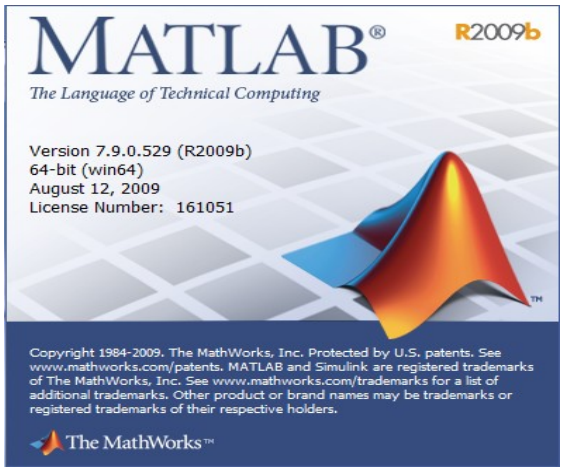
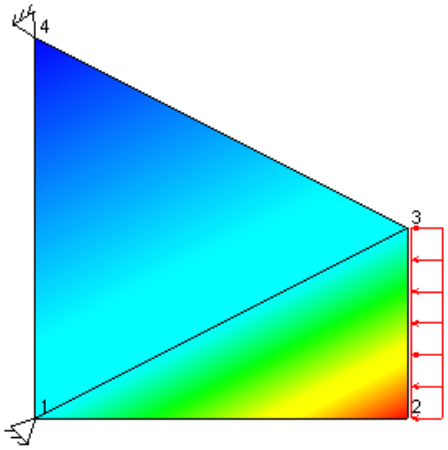
PROGRAMACIÓN DE UNA INTERFAZ GRÁFICA DE MATLAB PARA LA APLICACIÓN DEL MÉTODO DE LOS ELEMENTOS FINITOS





Pedro Luis Fuentes Camas
Proyecto Fin de Carrera
Ingeniería Técnica Industrial, especialidad Mecánica Industrial
Tutor: D. Enrique Nieto García
17/01/2012

ÍNDICE

1.- GENERALIDADES

La realización del presente proyecto se complementa con el uso del programa de distribución libre MEFI (Método de Elementos Finitos en Ingeniería), el cual es capaz de resolver gran variedad de problemas de cálculo de estructuras así como elementos planos o tridimensionales.

Nos centramos en la parte de resolución de problemas mediante el uso del método de los elementos finitos, concretamente en el uso de mallado triangular. El programa MEFI es capaz de resolver estos problemas, facilitando como resolución de los mismos valores "finales" de cargas y desplazamientos nodales.

El proyecto trata de una aplicación didáctica, orientada al análisis de la evolución numérica que sufren los distintos parámetros de entrada, para finalmente, obtener un determinado campo de tensiones y deformaciones, así como reacciones en los nodos fijos y desplazamientos de aquellos que lo tengan permitido.

El programa ha sido realizado en Matlab, concretamente se ha diseñado una interfaz gráfica (GUI) que permite a cualquier usuario sin conocimiento de éste, hacer uso del mismo. Presenta distintas ventanas, en las que se solicita al usuario que introduzca una serie de datos, tales como las posiciones nodales, el módulo elástico, el coeficiente de Poisson y el espesor de la placa que se analiza. Durante la ejecución del mismo, se solicitarán más datos, como aquellos necesarios para calcular la matriz de rigidez global de la estructura, o valores previos calculados en este software para el cálculo de tensiones y deformaciones, lo que implica un conocimiento por parte del usuario de los cálculos que se han de realizar. Este programa está centrado en la asignatura de 'Cálculo Avanzado de Estructuras' de tercer curso de la especialidad de mecánica.

El programa ha sido denominado "MatM" ya que muestra al usuario el trabajo matricial que hay detrás de un programa como es el MEFI.

2.- FUNDAMENTOS

Nos encontramos ante un problema de deformación bidimensional, este tipo de problemas de elasticidad son muy frecuentes en ingeniería, y de hecho son los primeros en los que se aplicó el Método de los Elementos Finitos (FEM). En este caso el medio continuo que se analiza es plano, y se considera situado en el plano XY. Se ha denominado *t* al espesor del dominio en su dirección transversal, el cual, normalmente se considera despreciable frente a las dimensiones del dominio en el plano XY.

La posición de un punto está definida por dos coordenadas (x,y) y su deformación tiene dos componentes u(x,y), v(x,y) en las direcciones x,y respectivamente. El campo de deformaciones es por tanto un vector:

$$u_e = \begin{Bmatrix} u(x,y) \\ v(x,y) \end{Bmatrix}$$

En primer lugar se tratará de obtener las ecuaciones que rigen el comportamiento de un elemento triangular como el de la figura siguiente:

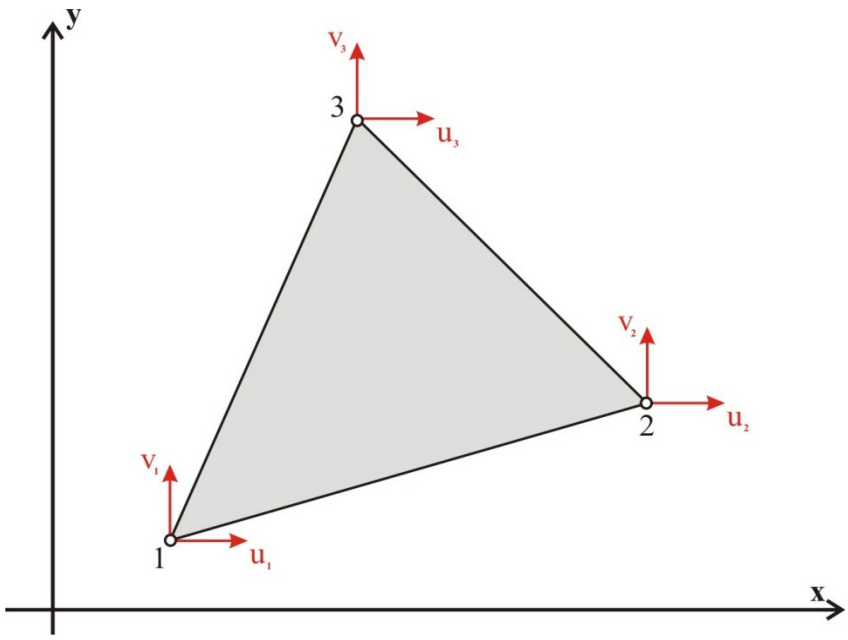


Ilustración 1: Desplazamientos nodales del elemento triangular de 3 nodos

Considerando un elemento triangular con un nodo en cada vértice y desplazamientos

nodales u_1, v_1, u_2, v_2, u_3, v_3, el vector desplazamientos del elemento finito $\{u_e\}$ puede expresarse aproximadamente por las funciones de desplazamientos

$$\begin{aligned} u(x,y) &= \alpha_1 + \alpha_2 x + \alpha_3 y \\ v(x,y) &= \alpha_4 + \alpha_5 x + \alpha_6 y \end{aligned} \quad (1.1)$$

que son polinomios que tienen un número total de parámetros α_i igual a 6, que es el número de grados de libertad del elemento. Las funciones de desplazamiento u(x,y), v(x,y) son polinomios completos de primer grado y, por ello, el elemento considerado es un *elemento lineal*. Dichas funciones pueden expresarse en la forma matricial:

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{Bmatrix} 1 & x & y & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x & y \end{Bmatrix} \cdot \begin{Bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \end{Bmatrix} \quad (1.2$$

es decir,

$$\{u_e\} = [P] * \{\alpha\} \qquad (1.3)$$

siendo

$$u_e = \begin{Bmatrix} u(x,y) \\ v(x,y) \end{Bmatrix} \quad (1.4)$$

$$[P] = \begin{Bmatrix} 1 & x & y & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x & y \end{Bmatrix} \quad (1.5)$$

$$\{\alpha\} = \begin{Bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \end{Bmatrix} \quad (1.6)$$

Los seis coeficientes α_i se hallan particularizando en la ecuación (1.2) los desplazamientos u, v en los tres nodos, obteniéndose :

$$\{\delta_e\} = [C] * \{\alpha\} \qquad (1.7)$$

Que de forma matricial quedaría:

$$\begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{Bmatrix} = \begin{Bmatrix} 1 & x_1 & y_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_1 & y_1 \\ 1 & x_2 & y_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_2 & y_2 \\ 1 & x_3 & y_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_3 & y_3 \end{Bmatrix} \cdot \begin{Bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \end{Bmatrix} \quad (1.8)$$

De la ecuación (1.7), se deduce:

$$\{\alpha\} = [C]^{-1} * \{\delta_e\} \qquad (1.9)$$

Que sustituida en (1.3), determina la función de desplazamientos:

$$\{u_e\} = [P] * [C]^{-1} * \{\delta_e\} \qquad (1.10)$$

Comparando (1.10) con:

$$\{u_e\} = [N_e] * \{\delta_e\} \qquad (1.11)$$

Se obtiene la matriz de interpolación del elemento:

$$[N_e] = [P] * [C]^{-1} \qquad (1.12)$$

Que determina las funciones de interpolación $N_i(x,y)$. Ahora bien, los desplazamientos $u(x,y)$ tienen que depender únicamente de los desplazamientos nodales u_1, u_2, u_3. De la misma forma, los desplazamientos $v(x,y)$ dependerán solamente de los desplazamientos nodales v_1, v_2, v_3 y, además, ambas dependencias han de ser iguales. En consecuencia, al utilizar las funciones de interpolación $N_i(x,y)$, las funciones de desplazamiento del elemento

$$\begin{aligned} u(x,y) &= N_1 u_1 + N_2 u_2 + N_3 u_3 \\ v(x,y) &= N_1 v_1 + N_2 v_2 + N_3 v_3 \end{aligned} \quad (1.13)$$

es decir:

$$\begin{aligned} u(x,y) &= N_1 u_1 + 0\cdot v_1 + N_2 u_2 + 0\cdot v_2 + N_3 u_3 + 0\cdot v_3 \\ v(x,y) &= 0\cdot u_1 + N_1 v_1 + 0\cdot u_2 + N_2 v_2 + 0\cdot u_3 + N_3 v_3 \end{aligned} \quad (1.14)$$

Estas expresiones equivalen a:

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{Bmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 \end{Bmatrix} \cdot \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{Bmatrix} \quad (1.15)$$

Según la cual la matriz de interpolación del elemento $[N_e]$ es:

$$[N_e] = \begin{Bmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 \end{Bmatrix} \quad (1.16)$$

y las submatrices de interpolación son:

$$[N_1] = \begin{Bmatrix} N_1 & 0 \\ 0 & N_1 \end{Bmatrix} \quad [N_2] = \begin{Bmatrix} N_2 & 0 \\ 0 & N_2 \end{Bmatrix} \quad [N_3] = \begin{Bmatrix} N_3 & 0 \\ 0 & N_3 \end{Bmatrix} \quad (1.17)$$

Donde los valores de N_i vienen dados por:

$$N_1 = \frac{x_2 y_3 - y_2 x_3 + x(y_2 - y_3) + y(x_3 - x_2)}{x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)}$$

$$N_2 = \frac{x_3 y_1 - y_3 x_1 + x(y_3 - y_1) + y(x_1 - x_3)}{x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)}$$

$$N_3 = \frac{x_1 y_2 - y_1 x_2 + x(y_1 - y_2) + y(x_2 - x_1)}{x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)}$$

Debido a que los nodos del elemento están numerados en sentido contrario a las agujas del reloj, el denominador resulta ser positivo.

Las funciones de interpolación N_i tienen valor unidad en el nodo *i* y valor cero en los restantes nodos y, por ello, son polinomios lagrangianos y, en consecuencia, el elemento triangular de tres nodos es un *elemento lagrangiano*. También se verifica que en cualquier punto del elemento. Al igual que en el caso de los elementos unidimensionales, la función N_i determina los desplazamientos de los puntos del elemento cuando se le da un valor unidad al desplazamiento del nodo *i* manteniendo nulos los desplazamientos de los otros nodos (ilustración 2). También en este caso, la representación de la función de interpolación N_i coincide con la forma que adquiere el elemento al disponer los anteriores desplazamientos en dirección perpendicular al elemento. Por esta razón, la función N_i se le llama también *función de forma* y a la matriz $[N_e]$ *matriz de forma.*

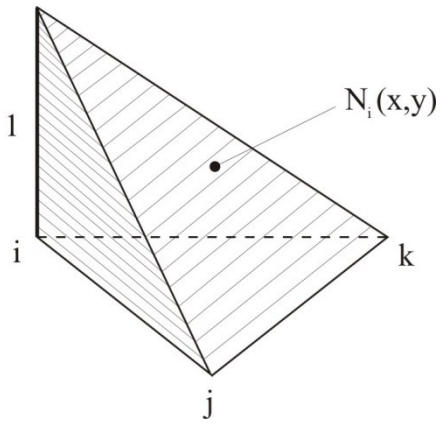


Ilustración 2: Función N_i

Las funciones de desplazamiento (1.1) cumplen la condición de isotropía por ser las funciones de desplazamientos polinomios cuyos términos son simétricos respecto a las coordenadas x e y completando la segunda fila del triángulo de Pascal. Además al ser polinomios, son funciones uniformes y continuas y, por ello, cualquier punto del interior del elemento satisface la condición de compatibilidad. Y también satisfacen la condición de compatibilidad interelemental (ilustración 3), ya que si los desplazamientos δ_i, δ_j de los nodos comunes *i, j* son iguales en los dos elementos, como los desplazamientos de los puntos del borde varían linealmente, no se puede producir ningún salto o discontinuidad en los desplazamientos de dichos puntos.

