*Universidad de Matanzas "Camilo Cienfuegos".*
*Facultad de Ingenierías Química y Mecánica.*


# LIBRERÍA DE MATLAB PARA LA SOLUCIÓN DE PROBLEMAS DE ELASTICIDAD LINEAL POR EL MÉTODO DE ELEMENTOS FINITOS.

Tesis Presentada en Opción al Título Académico de Máster en Ingeniería Asistida por Computadora.

**Autor: Lic. Omar López Armas.**

**Tutor: Dr.C. Ramón Quiza Sardiñas.**

Matanzas, 2009.

## RESUMEN.

El trabajo aborda la creación de una biblioteca de funciones en MATLAB, que permita la aplicación del método de elementos finitos a la solución de problemas de resistencia de materiales bidimensionales, en la región de deformaciones elástica. Se empleó la formación del método directo de rigidez. En la tesis se explican los algoritmos utilizados así como los códigos implementados, de forma tal que sirva como manual de usuario de la misma. Se ejecutaron estudios de caso para verificar el correcto funcionamiento de la biblioteca, y sus resultados fueron comparados con los obtenidos por otros programas.

## ABSTRACT.

This works describes the development of a toolbox for MATLAB, to allow application of finite elements method in solving two-dimensional strength of materials problems, into the elastic strains region. It was employed the direct stiffness method. In the thesis employed algorithms and implemented codes are explained, in order to be used as user's guide of the toolbox. Some cases were executed to verify the right functioning of the toolbox, and their outcomes were compared with obtained from other software.

## TABLA DE CONTENIDO.

# INTRODUCCIÓN.

El uso de los elementos finitos ha alcanzado una gran difusión dentro de diversas ramas de la ingeniería, ya que permiten resolver problemas de alta complejidad que, de otra forma serían insolubles. Especialmente, dentro de la resistencia de materiales, el método de elementos finitos ha sido aplicado con éxito a un gran grupo de problemas.

La principal dificultad en la aplicación del método de elementos finitos es la necesidad de contar con un programa de computación (software) que realice los complejos y laboriosos cálculos necesarios para resolver un problema (Fernández-Méndez y Huerta, 2004).

En la actualidad, existe una gran variedad de software disponible, pero la mayor parte son programas de propósito específico, que carecen de la flexibilidad necesaria para resolver otros problemas que no coincidan con aquellos para los que fueron concebidos.

También existen programas de código abierto, como Ansys y Algor, que permiten realizar cambios en la formulación, aunque al final siempre tienen limitaciones inherentes a su concepción. Adicionalmente, el uso de estos paquetes está limitado por la licencia propietaria que poseen (Koenraad, 2007).

Una tercera opción es el uso de código para programas de propósito general, como MATLAB o MathCAD (Merlín y Morandini, 2005). Esta variante permite resolver diversos problemas, con una gran

flexibilidad. También permite ser enriquecido con código nuevo. Existe, disponible en Internet, una gran cantidad de código de este tipo, pero o bien, no está documentado, o bien el código es poco legible y, por lo tanto, difícil de utilizar (Whiteley, 2009).

Ante esta situación, se puede identificar como **problema científico**, la no existencia de un código, en Matlab, para la solución de problemas de resistencia de materiales por el método de elementos finitos, legible y documentado, de forma tal que pueda ser utilizado con facilidad y extendido según las necesidades.

Para la solución de este problema, se propone la **hipótesis** de que es posible implementar un conjunto de funciones, en Matlab, que permitan aplicar con facilidad el método de elementos finitos a los problemas de resistencia de materiales.

Por lo tanto, se plantea como **objetivo** de este trabajo, implementar un conjunto de funciones, en Matlab, que permitan aplicar con facilidad el método de elementos finitos a los problemas de resistencia de materiales.

Se trazan como **tareas** necesarias para el cumplimiento de este objetivo, las siguientes:

- Revisión del estado del arte para identificar códigos que puedan ser utilizados.
- Concepción general del programa con la definición de funciones y procedimientos.

