuaciones Lineales mediante el método de Factorización LU.
- Se procedió a resolviendo el problema con ambos algoritmos utilizados para valores de $F = 1000$ y $G = 100$ evaluando el costo computacional de cada algoritmo.
- Se continuó utilizando el algoritmo del método de Factorización LU, para resolver el problema para múltiples combinaciones de carga adoptando $F = 1000 * (1 + i/10)$ y $G = 100 * (1 + i/10)$, con $i=0,1,...,9$. Luego se graficó los resultados obtenidos en todas las barras y se sacó conclusiones.
- Por último, con $F = 1000 \pm 200$, y $G = 100 \pm 10$, se utilizó el algoritmo de Eliminación Gaussiana para calcular el esfuerzo en las tres barras con su respectiva cota de error. Luego se resolvió el problema aplicando perturbaciones experimentales. Y finalmente se volcaron los resultados en una tabla.

A partir de esta serie metódica de pasos, se fue estudiando el comportamiento de la estructura y su análisis.

2. Desarrollo

Algoritmo de Eliminación Gaussiana

Función Diagonal Dominante

Para implementar el método de eliminación de gauss utilizando pivoteo parcial, se implementó antes una función llamada *es_diagonal_dominante* para determinar si la matriz es diagonal dominante o no, ya que si lo fuera no sería necesario aplicar el pivoteo. Se sabe que una matriz es diagonal dominante si se cumple que, para cada fila, la suma de los elementos de la fila en valor absoluto, sin incluir el de la diagonal, es menor o igual al valor de la diagonal en valor absoluto. Esto se puede expresar como se ve en la Figura 1.

$$|a_{ii}| \geq \sum_{j=1, j \neq i}^n |a_{ij}|, 1 \leq i \leq n$$

Figura 1

Detalle. Las ecuaciones no se presentan como figuras

Para implementar el algoritmo que verifique esto se utilizaron dos(2) ciclos FOR. El índice del primer ciclo FOR representa cada fila de la matriz, el índice del segundo representa la columna. Lo que se hace al ingresar al primer FOR es guardar el valor de la diagonal en valor absoluto y se inicializa la variable *suma*, en la cual se almacenará la suma de todos los valores de la fila, a cero. Luego, se ingresa al segundo ciclo y se recorren las columnas de la fila sumando todos los valores de cada posición en valor absoluto excluyendo el de la diagonal (lo que se hace con un condicional IF), esta suma se guarda en la variable *suma*. Después de sumar todos los valores de una fila se verifica si la *suma* es mayor al valor de la diagonal. Si esta suma es mayor al valor de la diagonal, se deja de recorrer la matriz y se devuelve *false*, con lo cual se considera que la matriz no es diagonal dominante. Si la suma no fuera mayor al valor de la diagonal, entonces se sigue recorriendo la matriz. Si se llega a la última fila y la suma sigue sin ser mayor entonces se termina el recorrido y se devuelve *true*, esto significa que la matriz es diagonal dominante.

Función Pivoteo Parcial

La próxima función que se implementó para programar el método de eliminación de Gauss fue la del **pivoteo parcial**. La idea del pivoteo parcial es la siguiente: en el paso K, se busca en la columna la fila *r* tal que se cumple lo que se ilustra en la figura 2.

$$|a_{rk}^{(k)}| = \max |a_{ik}^{(k)}|, k \leq i \leq n \quad \text{ídem}$$

Figura 2

Es decir, se barren las filas desde k hasta n y se ve en cuál de ellas está el máximo en módulo. Una vez que se identifica se intercambian las filas r y k . Esto es necesario ya que es muy posible que se amplifiquen errores de no hacerlo. Lo que garantiza el pivoteo parcial es que va a producir un mejor condicionamiento, va a reducir errores.

Para implementar este algoritmo lo primero que se hizo es guardar el valor del pivote actual que tiene la matriz en la posición a_{ii} en la variable *pivote* y se inicializa una variable llamada *fila_pos_max* a cero en la cual se guarda el índice de la fila en la que se encuentre un valor en módulo mayor al *pivote*. La función *pivoteo_parcial* recibe por parámetro una variable *fila* y una variable *columna* que es la posición donde está el pivote actual, entonces a continuación lo que se hace es recorrer las fila de dicha columna con un ciclo FOR, comprobando con un condicional IF si el valor en módulo de alguna fila es mayor al *pivote*. En el caso en el que se halle un valor mayor al pivote, se guarda el índice de esa fila y ese valor mayor ahora pasa a ser el nuevo *pivote*. Se sigue recorriendo la fila y si se encuentra otro valor mayor al pivote guardado, se actualiza la variable *fila_pos_max* y *pivote* nuevamente. Se prosigue hasta llegar a la última fila.

Luego de haber recorrido la matriz se procede a hacer los intercambios de filas de ser necesario. Con un condicional IF se verifica si la variable *fila_pos_max* tiene el índice del pivote original, es decir el de la posición a_{ii} , y de ser así no se hace el cambio de filas [ver Figura 3].

Método de Gauss (Pivote Parcial)

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^1 & a_{23}^1 & a_{24}^1 \\ 0 & a_{32}^1 & a_{33}^1 & a_{34}^1 \\ 0 & a_{42}^1 & a_{43}^1 & a_{44}^1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}$$

Pivote Máximo

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{42}^1 & a_{43}^1 & a_{44}^1 \\ 0 & a_{32}^1 & a_{33}^1 & a_{34}^1 \\ 0 & a_{22}^1 & a_{23}^1 & a_{24}^1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_4 \\ b_3 \\ b_2 \end{pmatrix}$$

Figura 3: Al encontrarse un valor (pivote máximo) mayor al pivote de la posición a_{ii} se intercambia la fila que contiene al pivote máximo con la fila de a_{ii} .

Se puede observar que esta función que realiza el pivoteo parcial también recibe un vector llamado *vector_permutacion* que sirve para guardar los intercambios de filas y luego poder crear la matriz de permutación. También se recibe una matriz L que de la descomposición LU ya que al hacerse el pivoteo también se debe hacer cambios en esta matriz. Los cambios de filas de este *vector_permutacion* y matriz U se realizan solamente si se llama a la función *pivoteo_parcial* desde el método que realiza la descomposición LU. La

tercera variable booleana llamada *lu* que recibe la función sirve para esto, para saber si la función fue llamada desde la función que realiza la descomposición LU.

Una vez realizado el pivoteo parcial, el cambio de filas, entonces se devuelve la matriz A con los intercambios de filas hechos y además el *vector_permutacion* y la matriz U que para el caso del método de eliminación de Gauss no sirven. Se podría haber creado otra función para el método de descomposición LU pero se decidió utilizar la misma función agregando algunos cambios.