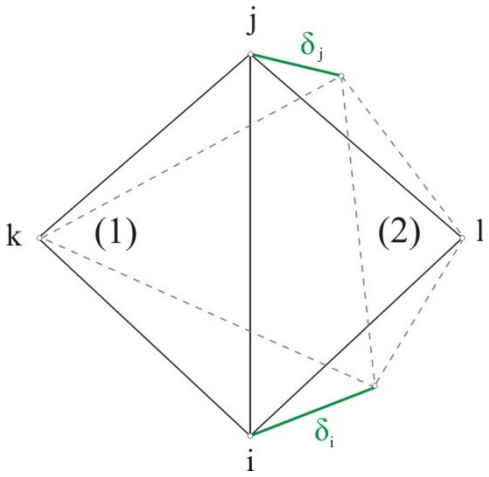


Ilustración 3: Compatibilidad interelemental

Para la construcción de la **matriz de rigidez del elemento** tenemos que realizar el siguiente cálculo:

$$[K_e]=\int_{Ve}[B_e]^T*[D]*[B_e]dV_e \quad (1.18)$$

Para ello tenemos que calcular los siguientes elementos:

- El estado de **deformación unitaria** viene definido por $\mathbf{B_e}$ que en este caso vale:

$$[B_e] = [\partial]\,[N_e]$$

De forma matricial tenemos que:

$$[B_e]=[\partial][N_e]=\begin{pmatrix} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial y} & 0 & \frac{\partial N_3}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} \end{pmatrix} \quad (1.19)$$

Esta matriz también se puede poner de la siguiente forma:

$$B = [\,B_1 \quad B_2 \quad B_3\,]$$

Si derivamos las funciones de forma respecto a cada una de las variables, podemos observar que la matriz B, es una matriz constante, y no depende de x e y, por lo tanto las deformaciones unitarias ε son constantes en todo el elemento, y también lo serán las tensiones, que son proporcionales a ellas.

A pesar de haber definido de este modo la matriz B, ha sido programada manteniendo la siguiente estructura:

$$[B_e]=[\partial][N_e]=\begin{pmatrix} \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial x} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\partial N_1}{\partial y} & \frac{\partial N_2}{\partial y} & \frac{\partial N_3}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} \end{pmatrix}$$

- La **matriz constitutiva elástica [D],** con las propiedades elásticas (mecánicas) del material, depende del tipo de problema:

  - Tensión plana

$$[D]=\frac{E}{1-v^2}*\begin{pmatrix} 1 & v & 0 \\ v & 1 & 0 \\ 0 & 0 & \frac{1-v}{2} \end{pmatrix} \quad (1.20)$$

  - Deformación plana

$$[D]=\frac{E\cdot(1-v)}{(1+v)\cdot(1-2\nu)}*\begin{pmatrix} 1 & \frac{v}{1-v} & 0 \\ \frac{v}{1-v} & 1 & 0 \\ 0 & 0 & \frac{1-2\nu}{2(1-v)} \end{pmatrix} \quad (1.21)$$

Donde E es el módulo de elasticidad longitudinal o Módulo de Young, y ν es el coeficiente de Poisson.

Con las matrices definidas hasta el momento, ya es posible determinar las matrices de rigidez de los elementos a calcular y siendo *t* el espesor del elemento, se obtiene la matriz de rigidez del elemento triangular de 3 nodos:

$$[K_e]=\int_{Ae}[B_e]^T*[D]*[B_e]*t*dV_e \quad (1.22)$$

de esta ecuación se deducen las submatrices de rigidez del elemento:

$$[K_{ij}^{(e)}]=\int_{Ae}[B_i]^T*[D]*[B_j]*t*dV_e \quad (1.23)$$

Es importante recalcar que la matriz [D] es constante en todo el elemento para el caso de elasticidad lineal y depende del tipo de material, por el contrario, la matriz con las derivadas de las funciones de forma [B] puede ser dependiente de las coordenadas (x,y) en función de la solución elegida para cada elemento. En estos casos, la matriz de rigidez se puede obtener integrando analíticamente si las expresiones son sencillas, o mediante las técnicas de integración numérica en caso contrario.

Una de las ventajas del elemento triangular de 3 nodos, es que la matriz [B] es constante en todo el elemento, es decir, que las deformaciones son constantes. Por tanto la expresión (1.23) se puede simplificar considerablemente porque el producto de matrices, al ser constante, puede salir fuera de la integral con lo que se llega a la siguiente expresión:

$$[K_{ij}^{(e)}]=[B_e]^T*[D]*[B_e]*t*A_e$$

donde A_e es el área del elemento finito.

Una vez que tenemos la matriz de rigidez de cada uno de los elementos en los que se discretiza la estructura, se procederá al ensamblaje de dichas matrices con el fin de obtener la matriz completa de rigidez de la estructura. Para ello se deberá ampliar cada una de las matrices de rigidez de cada uno de los elementos al tamaño de la estructura (esto se consigue usando submatrices nulas como submatrices complementarias). La suma de cada una de estas matrices ampliadas daría lugar a la matriz de rigidez completa de la estructura [K]:

$$[K]=\sum [K_e^0]$$

Una vez tenemos la matriz de rigidez completa de la estructura [K] estamos en condiciones para calcular las fuerzas aplicadas en cada uno de los nodos (ilustración 4)

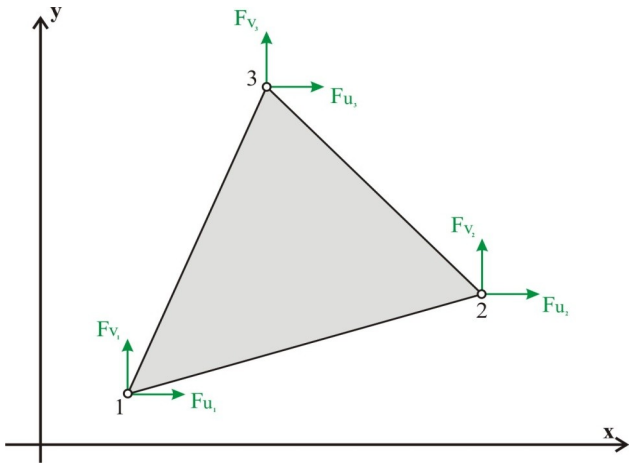


Ilustración 4: Fuerzas nodales del elemento triangular de 3 nodos

Todo sistema de fuerzas al que esté sometido el elemento objeto de estudio deberá ser reducido a cargas en los nodos.

3.- INTRODUCCIÓN

3.1.- SOFTWARE COMPERCIAL

El mercado está repleto de programas diseñados y desarrollados para la resolución de problemas estructurales, algunos de ellos como 'Cosmos' o 'Ansys' , tienen una característica común: no muestran cómo obtienen los resultados finales ni los resultados intermedios necesarios para la correcta resolución de un problema.

Estos programas disponen de una herramienta de representación muy potente, mostrando como solución el campo de tensión y deformación del elemento que se esté estudiando. Se trata de programas profesionales y requieren una licencia. A pesar de ello, debemos confiar plenamente en la solución que dan los programas ya que no permiten analizar los cálculos que se ocultan tras el programa. Por esta razón desarrollamos un software específico que permita estudiar y analizar los cálculos que son necesarios para obtener un campo de tensiones y/o deformaciones.

3.2.- MEFI

MEFI ha sido desarrollado por el Departamento de Estructuras y Construcción de la Universidad Politécnica de Cartagena para la realización de las prácticas de la asignatura 'El Método de los Elementos Finitos en Ingeniería' correspondiente a cuarto curso de Ingeniería Industrial.

El programa realiza el análisis estático (elástico-lineal), por el MEF, de problemas de elasticidad y problemas de campos en régimen permanente, y mediante análisis matricial, de estructuras planas articuladas o rígidas.

El principal objetivo ha sido conseguir un programa sencillo, que permita a los estudiantes del MEF una comprobación rápida de los resultados obtenidos con la aplicación del MEF. Las opciones se han reducido al mínimo indispensable, con objeto de que el programa sea fácil de usar, tenga un tiempo de aprendizaje mínimo, y sirva de ayuda para la comprensión del MEF, a pesar de ello es necesario aprender como introducir datos en el programa ya que responden a una estructura concreta que es necesaria conocer con ayuda de un manual.

El programa MEFI emplea un modelo definido mediante la geometría (puntos, líneas, áreas y volúmenes), los materiales, las propiedades, los elementos, los desplazamientos impuestos y las cargas. Para facilitar la modificación de los datos pueden utilizarse parámetros y expresiones matemáticas, siempre dentro de los parámetros de entrada impuestos por el programa.

Se trata de un programa de libre distribución de eficiencia comprobada, pero al igual que los programas profesionales, muestra como salida un campo de tensiones y deformaciones pero no indica ni facilita la forma en la que se ha llegado a dichos valores. Con el fin de conocer éstos, se ha programado el software MatM con base Matlab que resuelve este inconveniente.

3.3.- MatM

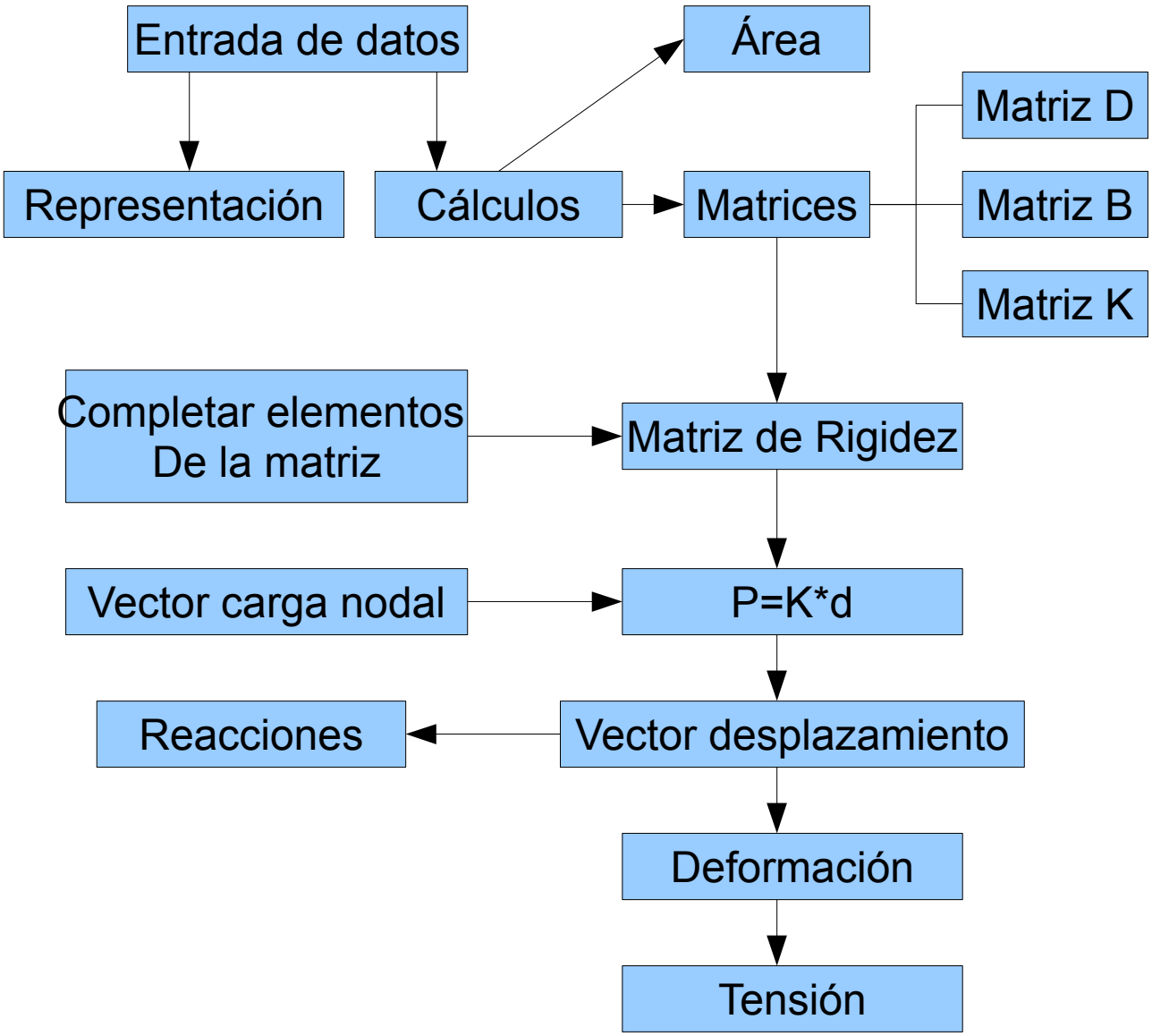
Este software ha sido desarrollado con la premisa de conocer y no perder de vista todos los cálculos necesarios para resolver un problema mediante el método de los elementos finitos (MEF). Se trata de una herramienta didáctica y como tal, es necesario conocer que pasos sigue el método. Presenta un entorno amigable y de fácil acceso.