- Diseño de los algoritmos de cada subrutina (función o procedimiento).
- Implementación de las subrutinas.
- Realización de estudios de caso para validar la eficacia y eficiencia del código desarrollado.

# CAPÍTULO 1. ESTADO DEL ARTE.

El objetivo de este capítulo es presentar una revisión crítica de la bibliografía científica publicada sobre el método de elementos finitos en problemas de elasticidad y los programas existentes para su solución, de forma tal que sirva se marco teórico-referencial al resto de la investigación.

## 1.1. Definición.

El método de elementos finitos (*finite element method*, FEM) es una técnica numérica que permite determinar soluciones aproximadas de ecuaciones diferenciales en derivadas parciales o ecuaciones integrales. El enfoque de la solución se basa en la eliminación completa de la ecuación diferencial (en problemas estacionarios) o en la transformación de la ecuación diferencial en derivadas parciales en un sistema de ecuaciones diferenciales ordinarias, que son luego integradas numéricamente (Öztorun, 2006).

En la solución de ecuaciones diferenciales en derivadas parciales, el reto principal es crear una ecuación que aproxime a la ecuación estudiada, pero que sea numéricamente estable (Saka, 2005). El método de elementos finitos es una buena opción para lograr esto, en problemas definidos sobre dominios complejos o cuando el dominio cambia (Zienkiewicz y Taylor, 2005).

## 1.2. Breve reseña histórica,

El método de elementos finitos se originó a partir de la necesidad de resolver problemas de análisis elástico y estructural en ingeniería civil y aeronáutica, principalmente. Su desarrollo se remonta a los trabajos de Hrennikoff, en 1941, y Courant, en 1942 (Zienkiewicz y Taylor, 2005). A pesar de las notables diferencias de los trabajos de estos pioneros con los enfoques actuales, ya contenían un elemento esencial: la discretización del dominio en elementos.

El desarrollo de este método, alcanzó un gran desarrollo en los años cincuenta, especialmente en las Universidades de Stuttgart, y Berkeley, destacándose los trabajos a Argyris y Clough. A finales de la década del 50, conceptos básicos como la matriz de rigidez y el ensamble de elementos, estaba ya bien establecidos (Wu y Li, 2006).

La formulación matemática rigurosa del método, fue llevada a cabo en 1973, a través del trabajo de Strang y Fix, convirtiéndose en una rama de las matemáticas aplicadas a la modelación de sistemas físicos en una gran variedad de disciplinas científicas e ingenieriles (Felippa, 2004; Ma *et al.* 2009).

## 1.3. Problema de elasticidad lineal general.

Las ecuaciones que rigen un problema de elasticidad lineal con contorno se basan en tres ecuaciones diferenciales tensoriales en derivadas parciales, que describen el balance de momento lineal y seis relaciones

entre el desplazamiento y las deformaciones infinitesimales. Este sistema de ecuaciones diferenciales se completa con un conjunto de ecuaciones constitutivas algebraicas (Shabana, 2008).

Las relaciones que representan la conservación de momento lineal (conocida como la Segunda Ley de Newton), y se puede escribir, en notación tensorial, de la forma (Duan y Lin, 2005):

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{F} = \rho \ddot{\mathbf{u}} ; \tag{1.1}$$

donde $\boldsymbol{\sigma}$ es el segundo tensor (de segundo orden) de tensiones de Piola-Kirchhoff; $\mathbf{F}$ es la fuerza distribuida por unidad de volumen; $\rho$ es la densidad de masa del material; $\mathbf{u}$ es el vector de desplazamientos; $\nabla\cdot(\bullet)$ es el operador de divergencia; y $(\ddot{\bullet})$ representa la segunda derivada con respecto al tiempo.