Función Triangular

Finalmente, en la implementación del algoritmo de eliminación de Gauss, que se encuentra en la función llamada *triangular*, lo que se hace es utilizar dos(2) ciclos FOR. El primer ciclo FOR recorre las columnas de la matriz (al índice se le restó 1 porque el pivoteo parcial no se le realiza a la última columna, ya que no se busca tener un cero en la última fila de esa columna).

Se puede notar también que se utiliza la función *rows()* de octave para obtener el número de columnas, y no *columns()*. Como la matriz A es cuadrada sería equivalente en este caso. El problema con usar *columns()* es que si la función recibe una matriz ampliada entonces el algoritmo va a aplicarle el pivoteo a toda la matriz ampliada, incluso a los vectores independientes si se le pasa más de uno, es decir, algo como $A|b_1 b_2 \dots b_n$. Utilizando *rows()* se está aplicando el pivoteo solamente a la matriz A, y la ventaja es que de esta manera sí se puede recibir una matriz ampliada con varios vectores independientes de la forma $A|b_1 b_2 \dots b_n$ donde solo se triangula A y la resta de filas se aplica a la matriz ampliada también.

Con el segundo FOR lo que se hace es recorrer cada fila de la columna, se calcula el multiplicador de la fila de la forma representada en la figura 4.

$$m_{ik} = \frac{a_{ij}^{(k)}}{a_{kk}^{(k)}}, \begin{cases} 2 \leq k \leq n \\ k+1 \leq i \leq n \end{cases} \quad \text{Ídem}$$

Figura 4

Para luego hacer la resta de la fila i, para la cual se calculó el multiplicador, menos la fila anterior i-1 multiplicada por el multiplicador calculado m_{ik} . De esta manera se sigue recorriendo cada fila de la columna colocando ceros debajo del elemento de la diagonal a_{ii} hasta llegar a la última fila, donde se termina el ciclo. Después de esto se pasa a la siguiente columna, se realiza el pivoteo si fuera necesario y se vuelve a poner ceros debajo del elemento $a_{i+1,i+1}$ de la diagonal. Se continúa hasta llegar a la columna j-1.

Función Sustitución Inversa

Una vez obtenida la matriz triangulada superiormente se aplica una **sustitución inversa** para poder obtener la solución x del sistema $Ux=b$.

$$U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdot & \cdot & \cdot & u_{1n} \\ 0 & u_{22} & u_{23} & \cdot & \cdot & \cdot & u_{2n} \\ 0 & 0 & u_{33} & \cdot & \cdot & \cdot & u_{3n} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & \cdot & \cdot & \cdot & u_{nn} \end{bmatrix} \quad b = \begin{bmatrix} b_1 \\ b_1 \\ \dots \\ b_n \end{bmatrix}$$

Para resolverlo se halla primero el valor de la última incógnita x_n utilizando la última fila, donde quedaría que el valor de la última incógnita es

$$x_n = \frac{b_n}{u_{nn}}$$

Y para hallar el valor de las demás incógnitas se va reemplazando en la fila anterior el valor de las incógnitas halladas y despejando. Esto se puede generalizar como se ilustra en la figura 5.

$$x_i = \frac{b_i - \sum_{k=i+1}^n u_{ik}x_k}{u_{ii}}; i = n, n-1, \dots, 1$$

Figura 5

Ídem

Para la implementación de este algoritmo primero se creó el vector *soluciones* en el cual se guardan las soluciones de cada incógnita hallada, esta matriz está inicializada con todos sus elementos iguales a cero.

El recorrido de la matriz se hace con dos(2) ciclos FOR en forma inversa. Es decir, se empieza desde la última fila de la matriz y “se va subiendo” hasta la primera fila. El segundo ciclo FOR lo que hace es recorrer las columnas de la fila y hacer la suma de la multiplicación de los coeficientes de cada incógnita con el valor hallado de esa incógnita, esto es lo mismo que reemplazar los valores hallados de las incógnitas y hacer la suma en el primer miembro. Esta suma excluye al elemento de la diagonal.

Una vez que se tiene esa suma, se guarda el valor del término independiente de esa fila y finalmente se hace la resta del término independiente b menos la suma que se obtuvo del primer miembro, dividiendo todo esto por el coeficiente de la incógnita que se quiere hallar y se lo guarda en el vector *soluciones*. Una vez que se termina de recorrer las filas de la matriz se devuelve este vector que contiene las soluciones.

Función Sustitución Directa

Análogamente, se implementó otra función que realiza una **sustitución directa** cuyo algoritmo es similar al de sustitución inversa con la diferencia que las filas de la matriz no se recorre de abajo hacia arriba sino de arriba hacia abajo. Este tipo de sustitución será útil para resolver un sistema con Descomposición LU.

Descomposición LU

El algoritmo utilizado para realizar la Descomposición LU es muy similar al del Método de Eliminación de Gauss. Una diferencia entre los métodos que se implementaron para estos dos es que en el caso del método de Gauss se devuelve solo la matriz A triangulada, en el de la descomposición se devuelven tres(3) matrices: L, U y P.

En este algoritmo se comienza creando una matriz L que tiene inicialmente ceros en todas sus posiciones, también se define una matriz U que contendrá los multiplicadores hallados para realizar la triangulación de la matriz y 1's en su diagonal. Otro vector que se define es el *vector_permutacion*, éste es un vector apuntador que sirve para ir guardando los cambios de filas que se van haciendo al triangular la matriz. El *vector_permutacion* está inicializado de manera que $p(i) = i$, $i = 1, 2, \dots, n$. Este vector nos sirve para luego crear la matriz permutación P.

$$P_{\pi} = \begin{bmatrix} \mathbf{e}_{\pi(1)} \\ \mathbf{e}_{\pi(2)} \\ \mathbf{e}_{\pi(3)} \\ \mathbf{e}_{\pi(4)} \\ \mathbf{e}_{\pi(5)} \end{bmatrix} = \begin{bmatrix} \mathbf{e}_1 \\ \mathbf{e}_4 \\ \mathbf{e}_2 \\ \mathbf{e}_5 \\ \mathbf{e}_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

Figura 6: Matriz de Permutaciones

Como se había mencionado, este algoritmo es prácticamente el mismo que se utiliza para realizar la triangulación pero agregan algunas cosas más. Una de ellas es que al realizar el pivoteo, en caso de que la matriz no sea diagonal dominante, además de pasarle la matriz A a la función que realiza el pivoteo parcial, también se le pasa la matriz U y el *vector_permutacion* para que en caso de realizar una permutación de filas también se realice dicho cambio en esta matriz y vector. Otra diferencia es que, al recorrer las filas de una columna en un paso K para poner ceros debajo de la diagonal, lo que se hace es guardar los multiplicadores calculados en la matriz U a medida que estos se van calculando, colocando también 1's en la diagonal.