Esta primera versión tan sólo resuelve problemas de tensión plana con un bajo número de elementos triangulares. Pero no presenta los inconvenientes de los demás programas. Se trata de un programa muy visual, es decir, la zona en la que se muestran las soluciones intermedias va precedida de otra en la que se introducen los datos, al finalizar dicha introducción, hay un botón para comenzar a operar y ver todas las matrices y cálculos intermedios necesarios para el correcto uso del MEF.

Durante su ejecución, muestra una pequeña ventana en la que ver la forma de la estructura plana que se analiza y se genera un matriz global, formada por tantas filas como elementos tenga la estructura en los que se reflejan todos los datos introducidos, así como, los valores numéricos y 'sub matrices' que tienen relación con dicho elemento, de manera que pueda ser exportable a otros soportes informáticos.

Su uso es muy sencillo y queda explicado en los siguientes puntos.

4.- DIAGRAMA DE FLUJO Y BLOQUES



En este diagrama apreciamos el orden en el que se suceden los cálculos del MatM.

En primer lugar hay una entrada de datos que definen la estructura, con ellos calculamos matrices características de cada uno de ellos, para más adelante, generar la matriz global de la estructura solicitando al usuario, que indique cómo ha de hacerse.

Una vez formada esta matriz, se ha de identificar que elementos de la misma permiten resolver la ecuación:

$$P = K * d$$

Es decir, cuales de las distintas ecuaciones que forman este sistema son resolubles, esto es algo que el usuario debe saber e introducirlo en el programa en forma de vector (se muestra como hacerlo en el siguiente apartado).

Una vez obtenido el vector de desplazamiento, se pueden calcular las reacciones, el campo de deformaciones y el de tensiones del elemento. La forma que tienen los vectores solución del programa, así como aquellos que hay que introducir, está indicado en el mismo programa. Todos los vectores que se introduzcan han de introducirse como vectores fila, lo cual, se hace en Matlab dejando un espacio en blanco entre cada elemento del vector.

5.- SOFTWARE MEFI VS MATM: APLICACIÓN

A modo de ejemplo realizaremos un ejercicio sencillo en el que poder ver las propiedades del programa creado y su uso, así como los inconvenientes que presentan los programas profesionales o de distribución libre como el MEFI, comentados anteriormente.

5.1.- MEFI

Los datos del caso son los siguientes:

Nodos	X(m)	Y(m)
1	0	0
2	0,4	0
3	0,4	0,2
4	0	0,4

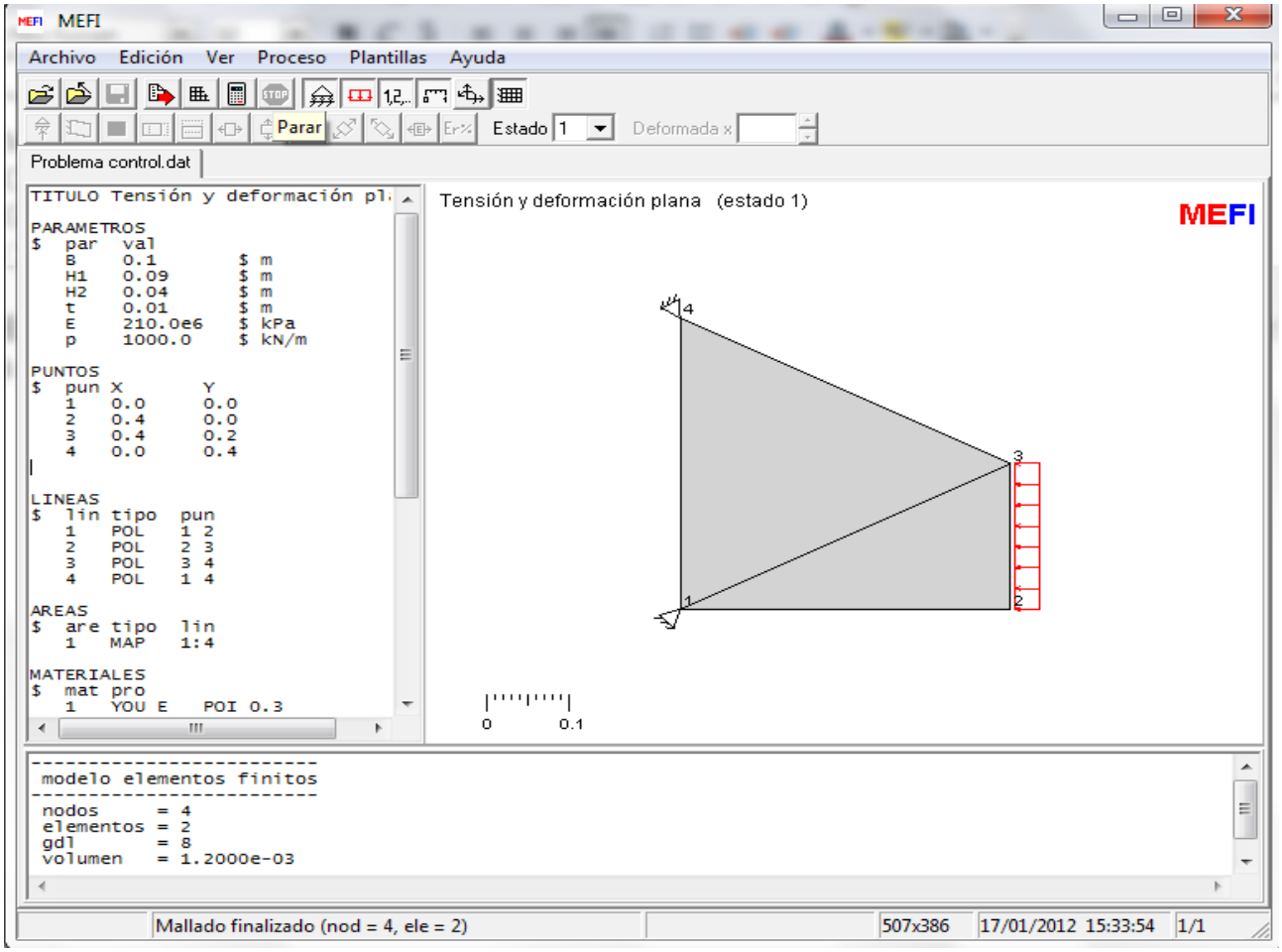
Módulo de elasticidad, E= 210 GPa = $2.1 \cdot 10^5$ N/mm^2 // 1Pa = 1N/m^2

Coeficiente de Poisson, v = 0.3

Grosor de la placa, t = 0.01 m

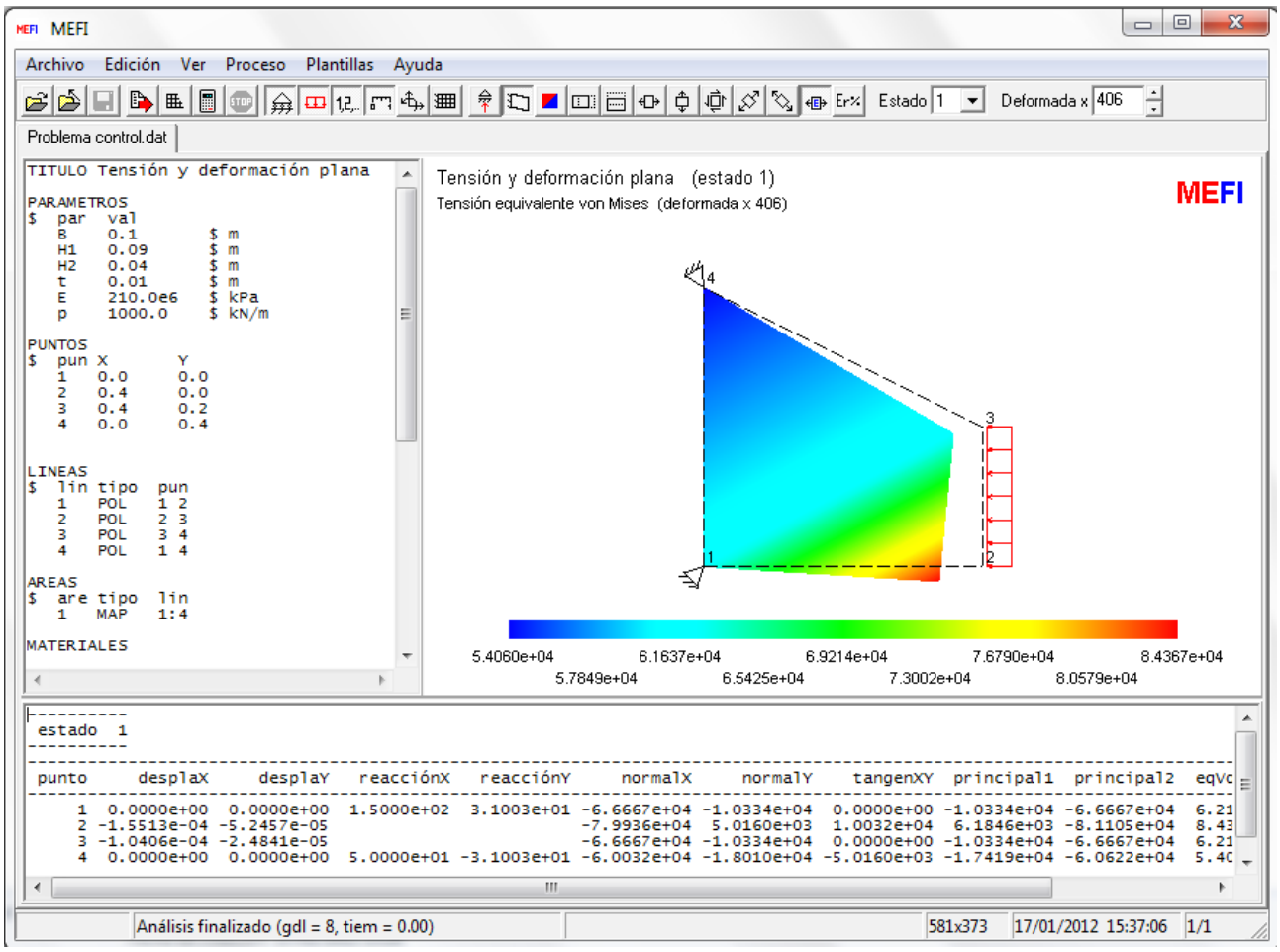
Carga distribuida, P = 1000 kN/m = 10^6 N/m

Quedando el programa de la siguiente forma:



En la parte izquierda de la venta se introducen todos los parámetros, los cuales son interpretados y representados en la parte derecha. La forma en que éstos han de ser introducidos es algo compleja y requiere analizar con detenimiento el manual y conocer las distintas instrucciones para definir correctamente la estructura.

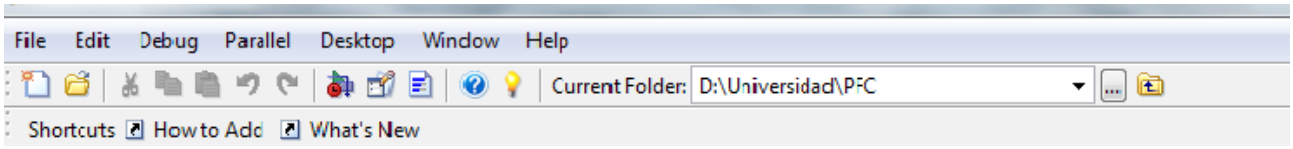
Una vez introducidos todos los datos, tan solo hay que pulsar en el botón 'analizar' y el programa calcula de forma automática el campo de tensiones y deformación sin mostrar cómo llega a obtenerlos.