Por su parte, la relación entre desplazamientos y tensiones, viene dada por la expresión:

$$\boldsymbol{\varepsilon} = \tfrac{1}{2}[\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathrm{T}}] ; \tag{1.2}$$

en la cual $\boldsymbol{\varepsilon}$ es tensor (de segundo orden) de desplazamientos infinitesimales de Green-Lagrange; $\nabla(\bullet)$ es el operador gradiente; y $(\bullet)^{\mathrm{T}}$ representa la transpuesta.

Finalmente, la ecuación constitutiva, en su forma tensorial, tiene la forma:

$$\boldsymbol{\sigma} = \boldsymbol{\mathcal{C}} : \boldsymbol{\varepsilon} ; \tag{1.3}$$

donde $\boldsymbol{\mathcal{C}}$ es el tensor (de cuarto orden) de elasticidad del material.

En el Anexo 1, se muestra la notación detallada de cada una de las magnitudes involucradas en el problema de elasticidad lineal.

Las relaciones anteriores, se pueden representar gráficamente a través del diagrama de Tonti (Felippa, 2004). En el mismo se indican, con rectángulos, las magnitudes incógnitas y, con óvalos, las magnitudes conocidas.

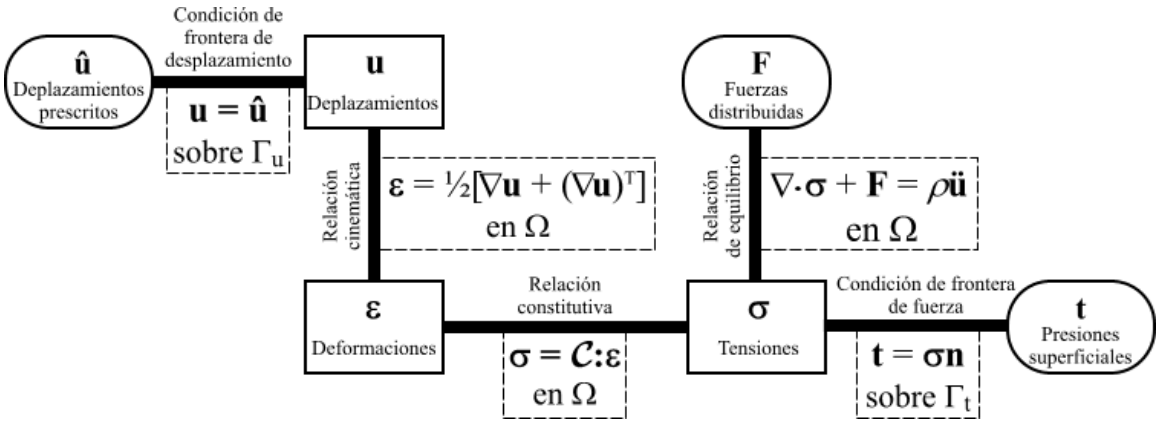



**Figura 1.1.** Diagrama de Tonti para el problema general de elasticidad lineal.

En este diagrama, aparecen también las condiciones de frontera, tanto de desplazamientos:

$$\mathbf{u} = \hat{\mathbf{u}}, \quad \forall \mathbf{x} \in \Gamma_u ; \tag{1.4}$$

como de fuerzas:

$$\mathbf{t} = \boldsymbol{\sigma}\mathbf{n}, \quad \forall \mathbf{x} \in \Gamma_t ; \tag{1.5}$$

en las cuales $\hat{\mathbf{u}}$ es el vector de desplazamientos prescritos sobre el sector de la frontera $\Gamma_u$; $\mathbf{t}$ es el vector de presiones superficiales (provocadas por las fuerzas externas) sobre el segmento de la frontera $\Gamma_t$; y $\mathbf{n}$ es el vector normal a la frontera.

Como es bien conocido, la solución analítica de este problema, para dominios complejos es muy difícil o, incluso, imposible, por lo cual se requiere la aplicación de técnicas de solución numéricas aproximadas, de las cuales la más popular es el método de elementos finitos (Braess, 2007).