Recordar que para realizar la triangulación se usan dos(2) ciclos FOR. Una vez que se sale de ambos ciclos lo que se hace es poner un 1 adicional faltante en la posición a_{nn}

de la matriz U. Luego, se sabe que la matriz L es igual a la matriz A ya triangulada así que lo único que se hace es devolver A triangulada como L.

Diferencias entre Crout, Doolittle y Cholesky

El método de Factorización LU es una manera de descomponer la matriz con el fin de disminuir la cantidad de operaciones y así también buscar una solución con un error menor. Dentro de la Factorización LU existen diferentes métodos. Ellos son: Crout, Doolittle y Cholesky.

Si $l_{ii} = 1$ se denomina factorización de Doolittle.

Si $u_{ii} = 1$ se denomina factorización de Crout.

Si $l_{ii} = u_{ii}$ se denomina factorización de Cholesky (para aplicar Cholesky es necesario que la matriz sea simétrica definida positiva)

Si bien siempre es posible realizar factorización LU por los métodos de Doolittle y Crout, lo primero que tiene que hacer uno es ver si la matriz cumple con los requerimientos necesarios para ser factorizada mediante el método de Cholesky. Es decir, si es simétrica y definida positiva, es factible factorizar mediante Cholesky, un método más eficiente que los otros.

Entonces, se sabe que la eliminación de Gauss equivale a factorizar la matriz de coeficientes $A = L U$ como el producto de una matriz triangular inferior unitaria L y una triangular superior U. A esta factorización LU se la denomina de Doolittle cuando la diagonal de la matriz L vale 1 ($l_{ii} = 1$). También se puede factorizar la matriz $A = L' U'$ en una matriz triangular inferior L' y una matriz triangular superior unitaria U' donde ahora se pretende que la diagonal principal de la matriz U sea 1, es decir $u_{ii} = 1$. A este procedimiento se lo denomina factorización LU de Crout.

Por último, cuando se cuenta con matrices simétricas y se aplica la factorización a LU, la matriz U no corresponde a L^t (L traspuesta). Toda matriz real A que tenga factorización LU única, y que sea simétrica y definida positiva, entonces tiene una factorización única de la forma $A = L L^t$, donde L es una matriz triangular inferior con diagonal positiva, y se dice que A tiene factorización de Cholesky.



Costo computacional de cada algoritmo

Se procede con el trabajo resolviendo con ambos algoritmos el problema en cuestión. Como ya se detalló recientemente, se utilizaron como métodos de resolución tanto la Eliminación Gaussiana y la Factorización LU, teniendo en cuenta que para la esta última ya se contaba con la triangulación hecha, así que lo único que restaba hacer en este caso es resolver las ecuaciones $Ly=Pb$ y $Ux=y$ mediante sustitución directa e inversa. Para evaluar el costo computacional de cada algoritmo se fueron tomando los tiempos de ejecución de cada uno utilizando las funciones **tic()** **toc()** de Octave y se los representó en una tabla para poder compararlos y obtener conclusiones. La tabla que se obtuvo se muestra en la figura 7:

TIEMPO DE EJECUCIÓN [segundos]	
Eliminación de Gauss	Descomposición LU
0.004241	0.001587
0.006732	0.001605
0.011841	0.001629
0.005507	0.001579
0.011819	0.001634

¿Resultados?

Figura 7



Como se puede observar en la tabla, los tiempos de la Descomposición LU son menores que los del Método de Eliminación de Gauss ya que en la Descomposición LU, una vez que se hace la triangulación y se obtienen L y U, la resolución por sustitución directa e inversa de los sistemas $Ly=Pb$ y $Ux=y$ son mucho menos costosas que si se resuelve el sistema $Ax=b$ utilizando el Método de Eliminación de Gauss.

Múltiples combinaciones de carga

Para resolver el problema de múltiples combinaciones de carga, se utiliza la Matriz obtenida con el método de Factorización LU y adoptando $F = 1000 * (1 + i/10)$ y $G = 100 * (1 + i/10)$, con $i=0,1,...,9$.

En primer lugar se utilizó la función **imagesc()** y **colorbar()** para realizar el gráfico. Se pasa a la matriz en dicha función y a cada valor se le asigna un color diferente.

Los valores de esta matriz son los valores N_{12} que se obtienen al resolver la ecuación $Ax=b$ con diferentes combinaciones F y G. N_{12} es el primer elemento del vector x. Luego se realiza el mismo procedimiento para N_{23} y N_{13} . A continuación se muestra como se obtuvieron los gráficos.

Las combinaciones son así: F varía en cada fila y G varía en cada columna.

Para el valor N_{12} de la posición (1,1) de la matriz uso $i=0$ para calcular F y $i=0$ para calcular G


Para el valor N_{12} de la posición (1,2) de la matriz uso $i=0$ para calcular F y $i=1$ para calcular G

Para el valor N_{12} de la posición (1,3) de la matriz uso $i=0$ para calcular F y $i=2$ para calcular G

Para el valor N_{12} de la posición (1,4) de la matriz uso $i=0$ para calcular F y $i=3$ para calcular G, y así sucesivamente.

El formato de la posición es (fila,columna). Luego de terminar con la fila 1 y con las 9 columnas de G, se comienza a pasar los datos de la segunda fila para F con $i=1$. Se realiza el mismo procedimiento hasta completar una matriz de 10x10 y así quede la matriz N_{12} . Finalmente, se hace un procedimiento análogo para obtener las matrices N_{23} y N_{13} .

En la tabla de la figura 8 se pueden observar los distintos valores que van tomando tanto F como G a medida que corren i. Una de las dos fuerzas resulta ser negativa ya que como se mostró en el diagrama de fuerzas, una va en contra de nuestro sistema de coordenadas.



i	F	-G
0	1000	-100
1	1100	-110
2	1200	-120
3	1300	-130
4	1400	-140
5	1500	-150
6	1600	-160
7	1700	-170
8	1800	-180
9	1900	-190

Figura 8

Continuando con el análisis. En la figura 9 se puede observar el resultado para la barra N_{12} con diferentes valores de F y G. Los valores de mayor magnitud se expresan en un color violeta y a medida que baja la intensidad de la fuerza van tomando un color amarillento en forma degradé. Se puede observar los valores resultantes de N_{12} a medida que varían F y G. La zona de mayor esfuerzo de N_{12} es cuando F tomo i grande y G uno chico.