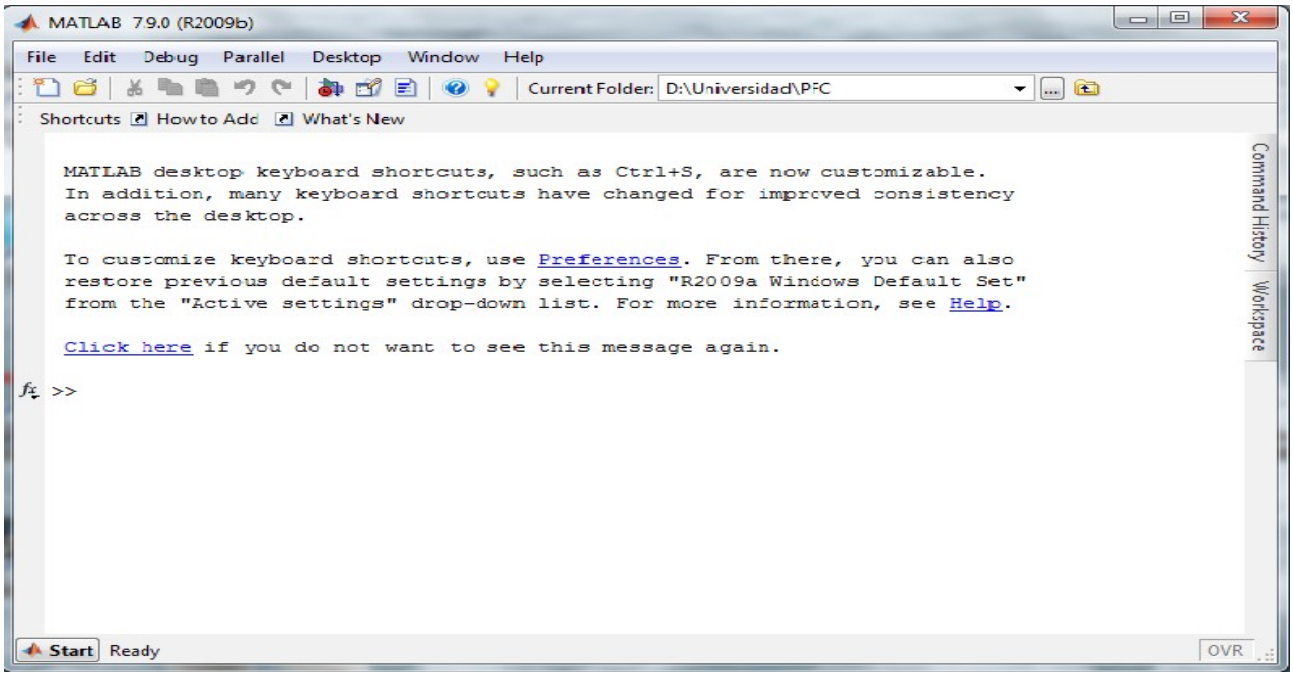


Como se comentó con anterioridad, podemos ver en la parte inferior de la ventana, los valores obtenidos para los distintos puntos, como son el desplazamiento en X, el desplazamiento en Y, valores de reacción, tensiones principales, etc. La pregunta que nos planteamos ahora es: ¿cómo se han obtenido dichos valores? Y es, en respuesta a esta pregunta, la razón principal para la realización de este proyecto.

5.2.- MatM

Lo primero que debemos hacer es saber bien en que carpeta guardamos los archivos con extensión .fig y .m pertenecientes a Matlab en los que se ha realizado el programa, lo que se denomina “Current Folder”. Una vez ubicados todos en la misma carpeta, ejecutaremos Matlab.





El software está desarrollado de manera que se puedan extraer datos, y guardarlos en un archivo con cualquiera de la extensiones que Matlab puede transformar, tales como excel o txt. Para ello hemos de volcar los datos de la GUI en el Workspace de Matlab. Eso lo conseguimos llamando a la variable global en la que están metidos en forma de fila, todos los parámetros de cada uno de los elementos.

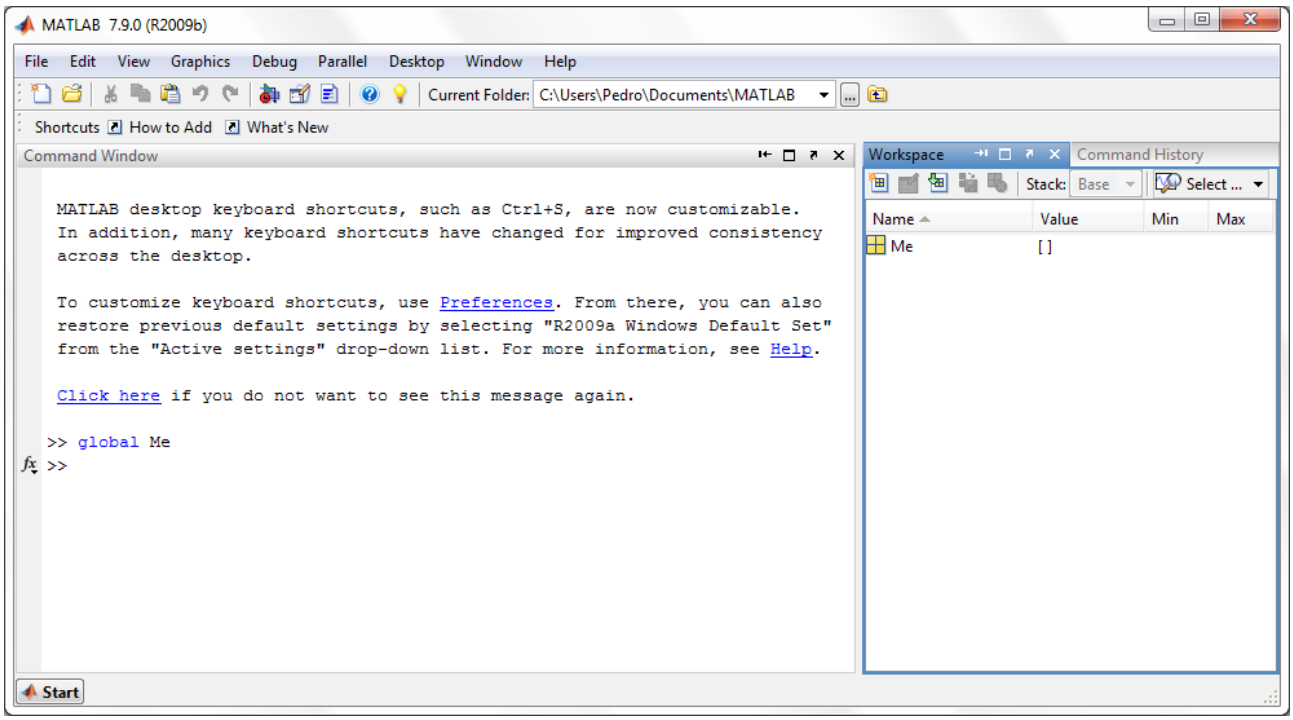
Estos parámetros son:

- Coordenadas de los nodos.
- Módulo elástico
- Coeficiente de Poisson.
- Espesor.
- Área del elemento triangular.
- Matriz constitutiva. D.
- Matriz de deformación. B.
- Matriz de rigidez de cada elemento K.

Para realizar el volcado, antes de ejecutar el programa debemos escribir la siguiente instrucción:

```
>> global Me
```

Automáticamente se nos genera una matriz (Me) que iremos modificando con ayuda del MatM y cuyo seguimiento podemos hacer en el Workspace.

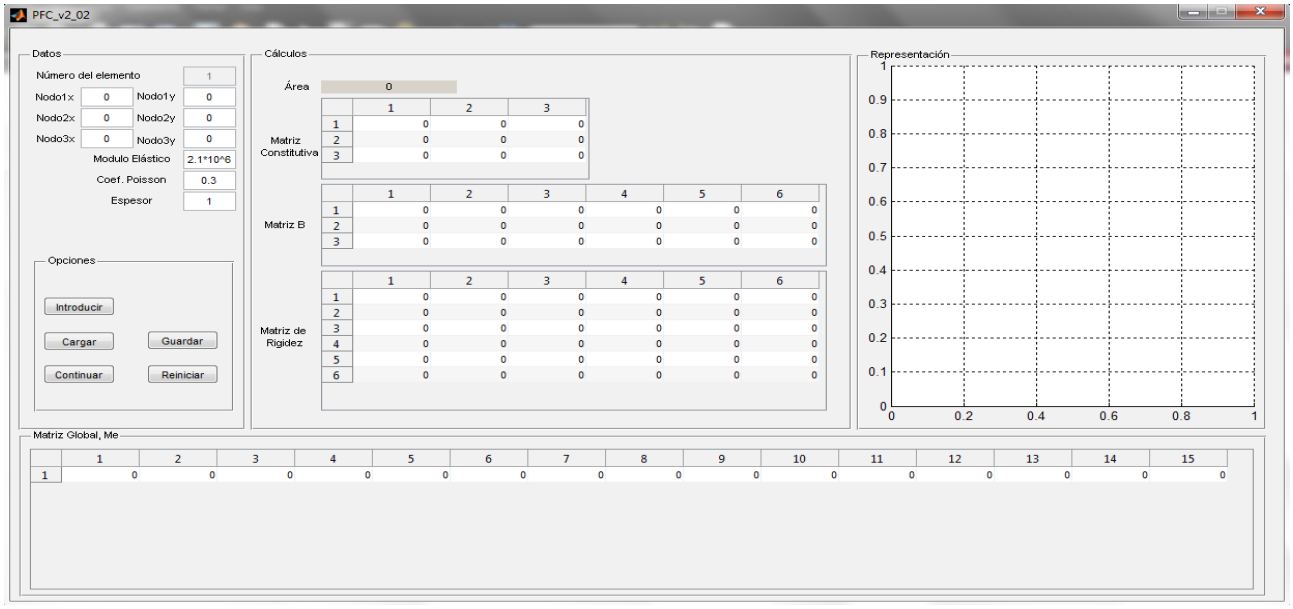


El siguiente paso es llamar a MatM con la siguiente instrucción:

>>MatM

Y se nos abrirá la ventana de presentación del mismo.

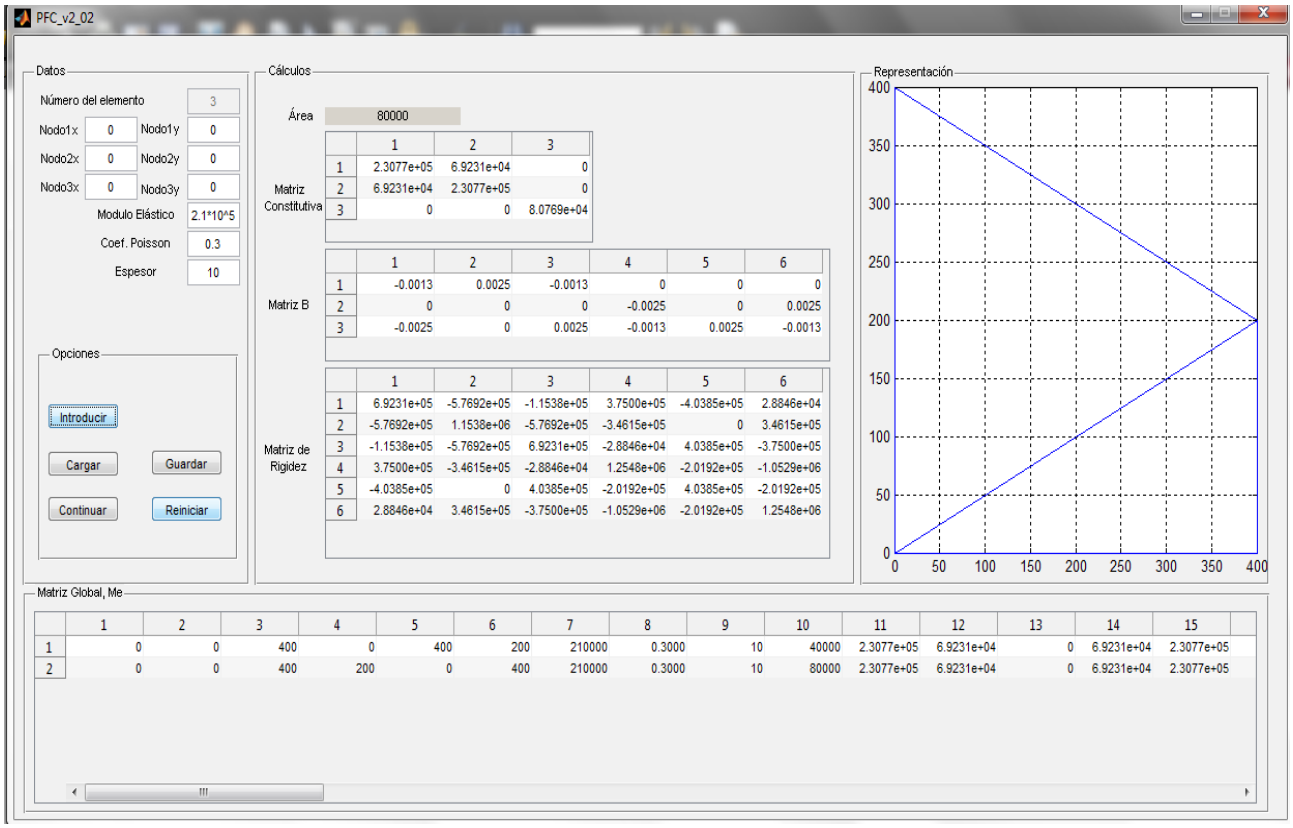
Si le damos al botón continuar se nos abrirá la segunda ventana del mismo.