Un aspecto importante es que, para un material cualquiera, el tensor de elasticidad tiene sólo 21 términos independientes (Avril y Pierron, 2007), por lo cual la expresión (1.3) puede ser escrita en forma de producto matricial como (Sadd, 2005):

$$[\boldsymbol{\sigma}] = [\mathbf{C}][\boldsymbol{\varepsilon}]; \tag{1.6}$$

en la cual:

$$[\boldsymbol{\sigma}] = \begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \tau_{12} \\ \tau_{23} \\ \tau_{31} \end{bmatrix} \quad [\mathbf{C}] = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} & c_{16} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} & c_{26} \\ c_{31} & c_{32} & c_{33} & c_{34} & c_{35} & c_{36} \\ c_{41} & c_{42} & c_{43} & c_{44} & c_{45} & c_{46} \\ c_{51} & c_{52} & c_{53} & c_{54} & c_{55} & c_{56} \\ c_{61} & c_{62} & c_{63} & c_{64} & c_{65} & c_{66} \end{bmatrix} \quad [\boldsymbol{\varepsilon}] = \begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{33} \\ 2\varepsilon_{12} \\ 2\varepsilon_{23} \\ 2\varepsilon_{31} \end{bmatrix}.$$

No obstante, para el caso particular de un material isotrópico y homogéneo (tal como la mayoría de las aleaciones metálicas), la matriz de los coeficientes de elasticidad se reduce a (Meyers y Chawla, 2008, p. 101):

$$[\mathbf{C}]=\begin{bmatrix} \lambda+2\mu & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda+2\mu & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda+2\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & 2\mu & 0 & 0 \\ 0 & 0 & 0 & 0 & 2\mu & 0 \\ 0 & 0 & 0 & 0 & 0 & 2\mu \end{bmatrix}; \qquad (1.7)$$

donde $\lambda$ y $\mu$ son los coeficientes de Lamé (Jadamba, 2008) que, a su vez, dependen del módulo de Young, $E$, y de la relación de Poisson, $\upsilon$, según (Shabana, 2008):

$$\lambda=\frac{\upsilon E}{(1+\upsilon)(1-2\upsilon)} \qquad \mu=\frac{E}{2(1+\upsilon)}. \qquad (1.8)$$

### 1.4. Definición de los estados tensional y deformacional planos.

Muchos casos se la práctica ingenieril, se pueden reducir a problemas bidimensionales, donde la tercera dimensión (llamada espesor, $h$) es constante o, por lo menos simétrica con respecto a un cierto plano, llamado plano medio. Convencionalmente, el eje $z$ ($x_3$) del sistema de coordenadas se coloca en la dirección normal al plano medio de la placa, mientras que los ejes $x$ ($x_1$) y $y$ ($x_2$) se colocan sobre dicho plano medio (Fig. 1.2).

Este tipo de cuerpos (o de idealizaciones) se basan en las siguientes asunciones:

a) todas las cargas aplicadas actúan en el plano medio y son simétricas con respecto al plano medio;

b) todas las condiciones de apoyo son simétricas con respecto al plano medio;

c) los desplazamientos, deformaciones y tensiones pueden considerarse uniformes a lo largo del espesor;

En el caso de cuerpos idealmente delgados (Fig. 1.2), se cumple además que las componentes normales y tangenciales de la tensión, en dirección del eje z ($\sigma_z$, $\tau_{yz}$ y $\tau_{zx}$) son nulas o despreciables (Sadd, 2005).