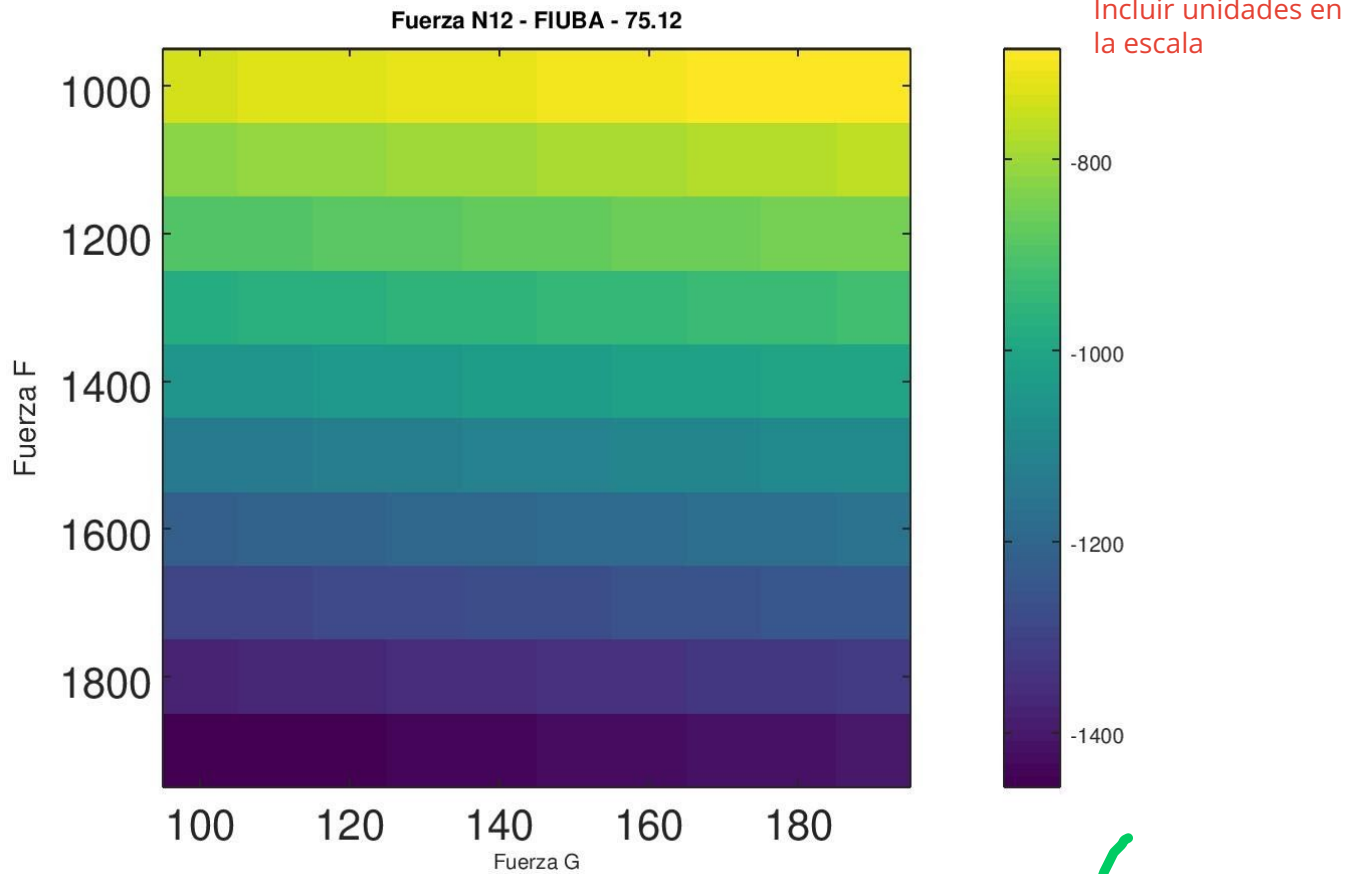


Figura 9: Gráfico de fuerzas aplicadas sobre la barra N_{12}

Se realiza un procedimiento análogo con N_{13} , aumentando el esfuerzo a medida que crecen en módulo los valores tanto de F como de G. Consecuentemente, el momento en el que hay menos fuerza en esta barra es cuando $F=1000$ y $G=100$.