En la parte superior izquierda se introducen los datos característicos de cada elemento, como es la posición de los nodos, el módulo elástico (que tiene un valor prefijado en Kg/cm^2) el coeficiente de Poisson y el espesor, todos ellos modificables salvo el número del elemento, el cual se incrementa una unidad cada vez que pulsamos el botón introducir.

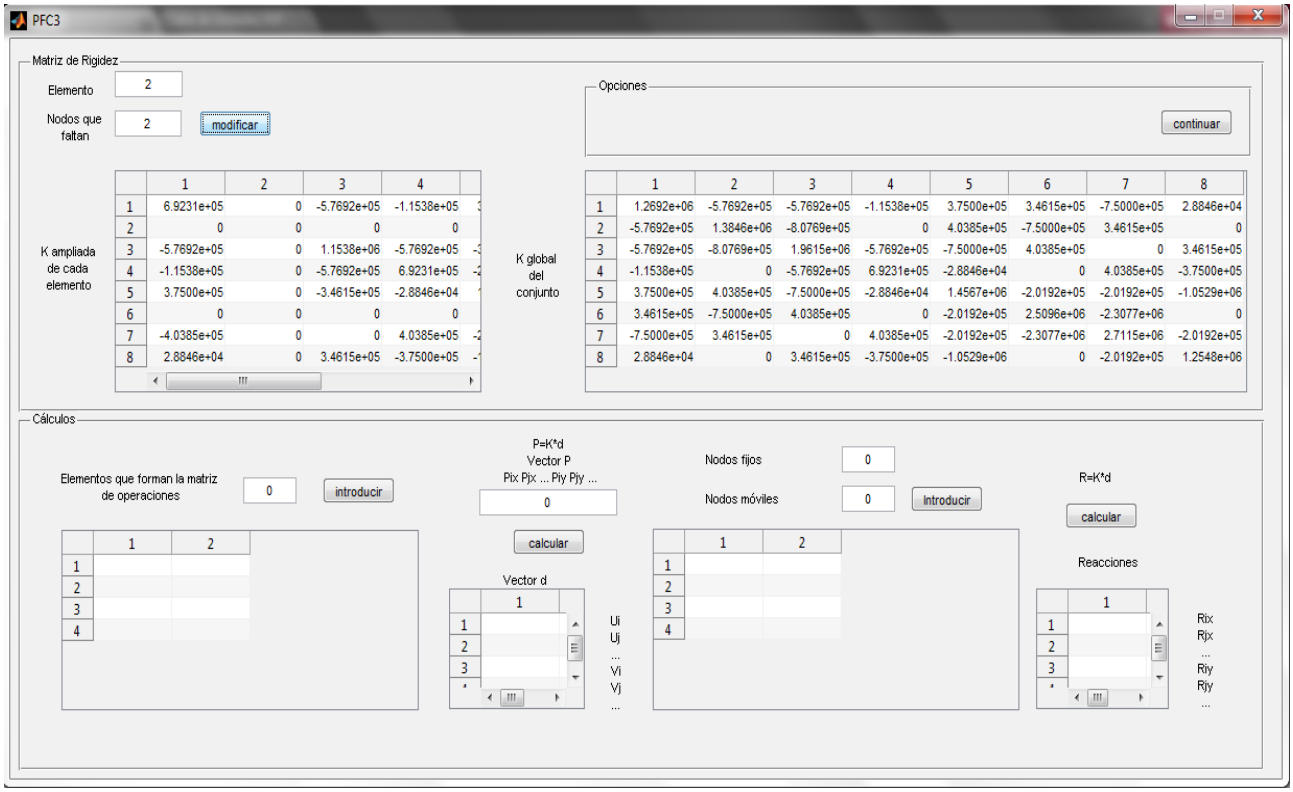
Este software presenta una serie de limitaciones, tales como una vez introducido una serie de valores, si ha sido erróneo, debemos reiniciarlo y comenzar de cero.

El botón "Introducir" calcula el área, la matriz constitutiva o matriz D, la matriz B y la matriz de rigidez de cada elemento triangular, y estos a su vez se almacenan en la matriz Me que aparece en la parte inferior. También dibuja los elementos que se van introduciendo para hacernos una idea de la forma del elemento plano.



Los valores nodales introducidos han sido en milímetros, así como el espesor, y el módulo elástico que tiene un valor de $2.1*10^5$ N/mm^2 . Los valores del campo *cálculo* se corresponden al del último elemento introducido en este caso el elemento dos, y se corresponde con la segunda fila de valores de la matriz *Me*.

En este punto, en el que ya hemos introducido los dos elementos y la matriz global Me está generada, podemos continuar con el programa, dejaremos para el final el exportar esta matriz a un archivo de excel.



Este módulo se subdivide en dos partes, la primera, que se muestra en la imagen de arriba, se corresponde con la tarea de completar la matriz de rigidez de cada elemento con los nodos que no pertenecen al mismo. En la figura se muestra como indicamos que trabajamos con el elemento dos, y el nodo de la estructura que no pertenece a este elemento es el dos también, y por ello completamos con ceros las filas y columnas de la matriz que están relacionadas con el nodo dos de la estructura, es lo que podemos ver en la primera matriz, llamada “K ampliada de cada elemento”.

De forma automática se va generando la K global del conjunto, en la que cada vez que introducimos un elemento modificado, se suma a los anteriores.

La segunda parte del programa necesita que el usuario sepa perfectamente como utilizar la ecuación matricial (P)=[K]*(d) ya que ésta sólo tiene sentido cuando en aquellas operaciones, en las que al multiplicar la matriz K por el vector d nos da un valor coherente de P. Estando éste último formado por la suma del fuerzas y reacciones en los nodos. El vector P tiene la siguiente forma.

$$P_e = R_e + F = \begin{Bmatrix} Rx_1 \\ -10^5 \\ -10^5 \\ Rx_4 \\ Ry_1 \\ 0 \\ 0 \\ Ry_4 \end{Bmatrix}$$ Está relacionado con este vector de desplazamiento. $$\begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{Bmatrix}$$

El valor de -10^5 N ha sido obtenido teniendo en cuenta que la carga distribuida aplicada en el sentido negativo del eje X está aplicada sobre dos nodos, el 2 y el 3. Realizando la siguiente operación:

$$Fx_2 = Fx_3 = \frac{1}{2} \cdot (-1000) \frac{N}{mm^2} \cdot 10\text{mm} \cdot 200\text{mm} = -10^5\ N$$

Siendo 10mm el espesor del elemento, y 200mm la longitud del lado 2-3 en el que está aplicada la carga.

En el campo "Elementos que forman la matriz de operaciones" debemos introducir el valor de los nodos cuyos desplazamientos están permitidos y por tanto, la ecuación P=K*d tiene sentido. Como se trata de dos nodos, debemos introducirlos como un vector fila, esto se hace escribiendo simplemente los números separados por un espacio en blanco. Una vez introducido, podemos proceder con la introducción del vector carga, que sólo tiene componente en u (x) y sólo se ven implicados los nodos 2 y 3.

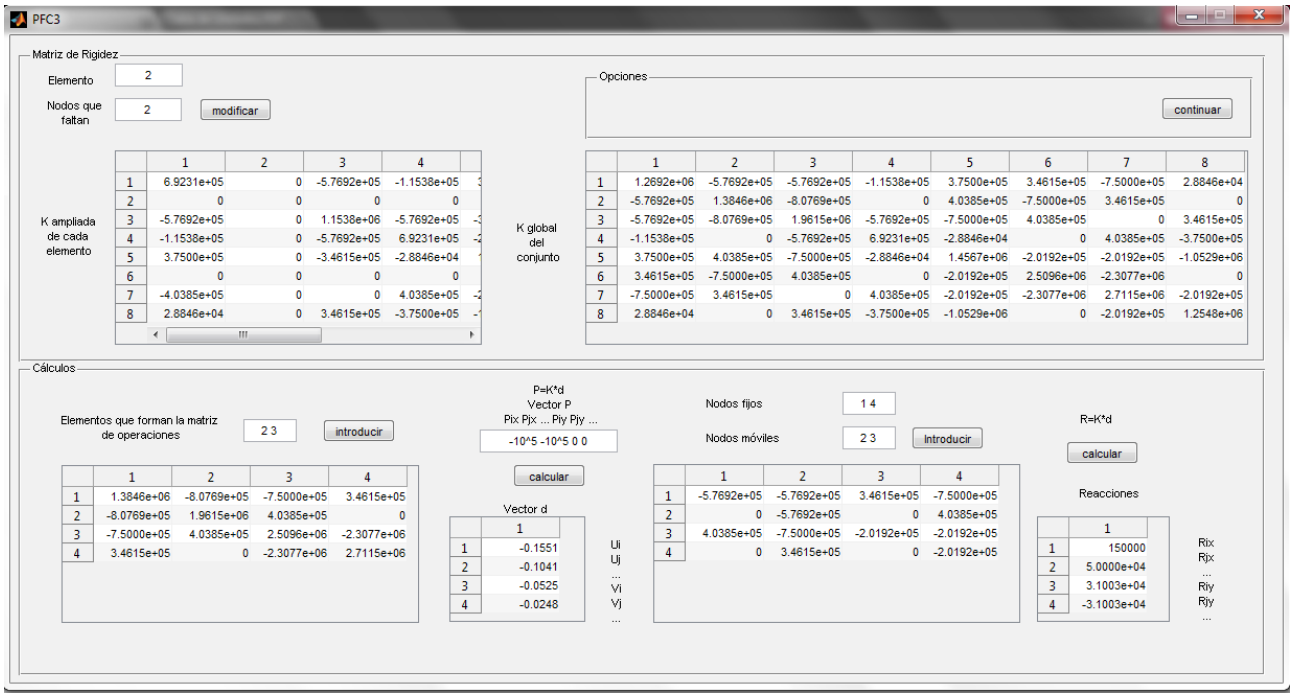
Respetando la nomenclatura con la que se ha realizado el programa, el vector carga es el siguiente:

$$F = \{-10^5 \; -10^5 \; 0 \; 0\} \quad \text{N}$$

El vector desplazamiento obtenido es el siguiente:

$$d=\begin{Bmatrix}-0.1551\\-0.1041\\-0.0525\\-0.0248\end{Bmatrix}mm$$ Que se corresponde con un vector d tal que $$d=\begin{Bmatrix}u_2\\u_3\\v_2\\v_3\end{Bmatrix}$$

Estos valores pueden ser comparados con los que facilita el MEFI, y vemos que son iguales, es decir, sólo se mueven los nodos 2 y 3, mientras que el 1 y 4 que son donde están las reacciones, permanecen fijos. Ahora para calcular el valor de las reacciones sólo debemos continuar rellenando los dos campos que faltan "nodos fijos" y "nodos móviles", introducir para obtener el valor de la matriz de cálculo, y finalmente pulsar el botón calcular para obtener el valor de las reacciones.