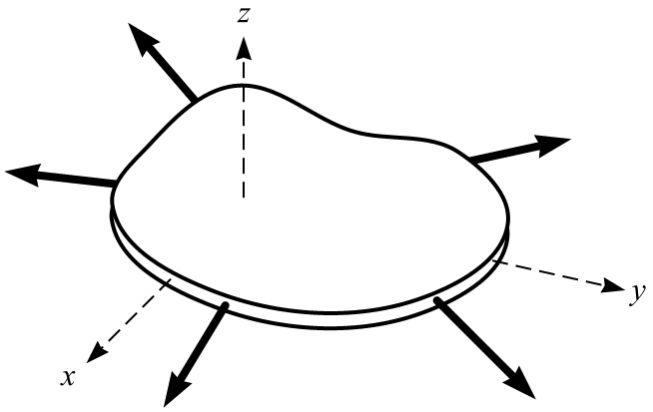



**Figura 1.2.** Representación de un cuerpo bajo estado tensional plano.

Estos casos se conocen como estado tensional plano. En la práctica se consideran como tales los cuerpos en los que el espesor no supera el 10% de la menor dimensión sobre el plano *xy*.

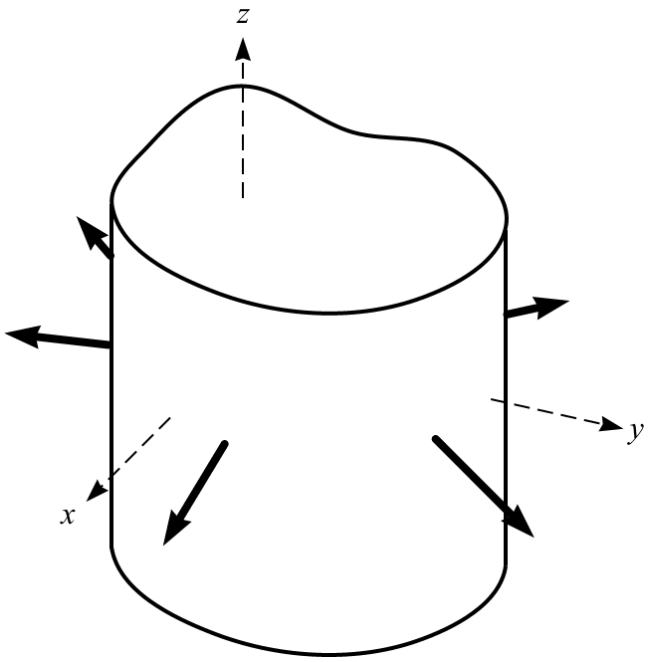



**Figura 1.3.** Representación de un cuerpo bajo estado deformacional plano.

Otro caso ideal es cuando el cuerpo es infinitamente ancho o, al menos, más ancho que la mayor dimensión sobre el plano *xy*. (Fig. 1.3). En esta situación, las componentes de las deformaciones sobre el eje *z*, son nulas o despreciables, y se conoce como estado deformacional plano (Sadd, 2005).

Tanto el estado tensional plano como el estado deformacional se pueden modelar matemáticamente como un problema bidimensional de contorno, definido sobre un dominio plano Ω y con una frontera curvilínea Γ (Fig. 1.4).

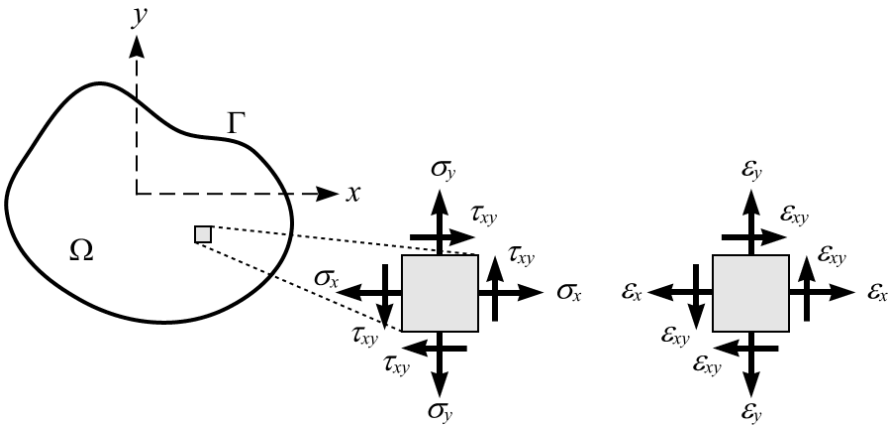



**Figura 1.4.** Modelo de los estados tensional y deformacional planos.

Los campos desconocidos del problema son las tensiones, las deformaciones y los desplazamientos. Teniendo en cuenta la naturaleza bidimensional del problema, la modelación del estado tensional plano, se realiza según las expresiones (Felippa, 2004):