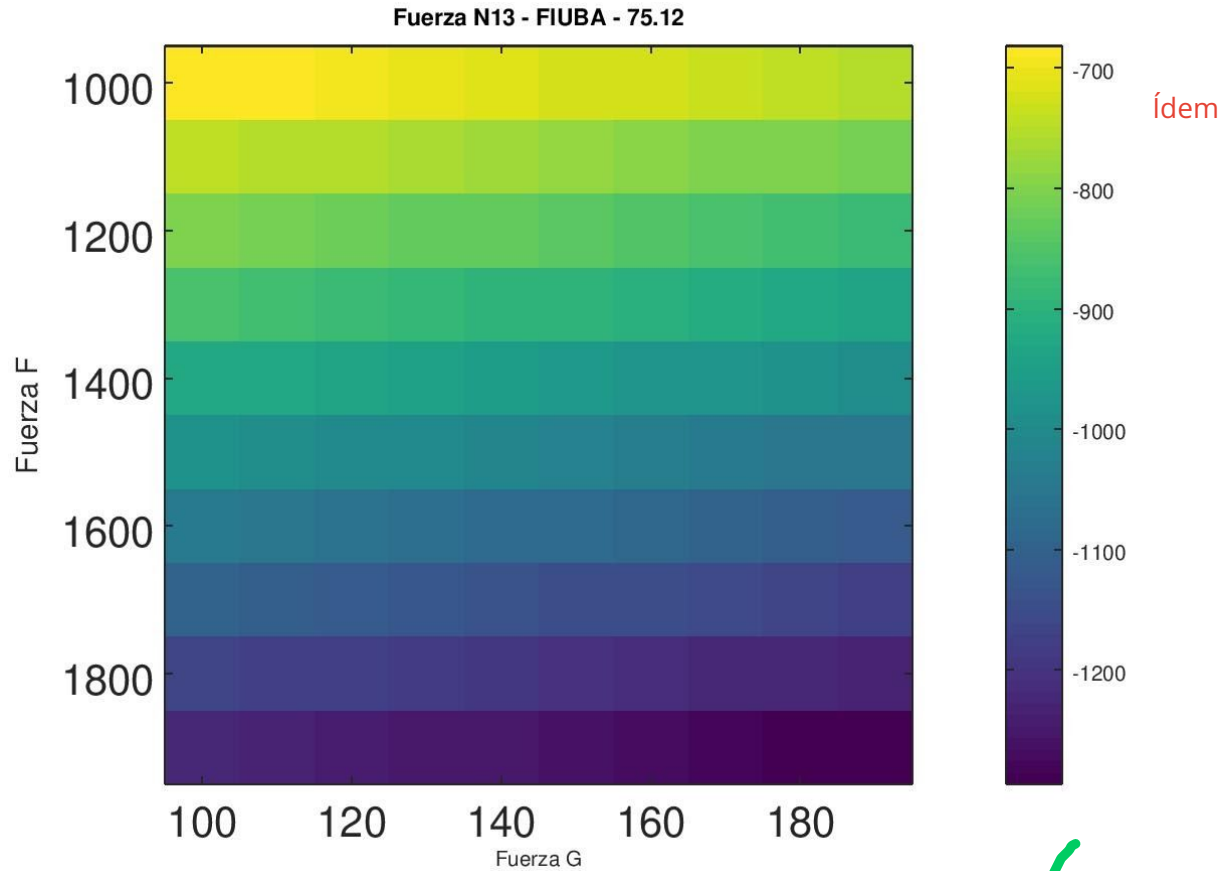


Figura 10: Gráfico de fuerzas aplicadas sobre la barra N_{13}

Finalmente, para N_{23} se toman los colores claros como los de mayor esfuerzo en módulo. En esta matriz se puede observar que el mayor trabajo se hace también cuando la fuerza $F=1000$ y $G=100$, en la posición (10,10) de la matriz.

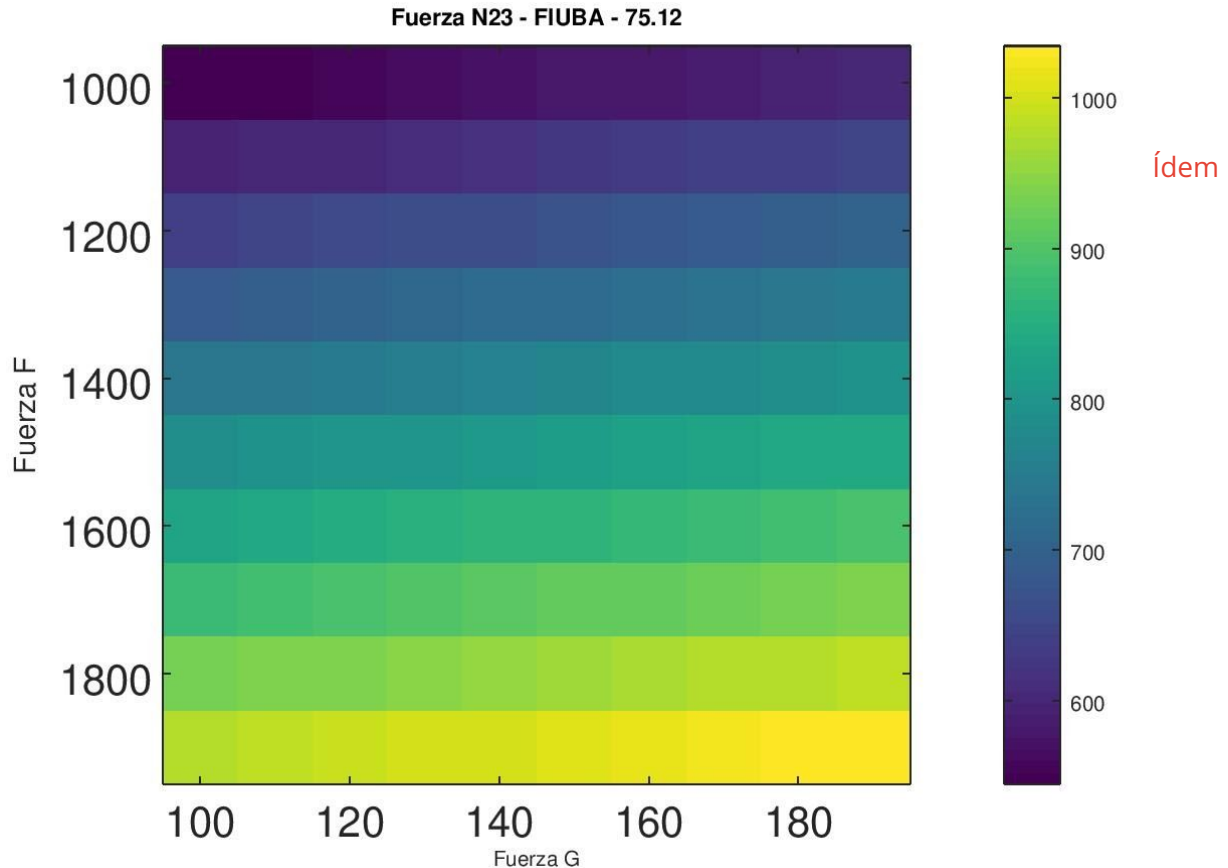


Figura 11: Gráfico de fuerzas aplicadas sobre la barra N_{23}

Perturbaciones experimentales

Para concluir con el trabajo. Se fijó un valor de F de 1000 con un error de 200 y un $G=100$ con 10 de error con el fin de calcular los esfuerzos resultantes con sus respectivas cotas de error. Para realizar esto se aplicaron perturbaciones experimentales. Para cada N se realizó un análisis diferente. Los resultados obtenidos fueron expresados en tablas. Se utilizaron distintos colores para diferenciar qué fuerza se dejaba fija y cuál variaba.

Se optó por el color rojo a la hora de dejar fija la fuerza F , y el verde cuando la fuerza que se mantenía constante era G . Se obtuvieron las siguientes tablas ilustradas en las figuras 12, 13 y 14 junto con el cálculo de la cota obtenida utilizando perturbaciones experimentales.