Los valores de las reacciones obtenidos se corresponden con R_{1x}, R_{4x}, R_{1y} y R_{4y}. El vector de reacciones obtenido es el siguiente:

$$R=\begin{Bmatrix}150000\\5.0000e{+}04\\3.1003e{+}04\\-3.1003e{+}04\end{Bmatrix}N$$ Que se corresponde con un vector R tal que $$R=\begin{Bmatrix}R_{1x}\\R_{4x}\\R_{1y}\\R_{4y}\end{Bmatrix}$$

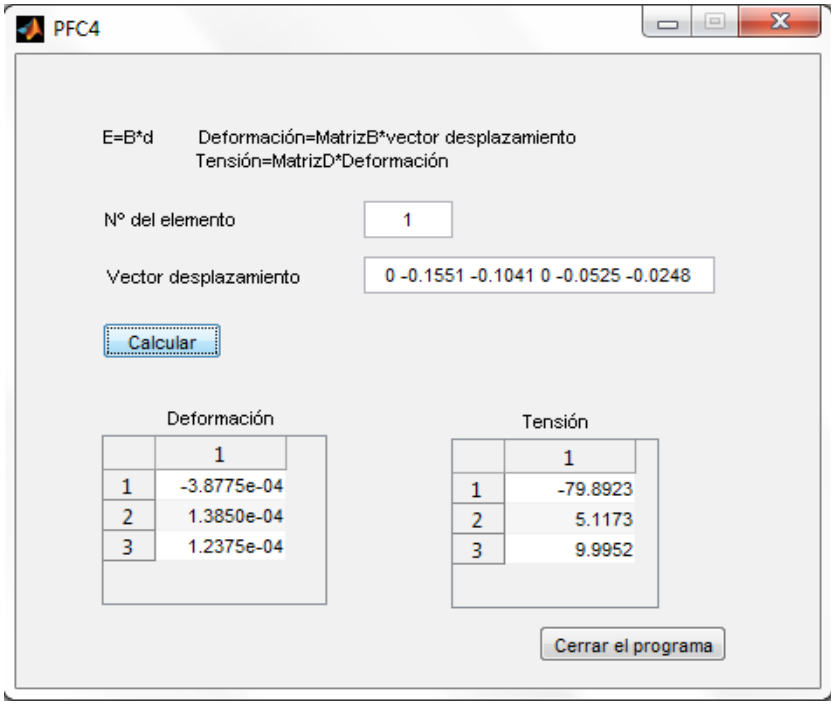
El vector desplazamiento de toda la estructura, así como el vector de cargas y reacciones nodales será:

$$d = \begin{Bmatrix} 0 \\ -0.1551 \\ -0.1041 \\ 0 \\ 0 \\ -0.0525 \\ -0.0248 \\ 0 \end{Bmatrix} mm \qquad R = \begin{Bmatrix} 150000 \\ -10^5 \\ -10^5 \\ 5.0000e{+}04 \\ 3.1003e{+}04 \\ 0 \\ 0 \\ -3.1003e{+}04 \end{Bmatrix} N$$

Valores que se corresponden con los obtenidos en el software MEFI.

Finalmente queda calcular el vector tensión y el de deformación unitaria, para ello pulsamos una última vez en el botón continuar y completamos los campos vacíos.

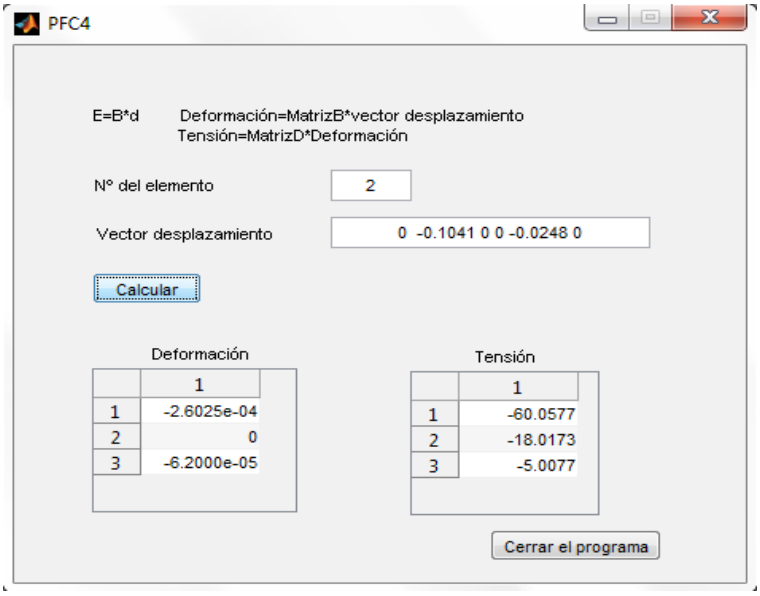
El primero es el número con el que se ha identificado a cada elemento, y el segundo campo se corresponde al desplazamiento, que sufren los nodos que lo forman. De manera que para el primer elemento deberíamos escribir y obtener los siguientes resultados.



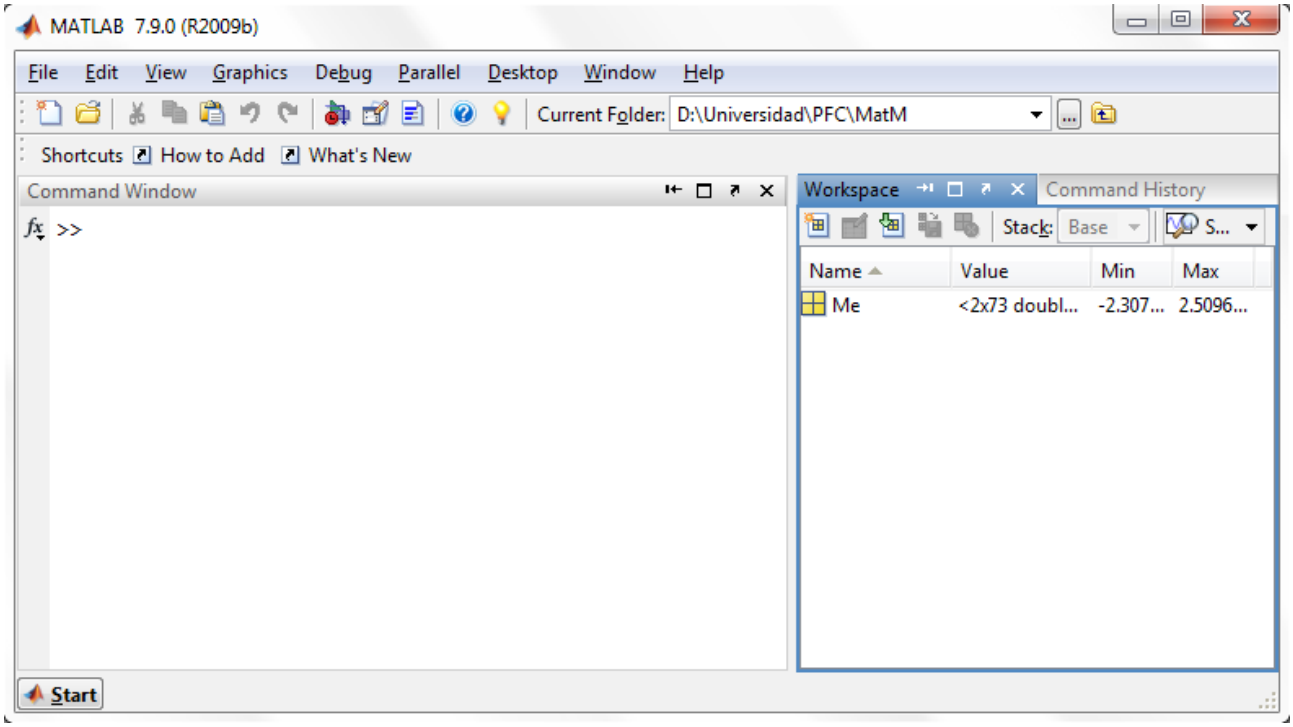
Aquí hemos obtenido los vectores de deformación unitaria y de tensión del primer elemento, y estos tienen la siguiente estructura:

$$\{\varepsilon\}=\begin{Bmatrix}\varepsilon_x\\ \varepsilon_y\\ \gamma_{xy}\end{Bmatrix} \qquad \{\sigma\}=\begin{Bmatrix}\sigma_x\\ \sigma_y\\ \tau_{xy}\end{Bmatrix}$$

De forma análoga calculamos tensión y deformación del segundo elemento. Sólo modificando el campo Nº del elemento por el valor 2 y el vector desplazamiento por el correspondiente a los nodos 1, 3 y 4.



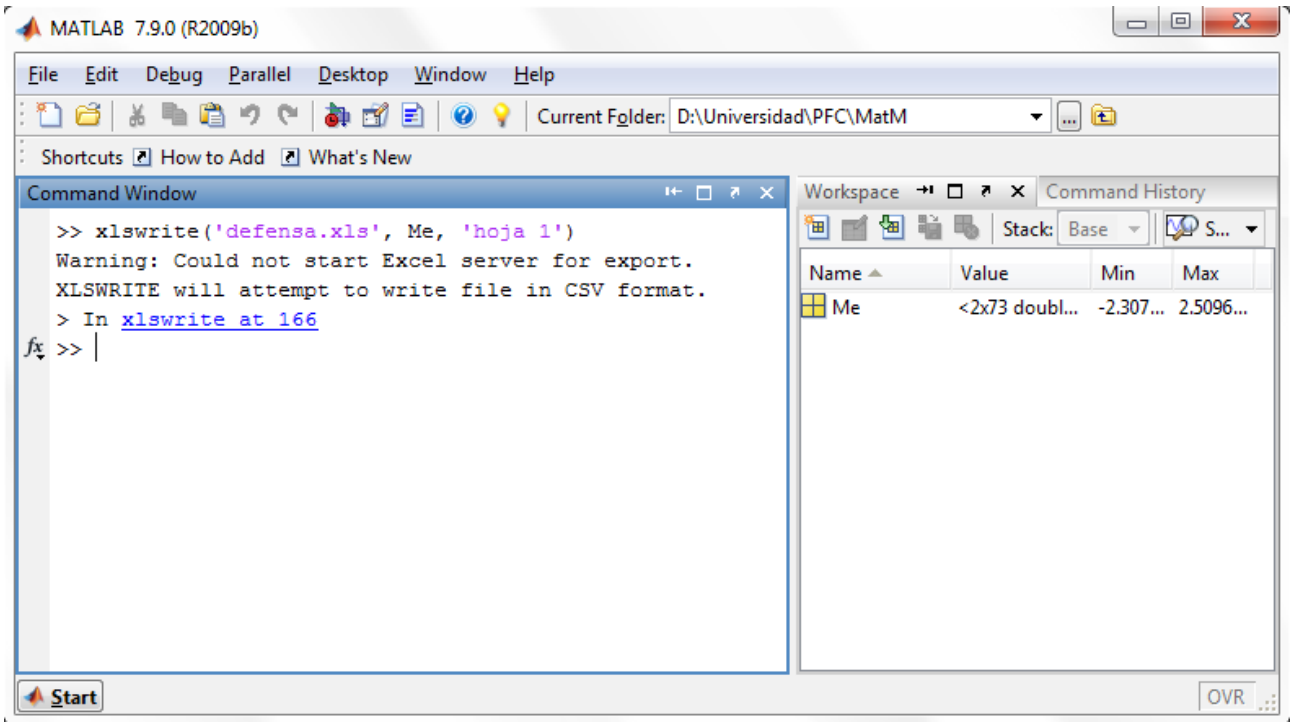
Veamos ahora como exportar nuestros datos a una tabla excel. Antes de iniciar el MatM ejecutamos la sentencia que llamaba a la variable global Me en la que está contenida todos los parámetros calculados en la segunda ventana del programa. Podemos ver como ésta se ha modificado desde su creación.



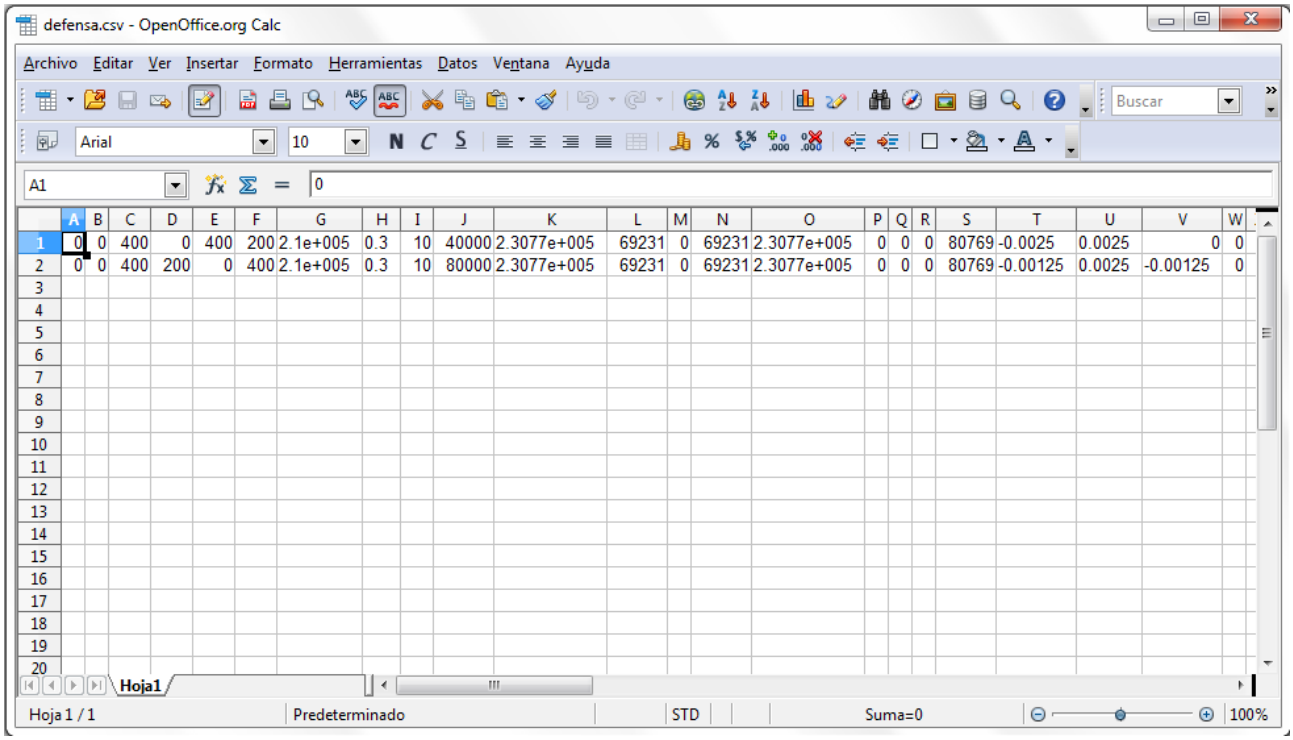
Ya podemos llamar a la variable desde la ventana de comandos, y de este modo, guardarla en un archivo para utilizar en una hoja de cálculo. Para ello bastará con escribir la siguiente instrucción:

```
>> xlswrite('defensa.xls', Me, 'hoja 1')
```

Hemos de tener presentes que el archivo se guarda en el "current folder" de Matlab, por lo que deberemos ir a esa carpeta para copiar el archivo y guardarlo donde deseemos.



El warning que nos lanza Matlab se debe a que no encuentra el software Excel instalado en este equipo, pero el archivo .xls se ha creado con éxito y podemos abrirlo y comprobarlo con OpenOffice.



6.- CONCLUSIONES

Existe gran variedad de programas de cálculo estructural que se basan en el método de los elementos finitos: dados unos parámetros de entrada, que definen un problema, ofrecen una solución al mismo, que a priori, desconocemos como ha sido obtenida.

Este proyecto nace con la intención de poder estudiar y analizar, la evolución numérica y los cálculos intermedios que son necesarios realizar para la resolución de un problema por el método de los elementos finitos para elementos triangulares, y en concreto, problemas de tensión plana.

La mayor dificultad ha residido en la falta de bibliografía e información referente a la programación de una interfaz gráfica en el software Matlab. Cualquier usuario de Matlab puede resolver estos problemas de tensión plana sin necesidad de excesivos quebraderos de cabeza. El problema aparece cuando la mayoría de conocimientos aplicables al 'command windows' no funcionan como cabría esperar. Es por ello por lo que finalmente he optado por usar una variable global extraible y manipulable en la ventana de comandos, para conseguir, de ese modo, exportar los valores obtenidos a otras plataformas.

Por ello, ha sido necesario investigar, indagar y buscar en numerosos foros y blogs, en diversas lenguas como castellano, inglés o francés, e incluso visualizar videos en YouTube, en los que poder obtener orientación en la programación del GUI.

Nos encontramos, que debemos trabajar con tres tipos de variables:

- Variables Locales. Definidas y útiles sólo dentro de cada función.
- Variables Globales dentro del GUI. Variables que pueden ser utilizadas por cualquier función dentro del mismo GUI. Si tenemos una variable llamada 'pedro' para hacerla útil en cualquier función del módulo, debemos hacer la siguiente asignación:
  handles.pedro=pedro;
  Lo cual hace que cuando trabajamos con muchas variables, la longitud de sus nombres sea tedioso de manejar.
- Variables Globales en Matlab. Variables que pueden ser utilizadas por distintos GUIs y llamadas al workspace para su manipulación por aquellos usuarios con un mayor

domino de Matlab. Además son las únicas variables que se mantienen al utilizar varias veces un mismo botón, cómo es el caso del botón 'Introducir' presente en el segundo módulo.

Esto es algo que he aprendido a lo largo de muchas horas de pruebas e investigación. Las GUIs presentan una serie de limitaciones que no he logrado salvaguardar, por ejemplo el tamaño de la ventana del programa, ésta sólo puede ser tan grande, como la pantalla en la que se programe, una vez completado el programa, no admite modificación alguna. Los datos se pueden manipular de diferentes maneras y la definición del tipo de variable es inicialmente muy compleja, y como todo programa, cuando mayor es éste, más difícil y tedioso resulta navegar por el editor del m-file para saber a ciencia cierta qué es lo que se está programando.

He usado los 'Static Text' para etiquetar otros elementos e incluso para mostrar algunos valores numéricos como el área del elemento triangular. Los 'Edit Text' son la forma utilizada para la introducción de parámetros. Inicialmente usé los Static Text para representar las matrices y vectores, pero cuando el tamaño de éstos excedía el del text, los valores se amontonaban y hacían imposible su identificación, por lo que finalmente opté por usar 'Tables', los cuales son ideales para la representación de matrices y vectores, pues mantienen un orden muy claro, este formato de muestra de datos también puede ser utilizado como zona de entrada de los mismos, aunque en este proyecto no se utilicen de ese modo.

La interfaz puede tener otro tipo de funciones, como una ventana para representar una imagen, gráficos o distintos tipos de botones, pero la investigación para determinar el potencial de los mismos se escapa del rango de este proyecto.

Nosotros, nos hemos conformado con presentar un programa, en el que la visualización de las distintas matrices es lo primordial, sin perder nunca de vista el trasfondo teórico que requiere utilizar el método de los elementos finitos. Y tras un arduo esfuerzo obtener un programa, utilizable por cualquier usuario.

Las funciones utilizadas pueden observarse en el anexo, así como todo el código de cada uno de los módulos que componen la interfaz.

7.- LÍNEAS DE DESARROLLO/AMPLIACIÓN

La aplicación MatM es la primera de este tipo realizada en la Escuela Politécnica Superior de Sevilla, es por ello que sólo permite resolver problemas de tensión plana.

Las posibilidades de Matlab son enormes, y una vez familiarizado con el entorno y los comandos de programación de una GUI el programa puede ser ampliado en muchos sentidos.

MatM trabaja con elementos triangulares en tensión plana, pero fácilmente podría crearse un menú en el que poder elegir el estado en el que trabaja, como puede ser el caso de deformación plana, o bien programar las distintas funciones para trabajar con elementos cuadráticos. Se trata de un duro trabajo de programación pero perfectamente realizable.

Los botones de 'Guardar' y 'Cargar' presentes en el módulo dos trabajan de la siguiente manera: 'Guardar' salva una "imagen" del módulo como un archivo con extensión .fig, y el botón 'Cargar' la recupera, el problema de hacer esto es que no se generan las variables globales, por lo que si pulsáramos el botón 'Continuar' no podríamos modificar las matrices de rigidez. La forma actual de salvaguardar esta limitación consiste en introducir un elemento ficticio y no perder de vista la intención con la que se ha creado. Intenté crear un botón para eliminar el último elemento introducido, pero ello implica un conocimiento de la asignación de variables al que aun no he llegado.

El gráfico del módulo dos se genera teniendo en cuenta que una vez introducido un dato, este no se puede eliminar a menos que reiniciemos el módulo, sería interesante generar una función que represente los elementos en función de los valores que tome la variable global *Me*.

La forma en la que se generan las matrices de rigidez también son susceptibles de mejora, cambiando la programación en la que los datos son ordenados para crear la función, sería posible minimizar el código y con ello aumentar la velocidad de cálculo. También es posible generar matrices con las que operar seleccionando directamente los elementos de un 'Table' y volcándolos en otro, un simple copiar y pegar aparentemente funciona, aunque no es una solución elegante. Es posible programar el código de manera que cada vez que seleccionemos un elemento de la matriz este se almacene manteniendo un determinado orden. De manera que cuando generamos la matriz para resolver la ecuación $P = K * d$ podríamos hacer click manualmente en lugar de indicar

cuales son los elementos (numéricamente) que formarán la matriz, ya que cuando tenemos una estructura con un número elevado de elementos triangulares es fácil confundir el orden de los nodos, mientras que si seleccionamos manualmente vemos perfectamente cuales estamos tomando como valores para posteriormente realizar los cálculos pertinentes.

También es posible generar una barra de herramientas en la parte superior del módulo, y programar distintos botones con sus funciones. Es sin lugar a duda, una gran solución al problema de espacio que presentan las GUIs, y que éstas sólo pueden tener como tamaño máximo el de la pantalla en el que hayan sido programadas originalmente, es posible que esto también pueda ser modificado y adaptado a distintas pantallas y resoluciones de las mismas. Con una barra de herramientas podríamos eliminar botones del entorno del programa y dejar solamente zonas de entrada y salida de datos, y colocar todas las operaciones que podemos hacer con ellos fuera.

Otra gran idea sería poder representar el estado tensional y el campo de deformaciones que MatM nos facilita.

8.- BIBLIOGRAFÍA

VÁZQUEZ, M.; LÓPEZ,E.: El método de los elementos finitos aplicado al análisis estructural. de. Noela, Madrid 2001.

MANUAL DE INTERFAZ GRAFICA DE USUARIO EN MATLAB. PARTE I. Autor: Diego Orlando Barragán Guerrero.

Direcciones web:

www.youtube.com/diegokillemall

http://www.youtube.com

http://www.mathworks.es/help/

http://www.blinkdagger.com/matlab/

http://web.usal.es/~gfdc/docencia/GuiSection.pdf

Material facilitado por el profesor y apuntes de clase de la asignatura "Cálculo Avanzado de Estructuras".

Material facilitado por el profesor y apuntes de clase de la asignatura "Métodos Matemáticos".

9.-ANEXO

Códigos de las distintas funciones e interfaces gráficas.

9.1.-Módulo uno. Inicio.

```
function varargout = MatM(varargin)

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @PFC1_OpeningFcn, ...
                   'gui_OutputFcn',  @PFC1_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before PFC1 is made visible.
function PFC1_OpeningFcn(hObject, eventdata, handles, varargin)
A=imread('US','jpg');
A=uint8(A);
Img=image(A,'US',handles.axes1);
set(handles.axes1,'Visible','off','YDir','reverse','XLim',get(Img,'XData'),'YLim
',get(Img,'YData'));
axis off

% Choose default command line output for PFC1
handles.output = hObject;
```

```
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes PFC1 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = PFC1_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;


% --- Executes on button press in pushbutton_continuar.
function pushbutton_continuar_Callback(hObject, eventdata, handles)
close PFC1
PFC_v2_02



% --- Executes on button press in pushbutton_salir.
function pushbutton_salir_Callback(hObject, eventdata, handles)
opc=questdlg('¿Desa salir del programa?','SALIR','Si','No','No');
if strcmp(opc,'No')
    return;
end
clear,clc,close all


% --- Executes during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)
```