No son figuras, son tablas

$$I(F, G) = F(x, y)$$

$$\Delta I = \left| \frac{\partial I}{\partial x} \right|_P \Delta x + \left| \frac{\partial I}{\partial y} \right|_P \Delta y = \left| \frac{\partial F(x, y)}{\partial x} \right|_P \Delta x + \left| \frac{\partial F(x, y)}{\partial y} \right|_P \Delta y$$

F	G	Fuerza N12
800	90	-584.74505595415
1000	100	-738.4540077320878
1200	110	-892.1629595100261
800	100	-578.7269057226295
1000	110	-732.4358575005674
1200	90	-904.199259973067
80	110	2.308811742941876
1000	90	-744.4721579636084



Figura 12

$$\frac{\delta F(x,y)}{\delta x} \approx \frac{\delta I}{\delta F} = \frac{-732,4358575005674 - (-892,1629595100261)}{1000-1200} = -0,79863551$$

$$\frac{\delta F(x,y)}{\delta y} \approx \frac{\delta I}{\delta G} = \frac{-578,7269057226295 - (-584,74505595415)}{100-90} = 0,6018150232$$

$$\Delta I = 0,79863551 \cdot 200 + 0,6018150232 \cdot 10 = 165,7452522$$

Adoptando un solo dígito para la cota nos queda que $\Delta I = 166$

$$\Delta I = 166 \Rightarrow 0,5 \times 10^{-t-1} < \Delta x < 0,5 \times 10^{-t} \Rightarrow 50 < \Delta x < 500 \Rightarrow t = -3$$

$$I = \hat{I} + \Delta I$$

Falta expresar los resultados correctamente redondeados

F	G	Fuerza N13
800	90	-553.329214425895
1000	100	-681.6785741567777
1200	110	-810.0279338876602
800	100	-561.315569526368
1000	110	-689.6649292572506
1200	90	-794.0552236867144
80	110	-135.9951079573661
1000	90	-673.6922190563048



Figura 13

$$\frac{\delta F(x,y)}{\delta x} \approx \frac{\delta I}{\delta F} = \frac{-689,6649292572506 - (-810,0279338876602)}{1000-1200} = -0,6018150232$$

$$\frac{\delta F(x,y)}{\delta y} \approx \frac{\delta I}{\delta G} = \frac{-561,315569526368 - (-553,329214425895)}{100-90} = -0,79863551$$

$$\Delta I = 0,6018150232 \cdot 200 + 0,79863551 \cdot 10 = 128,3493597$$

Adoptando un solo dígito para la cota nos queda que $\Delta I = 128$

Falta expresar los resultados correctamente redondeados

$$\Delta I = 128 \Rightarrow 0,5x10^{-t-1} < \Delta x < 0,5x10^{-t} \Rightarrow 50 < \Delta x < 500 \Rightarrow t = -3$$

$$I = \hat{I} + \Delta I$$

F	G	Fuerza N23
800	90	441.9083593870926
1000	100	544.4127157600094
1200	110	646.9170721329264
800	100	448.2865461661775
1000	110	550.7909025390945
1200	90	634.1606985747563
80	110	108.6105224074677
1000	90	538.0345289809245



Figura 14

$$\frac{\delta F(x,y)}{\delta x} \approx \frac{\delta I}{\delta F} = \frac{550,7909025390945 - (646,9170721329264)}{1000-1200} = 0,480630848$$

$$\frac{\delta F(x,y)}{\delta y} \approx \frac{\delta I}{\delta G} = \frac{448,2865461661775 - (441,9083593870926)}{100-90} = 0,6378186779$$

$$\Delta I = 0,480630848 \cdot 200 + 0,6378186779 \cdot 10 = 102,5043564$$

Adoptando un solo dígito para la cota nos queda que $\Delta I = 103$

$$\Delta I = 103 \Rightarrow 0,5x10^{-t-1} < \Delta x < 0,5x10^{-t} \Rightarrow 50 < \Delta x < 500 \Rightarrow t = -3$$

$$I = \hat{I} + \Delta I$$

Falta expresar los resultados correctamente redondeados

Como en todos los casos $t < 0$ se tendrá el último dígito significativo en la posición 10^{-t} y 1 dígito medianamente significativo en la posición 10^{-t-1} .

En cada tabla se puede observar los distintos valores que toman las fuerzas internas de la barra a medida que cambia el valor de F y G. El rango de valores es demasiado grande, por lo que se puede decir que el riesgo es grande también. Lo ideal sería trabajar con errores mucho menores para tener una mayor exactitud de los esfuerzos que realizan las barras.

3. Conclusión

Como conclusión se puede decir que se fue variando tanto las cargas como los métodos de resolución para poder ver cómo actúan las fuerzas internas y resultantes en una estructura.

Se pudo observar qué métodos son más eficaces o más rápidos (menos costo computacional) a la hora de realizar este tipo de operaciones. También, cómo dependen las fuerzas una de otra en el momento que se varía alguna de ellas. En el caso analizado se puede ver claramente (si se mira la posición (1,10), por ejemplo, en todos los gráficos de fuerzas de las barras) que si tenemos una fuerza F chica constante ($F = 1000$, por ejemplo) y G grande, la fuerza en módulo que se hace en las barras no es tan grande, pero si se aumenta F se nota un considerable aumento en la fuerza recibidas por las 3 barras.

Por último, se pudo trabajar en cómo responden las reacciones frente al cálculo de los errores y el riesgo de imprecisión que uno puede tener a la hora de calcular los esfuerzos de la barra cuando la cota de error es demasiado grande.

4. Referencias

<https://sites.google.com/site/procesosnumericoscasa1/metodo-cROUT>

<https://sites.google.com/site/procesosnumericosrafaelrincon/2-practica-2/1-marco-teorico/metodos-directos/cholesky>

5. Anexo: Código de programación

Matriz es diagonal dominante

```
% ES DIAGONAL DOMINANTE Devuelve true si la diagonal es dominante
function resultado = es_diagonal_dominante(A)
    dim_fil = rows(A);
    dim_col = columns(A);
    diagonal_es_dominante = true;

    for i=1:dim_fil
        valor_diagonal = abs(A(i,i));
        suma = 0;

        % Suma todos los valores de la fila i en valor absoluto sin
        % incluir el valor de la diagonal
        for j=1:dim_col-1 % excluye termino independiente
            if (j != i)
                suma = suma + abs(A(i,j));
            endif
        endfor

        if (suma > valor_diagonal)
            diagonal_es_dominante = false;
            break;
        endif
    endfor
    resultado = diagonal_es_dominante;
endfunction
```

Pivoteo parcial en una matriz

```
% PIVOTEO_PARCIAL aplica pivoteo parcial a la matriz A.
%
% A - matriz A. A podria ser la matriz ampliada.
% L - matriz L de la descomposicion LU.
% vector_permutacion - almacena los cambios de filas que se realizan.
% fila - indice de la fila en la que se encuentra el pivote
% columna - indice de la columna en la que se encuentra el pivote
% lu - es 0 si la funcion pivoteo_parcial se llama desde la funcion triangular,
%      es 1 si se llama desde la funcion factorizar.
function [A,L,v] = pivoteo_parcial(A, L, vector_permutacion, fila, columna, lu=0)
    dim_fila = rows(A);
    pivote = abs(A(fila,columna));
    fila_pos_max = 0;

    % Verifica si hay un valor mayor al pivote en cada paso k y guarda su posicion(fila) en
    fila_pos_max
    for i=fila+1:dim_fila
        valor_fila = abs(A(i,columna));
        if (valor_fila > pivote)
            pivote = valor_fila;
            fila_pos_max = i;
        endif
    endfor

    % Intercambia las filas si el nuevo pivote no es el pivote actual.
    % Se permuta la fila "fila" por la "fila_pos_max".
    if (fila_pos_max != fila && fila_pos_max != 0)

        % Permuta filas del vector permutacion
        vector_permutacion([fila fila_pos_max]) = vector_permutacion([fila_pos_max fila]);

        %Permuta filas de la matriz A
        A([fila fila_pos_max],:) = A([fila_pos_max fila],:);

        % Permuta filas del vector L
        if (lu == 1)
            L([fila fila_pos_max],:) = L([fila_pos_max fila],:);
        endif
    endif
    A = A;
    L = L;
    v = vector_permutacion;
endfunction
```