9.2.-Módulo dos. Cálculos.

```
function varargout = PFC_v2_02(varargin)

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @PFC_v2_02_OpeningFcn, ...
                   'gui_OutputFcn',  @PFC_v2_02_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before PFC_v2_02 is made visible.
function PFC_v2_02_OpeningFcn(hObject, eventdata, handles, varargin)
grid on;
set(handles.area,'String','0');
set(handles.table_matrizconstitutiva,'data',zeros(3));
set(handles.table_matrizb,'data',zeros(3,6));
set(handles.table_matrizderigidez,'data',zeros(6));
set(handles.uitable2,'data',zeros(1,15));
handles.output = hObject;
guidata(hObject, handles);


% --- Outputs from this function are returned to the command line.
function varargout = PFC_v2_02_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
```

```
function edit_nodo1x_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function edit_nodo1x_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit_nodo1y_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function edit_nodo1y_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton_modificar.
function pushbutton_modificar_Callback(hObject, eventdata, handles)
global Me
i=str2num(get(handles.edit_numerodelelemento,'String'));
ME(1)=str2num(get(handles.edit_nodo1x,'String'));
ME(2)=str2num(get(handles.edit_nodo1y,'String'));
ME(3)=str2num(get(handles.edit_nodo2x,'String'));
ME(4)=str2num(get(handles.edit_nodo2y,'String'));
ME(5)=str2num(get(handles.edit_nodo3x,'String'));
ME(6)=str2num(get(handles.edit_nodo3y,'String'));
ME(7)=str2num(get(handles.edit_moduloelastico,'String'));
ME(8)=str2num(get(handles.edit_coefpoisson,'String'));
ME(9)=str2num(get(handles.edit_espesor,'String'));
% --- Variables más fáciles de escribir en las distintas operaciones
x1=ME(1);
y1=ME(2);
x2=ME(3);
y2=ME(4);
```

```
x3=ME(5);
y3=ME(6);
E=ME(7);
v=ME(8);
t=ME(9);
% ---- Área del elemento triangular----
A=abs(1/2*det([1 x1 y1;1 x2 y2;1 x3 y3]));
ME=[ME,A];
set(handles.area,'String',num2str(A));
% ---- Matriz constitutiva----
D=E/(1-v^2)*[1 v 0;v 1 0;0 0 (1-v)/2];
Dl=[D(1,1:3),D(2,1:3),D(3,1:3)];
ME=[ME,Dl];
set(handles.table_matrizconstitutiva,'data',D);
% ---- Matriz B, o matriz de deformación----
B=(1/(2*A))*[y2-y3 y3-y1 y1-y2 0 0 0;0 0 0 x3-x2 x1-x3 x2-x1; x3-x2 x1-x3 x2-x1
y2-y3 y3-y1 y1-y2];
Dl=[B(1,1:6),B(2,1:6),B(3,1:6)];
ME=[ME,Dl];
set(handles.table_matrizb,'data',B);
% ---- Matriz de rigidez----
K=B'*D*B*A*t;
Dl=[K(1,1:6),K(2,1:6),K(3,1:6),K(4,1:6),K(5,1:6),K(6,1:6)];
ME=[ME,Dl];
set(handles.table_matrizderigidez,'data',K);
% ---- Condición de construcción de la matriz----
if i==1
    Me=ME;
else
    Me=[Me;ME];
end
% ---- Representación gráfica----
x=[x1 x2 x3 x1];
y=[y1 y2 y3 y1];
plot(x,y);hold on;grid on;
set(handles.uitable2,'data',Me);
i=i+1;
set(handles.edit_numerodelelemento,'String',num2str(i));%aumenta el parámetro
del campo número del elemento
set(handles.edit_nodo1x, 'String', '0');
set(handles.edit_nodo1y, 'String', '0');
```

```
set(handles.edit_nodo2x, 'String', '0');
set(handles.edit_nodo2y, 'String', '0');
set(handles.edit_nodo3x, 'String', '0');
set(handles.edit_nodo3y, 'String', '0');
guidata(hObject,handles);


% --- Executes on button press in pushbutton_reiniciar.
function pushbutton_reiniciar_Callback(hObject, eventdata, handles)
opc=questdlg('¿Desa reiniciar el programa?','REINICIAR','Si','No','No');
if strcmp(opc,'No')
    return;
end
set(handles.edit_numerodelelemento, 'String', '1');
set(handles.edit_nodo1x, 'String', '0');
set(handles.edit_nodo1y, 'String', '0');
set(handles.edit_nodo2x, 'String', '0');
set(handles.edit_nodo2y, 'String', '0');
set(handles.edit_nodo3x, 'String', '0');
set(handles.edit_nodo3y, 'String', '0');
set(handles.edit_moduloelastico, 'String', '2.1*10^6');
set(handles.edit_coefpoisson, 'String', '0.3');
set(handles.edit_espesor, 'String', '0');
set(handles.area,'String','0');
set(handles.table_matrizconstitutiva,'data',zeros(3));
set(handles.table_matrizb,'data',zeros(3,6));
set(handles.table_matrizderigidez,'data',zeros(6));
set(handles.uitable2,'data',zeros(1,15));
hold off;
plot(0,0);grid on;
Me=zeros(1);
guidata(hObject,handles);


function edit_nodo2x_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function edit_nodo2x_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end


function edit_nodo2y_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function edit_nodo2y_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit_nodo3x_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function edit_nodo3x_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit_nodo3y_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function edit_nodo3y_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit_moduloelastico_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
```

```
function edit_moduloelastico_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit_coefpoisson_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function edit_coefpoisson_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit_espesor_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function edit_espesor_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function edit_numerodelelemento_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function edit_numerodelelemento_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on key press with focus on text_m_elemento and none of its
controls.
function text_m_elemento_KeyPressFcn(hObject, eventdata, handles)


function text_m_b_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function text_m_b_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit14_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function edit14_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function text_m_k_Callback(hObject, eventdata, handles)
```

```
% --- Executes during object creation, after setting all properties.
function text_m_k_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)


% --- Executes on button press in pushbutton_eliminar.
function pushbutton_eliminar_Callback(hObject, eventdata, handles)


% Guardamos los cálculos obtenidos como una UI de Matlab
function pushbutton_guardar_Callback(hObject, eventdata, handles)
[filename,pathname]=uiputfile('default','Salvar')
if pathname == 0
    return
end
saveDataName=fullfile(pathname,filename);
hgsave(saveDataName);


% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
PFC3


% --- Executes on button press in pushbutton_cargar.
function pushbutton_cargar_Callback(hObject, eventdata, handles)
[filename, pathname] = uigetfile('*.fig', 'Choose the GUI settings file to
load');
loadDataName = fullfile(pathname,filename);
theCurrentGUI = gcf;
hgload(loadDataName);
close(theCurrentGUI);
```

```
% --- Executes during object creation, after setting all properties.
function matrizderigidez_CreateFcn(hObject, eventdata, handles)


function matrizb_ButtonDownFcn(hObject, eventdata, handles)
```

9.3.-Módulo tres. Operaciones.

```
function varargout = PFC3(varargin)


% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @PFC3_OpeningFcn, ...
                   'gui_OutputFcn',  @PFC3_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before PFC3 is made visible.
function PFC3_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);


% --- Outputs from this function are returned to the command line.
function varargout = PFC3_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;


function edit_nodo_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
```

```
function edit_nodo_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit_elemento_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function edit_elemento_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function C=completarbis(K,V)
i=1;
[c,d]=size(V);
while i<=d
    s=V(i);
    [a,b]=size(K);
    K1=K(:,1:s-1);
    K2=K(:,s:b);
    C=[K1 zeros(a,1) K2];
    K1=C(1:s-1,:);
    K2=C(s:a,:);
    C=[K1;zeros(1,b+1);K2];
    [a1,b2]=size(C);
    K1=C(:,1:(s+b/2+1)-1);
    K2=C(:,(s+b/2+1):b2);
    C=[K1 zeros(a1,1) K2];
    K1=C(1:(s+b/2+1)-1,:);
    K2=C((s+b/2+1):a1,:);
    C=[K1;zeros(1,b2+1);K2];
    K=C;
    i=i+1;
end
```

```
% --- Executes on button press in modificar.
function modificar_Callback(hObject, eventdata, handles)
global Me Kg
i=str2num(get(handles.edit_elemento,'String'));
V=str2num(get(handles.edit_nodo,'String'));
L=[Me(i,38:43);Me(i,44:49);Me(i,50:55);Me(i,56:61);Me(i,62:67);Me(i,68:73)];
Z=completarbis(L,V);
if i==1
    Kg=Z;
else
    Kg=Kg+Z;
end
[e,b]=size(Me);
handles.e=(e+2);
set(handles.tabla1,'data',Z);
set(handles.tabla2,'data',Kg);


% --- Executes on button press in pushbutton_introducir.
function pushbutton_introducir_Callback(hObject, eventdata, handles)
global Kg
V=str2num(get(handles.edit5,'String'));
handles.V=V;
a=length(Kg);
i=1;
j=1;
d=length(V);
while i<=d
    while j<=d
       B=Kg(V(i),V(j));
       if j==1
           A=B;
       else
           A=[A;B];
       end
       j=j+1;
    end
    if i==1
        C11=A;
    else
```

```
        C11=[C11,A];
    end
    i=i+1;
    j=1;
end
i=1;
j=1;
Z=V+a/2;
while i<=d
    while j<=d
       B=Kg(Z(i),Z(j));
       if j==1
           A=B;
       else
           A=[A;B];
       end
       j=j+1;
    end
    if i==1
        C22=A;
    else
        C22=[C22,A];
    end
    i=i+1;
    j=1;
end
i=1;
j=1;
while i<=d
    while j<=d
       B=Kg(V(i),Z(j));
       if j==1
           A=B;
       else
           A=[A;B];
       end
       j=j+1;
    end
    if i==1
        C21=A;
    else
```

```
        C21=[C21,A];
    end
    i=i+1;
    j=1;
end
i=1;
j=1;
while i<=d
    while j<=d
       B=Kg(Z(i),V(j));
       if j==1
           A=B;
       else
           A=[A;B];
       end
       j=j+1;
    end
    if i==1
        C12=A;
    else
        C12=[C12,A];
    end
    i=i+1;
    j=1;
end
C=[C11 C12;C21 C22];
set(handles.tabla3,'data',C);
handles.C=C;
guidata(hObject,handles);


% --- Executes on button press in pushbutton_calcular.
function pushbutton_calcular_Callback(hObject, eventdata, handles)
global d
p=str2num(get(handles.edit_carga,'String'));
handles.p=p;
p=p';
d=handles.C\p;
set(handles.tabla4,'data',d);
handles.d=d;
guidata(hObject,handles);
```

```
% --- Executes on button press in pushbutton_introducir2.
function pushbutton_introducir2_Callback(hObject, eventdata, handles)
global Kg
V=str2num(get(handles.edit_nodosreaccion,'String'));
F=str2num(get(handles.edit_nodosdesplazables,'String'));
a=length(Kg);
i=1;
j=1;
e=length(F);
d=length(V);
while i<=d
    while j<=e
        B=Kg(V(i),F(j));
        if j==1
            A=B;
        else
            A=[A B];
        end
        j=j+1;
    end
    if i==1
        C11=A;
    else
        C11=[C11;A];
    end
    i=i+1;
    j=1;
end
i=1;
j=1;
v=V+a/2;
f=F+a/2;
while i<=d
    while j<=e
        B=Kg(V(i),f(j));
        if j==1
            A=B;
        else
            A=[A B];
```

```
        end
        j=j+1;
    end
    if i==1
        C12=A;
    else
        C12=[C12;A];
    end
    i=i+1;
    j=1;
end
i=1;
j=1;
while i<=d
    while j<=e
        B=Kg(v(i),F(j));
        if j==1
            A=B;
        else
            A=[A B];
        end
        j=j+1;
    end
    if i==1
        C21=A;
    else
        C21=[C21;A];
    end
    i=i+1;
    j=1;
end
i=1;
j=1;
while i<=d
    while j<=e
        B=Kg(v(i),f(j));
        if j==1
            A=B;
        else
            A=[A B];
        end
```

```
        j=j+1;
    end
    if i==1
        C22=A;
    else
        C22=[C22;A];
    end
    i=i+1;
    j=1;
end
X=[C11 C12;C21 C22];
set(handles.tabla6,'data',X);
handles.X=X;
guidata(hObject,handles);


% --- Executes on button press in pushbutton_calcular2.
function pushbutton_calcular2_Callback(hObject, eventdata, handles)
R=handles.X*handles.d;
set(handles.tabla5,'data',R);


function edit_totalelementos_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function edit_totalelementos_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit5_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end


function edit_carga_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function edit_carga_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit_nodosreaccion_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function edit_nodosreaccion_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit_nodosdesplazables_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function edit_nodosdesplazables_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton_continuar.
function pushbutton_continuar_Callback(hObject, eventdata, handles)
PFC4
```

```
function edit_nnodos_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function edit_nnodos_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

9.4.-Módulo cuatro. Tensión y deformación.

```
function varargout = PFC4(varargin)

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @PFC4_OpeningFcn, ...
                   'gui_OutputFcn',  @PFC4_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before PFC4 is made visible.
function PFC4_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);


% --- Outputs from this function are returned to the command line.
function varargout = PFC4_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;


function edit_nelemento_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function edit_nelemento_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit_vecdesp_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function edit_vecdesp_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton_calcular1.
function pushbutton_calcular1_Callback(hObject, eventdata, handles)
global Me
i=str2num(get(handles.edit_nelemento,'String'));
f=str2num(get(handles.edit_vecdesp,'String'));
f=f';
L=[Me(i,20:25);Me(i,26:31);Me(i,32:37)];
E=L*f;
set(handles.tabla1,'data',E);
D=[Me(i,11:13);Me(i,14:16);Me(i,17:19)];
O=D*E;
set(handles.tabla2,'data',O);


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
opc=questdlg('¿Desa cerrar del programa?','CERRAR','Si','No','No');
if strcmp(opc,'No')
    return;
end
clear,clc,close all
```