Triangulador de matrices

```
% TRIANGULAR triangula la matriz A usando metodo de eliminacion de gauss
%
% A_ampliada - es la matriz ampliada A|b
function A_triangularada = triangular(A_ampliada)
    diagonal_es_dominante = es_diagonal_dominante(A_ampliada);
    v = [1:rows(A_ampliada)];
    L = [];

    for j=1:rows(A_ampliada)-1
        % Si A es diagonal dominante no se aplica la permutacion
        if (diagonal_es_dominante == 0)
            [A_ampliada,L,v] = pivoteo_parcial(A_ampliada,L,v,j,j);
        endif

        % Calcula el multiplicador y realiza la resta de las filas
        for i=j+1:rows(A_ampliada)
            multiplicador = A_ampliada(i,j)/A_ampliada(j,j);
            A_ampliada(i,:) = A_ampliada(i,:) - multiplicador*A_ampliada(j,:);
        endfor
    endfor
    A_triangularada = A_ampliada;
endfunction
```

Sustitución inversa de matriz triangular superior

```
% SUSTITUCION_INVERSA realiza sustitucion inversa y devuelve un vector que
% contiene el valor de las incognitas. A debe estar triangulada
function vector_soluciones = sustitucion_inversa(A_ampliada)
    soluciones = zeros(rows(A_ampliada),1);

    for i=rows(A_ampliada):-1:1
        suma = 0;
        % Reemplaza las incognitas conocidas y los paso al segundo miembro
        for j=i+1:columns(A_ampliada)-1
            if (j != i)
                suma = suma + A_ampliada(i,j)*soluciones(j);
            endif
        endfor

        % Calcula el valor de la incognita y se guarda en el vector soluciones
        b = A_ampliada(i,columns(A_ampliada));
        soluciones(i) = (b - suma) / A_ampliada(i,i);
    endfor
    vector_soluciones = soluciones;
endfunction
```

Realiza una sustitución directa

```
% SUSTITUCION_DIRECTA realiza sustitucion directa y devuelve un vector que
% contiene el valor de las incognitas
%
% A_ampliada - es la matriz A|b. A debe estar triangulada inferiormente, esto es
% con cero por encima de la diagonal.
function vector_soluciones = sustitucion_directa(A_ampliada)
    soluciones = zeros(rows(A_ampliada),1);

    for i=1:rows(A_ampliada)
        suma = 0;

        % Reemplaza las incognitas conocidas y los paso al segundo miembro
        for j=1:i-1
            if (j != i)
                suma = suma + A_ampliada(i,j)*soluciones(j);
            endif
        endfor

        % Calcula el valor de la incognita y la guarda en el vector soluciones
        b = A_ampliada(i,columns(A_ampliada));
        soluciones(i) = (b - suma) / A_ampliada(i,i);
    endfor
    vector_soluciones = soluciones;
endfunction
```

Creador matriz de permutación

```
% CREAR_MATRIZ_PERMUTACION crea la matriz de permutacion y la devuelve
%
% vector_permutacion - contiene la permutacion de filas que se hizo durante el
% pivoteo parcial.
function P = crear_matriz_permutacion(vector_permutacion,dim_fil,dim_col)
    P = zeros(dim_fil,dim_col);

    for i=1:columns(vector_permutacion)
        P(i,vector_permutacion(i)) = 1;
    endfor
    P = P;
endfunction
```

Descomposición LU

```
% FACTORIZAR realiza la factorizacion LU y devuelve tres(3) matrices: L,U y P
% L es una matriz triangular inferior
% U es una matriz triangular superior
% P es la matriz de permutacion
%
% A - es la matriz A, no es la matriz ampliada.
function [L,U,P] = factorizar(A)
    diagonal_es_dominante = es_diagonal_dominante(A);
    dim_fil = rows(A);
    dim_col = columns(A);
    L = zeros(dim_fil,dim_col);
    U = [];
    vector_permutacion = [1:dim_fil];

    for j=1:dim_col-1
        % Si A es diagonal dominante no es necesario aplicar el pivoteo
        if (diagonal_es_dominante == 0)
            [A,L,vector_permutacion] = pivoteo_parcial(A,L,vector_permutacion,j,j,1);
        endif

        % Calcula el multiplicador y hace la resta de filas
        for i=j+1:dim_fil
            multiplicador = A(i,j)/A(j,j);
            A(i,:) = A(i,:) - multiplicador*A(j,:);

            L(j,j) = 1;
            L(i,j) = multiplicador;
        endfor
    endfor
    L(rows(L),columns(L)) = 1; % Coloca el ultimo 1 que falta en la diagonal de L

    U = A;
    L = L;
    P = crear_matriz_permutacion(vector_permutacion,dim_fil,dim_col);
endfunction
```


Graficador de fuerzas N_{12} , N_{23} , N_{13}

```
function resultado = graficar_fuerza(matriz,titulo,grafico_numero)
% Graficador
imagesc([100 190], [1000 1900], matriz);
colorbar();

% Título
title(titulo);

% Rótulos de ejes
xlabel("Fuerza G",'fontsize',10)
ylabel("Fuerza F",'fontsize',14)

% Tamaño de letra de los n
set(gca,'fontsize',20); % sets font of numbers on axes

if (grafico_numero == 12)
    print "punto_d_grafico_fuerza12.jpg";
elseif (grafico_numero == 23)
    print "punto_d_grafico_fuerza23.jpg";
elseif (grafico_numero == 13)
    print "punto_d_grafico_fuerza13.jpg";
endif
endfunction
```

Código de resolución de punto C

```
clear all; close all; 1;
% Convierte los grados 53 y 37 a radianes
ALPHA = (53*pi)/180;
BETA = (37*pi)/180;

A = [-cos(ALPHA) 0 cos(BETA) 0 0 0;
-sin(ALPHA) 0 -sin(BETA) 0 0 0;
cos(ALPHA) 1 0 1 0 0;
sin(ALPHA) 0 0 0 1 0;
0 -1 -cos(BETA) 0 0 0 ;
0 0 sin(BETA) 0 0 1];
```

```
F = 1000;
G = 100;
b = [-G; F; 0; 0; 0; 0];

disp("EJERCICIO C:\n");

% Resuelve utilizando gauss
tic;
A(:,7) = b; # Agrega el vector b a la columna 7 de la matriz A para formar la matriz ampliada A|b
A_triangular = triangular(A);
x = sustitucion_inversa(A_triangular);
tiempo_gauss = toc;

% Resuelve utilizando descomposicion LU
[L,U,P] = factorizar(A(:,1:6)); % factorizar recibe la matriz no-ampliada entonces le quito la ultima columna

% Resuelve Ly=Pb
tic;
L(:,7) = P*b; % Agrega el vector P*b en la columna 7 de L. O sea, es la matriz ampliada L|P*b
y = sustitucion_directa(L);

% Resuelve Ux=y
U(:,7) = y; % Analogamente, es la matriz ampliada U|y
x = sustitucion_inversa(U);
tiempo_lu = toc;

printf('Eliminacion de Gauss tardó: %f segundos\n', tiempo_gauss);
printf('Descomposicion LU tardó: %f segundos\n', tiempo_lu);
```

Código de resolución de punto D

```
clear all;
close all;

1; %Le dice a octave que esto no es un function file

% Convierte los grados 53 y 37 a radianes
ALPHA = (53*pi)/180;
BETA = (37*pi)/180;

A = [-cos(ALPHA) 0 cos(BETA) 0 0 0;
     -sin(ALPHA) 0 -sin(BETA) 0 0 0;
     cos(ALPHA) 1 0 1 0 0;
```

```
sin(ALPHA) 0 0 0 1 0;
0 -1 -cos(BETA) 0 0 0 ;
0 0 sin(BETA) 0 0 1];
```

[L,U,P] = factorizar(A); % Como factorizar recibe la matriz A y no la ampliada A|b, le quito b.

% Arma para cada fuerza una matriz de 10x10

```
matriz_fuerzaN12 = [];
```

```
matriz_fuerzaN23 = [];
```

```
matriz_fuerzaN13 = [];
```

```
for i=0:9 % fila
```

```
    F = 1000*(1 + i/10);
```

```
    for j=0:9 % columna
```

```
        G = 100*(1 + j/10); % G varia en las columnas
```

```
        b = [-G; F; 0; 0; 0; 0];
```

```
        % Resuelve Ly=Pb
```

```
        L(:,7) = P*b;
```

```
        y = sustitucion_directa(L);
```

```
        % Resuelve Ux=y
```

```
        U(:,7) = y;
```

```
        x = sustitucion_inversa(U);
```

```
        % Guardo los valores en sus respectivas matrices
```

```
        matriz_fuerzaN12(i+1, j+1) = x(1);
```

```
        matriz_fuerzaN23(i+1, j+1) = x(2);
```

```
        matriz_fuerzaN13(i+1, j+1) = x(3);
```

```
    endfor
```

```
endfor
```

```
graficar_fuerza(matriz_fuerzaN12,"Fuerza N12 - FIUBA - 75.12",12);
```

```
graficar_fuerza(matriz_fuerzaN23,"Fuerza N23 - FIUBA - 75.12",23);
```

```
graficar_fuerza(matriz_fuerzaN13,"Fuerza N13 - FIUBA - 75.12",13);
```

Código de resolución de punto E

```
clear all; close all; 1;
```

```
% Convierte los grados 53 y 37 a radianes
```

```
ALPHA = (53*pi)/180;
```

```
BETA = (37*pi)/180;
```

```
A = [-cos(ALPHA) 0 cos(BETA) 0 0 0;
```

```
    -sin(ALPHA) 0 -sin(BETA) 0 0 0;
```

```
    cos(ALPHA) 1 0 1 0 0;
```

```

sin(ALPHA) 0 0 0 1 0;
0 -1 -cos(BETA) 0 0 0 ;
0 0 sin(BETA) 0 0 1];

F = [800 1000 1200 800 1000 1200 80 1000];
G = [90 100 110 100 110 90 110 90];

% Arma la matriz ampliada de A
A(1,7:14) = (-1)*G;
A(2,7:14) = F;

% La primera columna de esta matriz llamada soluciones son los valores de F, la
% segunda columna son los valores de G. Las 3 columnas restantes son las fuerzas
% (soluciones de haber resuelto la matriz). La columna 3 tiene valores de N12,
% la 4 de N23 y la 5 de N13.
soluciones = [];
soluciones(:,1) = transpose(F);
soluciones(:,2) = transpose(G);

A_triangular = triangular(A); % Triangula una matriz de 14 columnas
A = A_triangular(:,1:6);

% Resuelvo 8 sistemas Ax=b
for i=1:8
    A(:,7) = A_triangular(:,i+6); % Arma la matriz ampliada Ax|b
    x = sustitucion_inversa(A); % Encuentra las soluciones

    % Guardo las soluciones
    soluciones(i,3) = x(1); % Guarda la fuerza N12
    soluciones(i,4) = x(2); % Guarda la fuerza N23
    soluciones(i,5) = x(3); % Guarda la fuerza N13
endfor

% Calcula las perturbaciones
error_n12_f_fijo = abs( (soluciones(4,3) - soluciones(1,3)) / (soluciones(4,2) - soluciones(1,2))
);
error_n12_g_fijo = abs( (soluciones(5,3)-soluciones(3,3)) / (soluciones(5,1) - soluciones(3,1))
);
error_n12 = round(error_n12_g_fijo*200 + error_n12_f_fijo*10) % Calcula el error absoluto

error_n23_f_fijo = abs( (soluciones(4,4) - soluciones(1,4)) / (soluciones(4,2) - soluciones(1,2))
);
error_n23_g_fijo = abs( (soluciones(5,4)-soluciones(3,4)) / (soluciones(5,1) - soluciones(3,1))
);
error_n23 = round(error_n23_g_fijo*200 + error_n23_f_fijo*10) % Calcula el error absoluto

error_n13_f_fijo = abs( (soluciones(4,5) - soluciones(1,5)) / (soluciones(4,2) - soluciones(1,2))
);

```

```
error_n13_g_fijo = abs( (soluciones(5,5)-soluciones(3,5)) / (soluciones(5,1) - soluciones(3,1))  
);
```

```
error_n13 = round(error_n13_g_fijo*200 + error_n13_f_fijo*10) % Calcula el error absoluto
```

```
% Exporta datos a un archivo .csv para poder armar la tabla
```

```
csvwrite ("punto_e_tabla.csv",soluciones);
```