$$\begin{bmatrix} \varepsilon_x \\ \varepsilon_x \\ 2\varepsilon_{xy} \end{bmatrix} = \begin{bmatrix} \partial/\partial x & 0 \\ 0 & \partial/\partial y \\ \partial/\partial y & \partial/\partial x \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix} \text{ (cinemática);} \tag{1.9}$$

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix} \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ 2\varepsilon_{xy} \end{bmatrix} \text{ (relación constitutiva);} \tag{1.10}$$

$$\begin{bmatrix} \partial/\partial x & 0 & \partial/\partial y \\ 0 & \partial/\partial y & \partial/\partial x \end{bmatrix} \begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{bmatrix} + \begin{bmatrix} F_x \\ F_y \end{bmatrix} = \rho \begin{bmatrix} \ddot{u}_x \\ \ddot{u}_y \end{bmatrix} \text{ (equilibrio).} \tag{1.11}$$

y las condiciones de frontera:

$$\begin{bmatrix} u_x \\ u_y \end{bmatrix} = \begin{bmatrix} \hat{u}_x \\ \hat{u}_y \end{bmatrix} \text{ (condición de frontera de desplazamiento);} \tag{1.12}$$

$$\begin{bmatrix} n_x & 0 & n_y \\ 0 & n_y & n_x \end{bmatrix} \begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} \text{ (condición de frontera de fuerza).} \quad (1.13)$$

Las expresiones (1.9), (1.10), (1.11), (1.12) y (1.13) se pueden escribir de forma compacta, utilizando la notación matricial:

$$\boldsymbol{\varepsilon} = \mathbf{D}\mathbf{u}\,; \quad (1.14)$$

$$\boldsymbol{\sigma} = \mathbf{C}\boldsymbol{\varepsilon}\,; \quad (1.15)$$

$$\mathbf{D}^{\mathrm{T}}\boldsymbol{\sigma} + \mathbf{F} = \rho\ddot{\mathbf{u}}\,; \quad (1.16)$$

$$\mathbf{u} = \hat{\mathbf{u}}\,; \quad (1.17)$$

$$\mathbf{N}\boldsymbol{\sigma} = \mathbf{t}\,. \quad (1.18)$$

La matriz de elasticidad del material, en el caso del estado tensional plano, se reduce, de las ecuaciones (1.7) y (1.8) a la forma (Felippa, 2004):

$$\mathbf{C} = \frac{E}{1-\upsilon^2}\begin{bmatrix} 1 & \upsilon & 0 \\ \upsilon & 1 & 0 \\ 0 & 0 & (1-\upsilon)/2 \end{bmatrix}; \quad (1.19)$$

y en el caso del estado deformacional plano:

$$\mathbf{C} = \frac{E(1-\upsilon)}{(1+\upsilon)(1-2\upsilon)}\begin{bmatrix} 1 & \upsilon/(1-\upsilon) & 0 \\ \upsilon/(1-\upsilon) & 1 & 0 \\ 0 & 0 & (1-2\upsilon)/(2-2\upsilon) \end{bmatrix}. \quad (1.20)$$

### 1.5. Formulación débil de los estados planos.

La formulación débil es una forma, generalmente integral, de las ecuaciones diferenciales, que no requiere condiciones estrictas de continuidad y derivabilidad en el dominio del problema (Liu, 2003). Para

los problemas de elasticidad lineal, ésta se obtiene, frecuentemente aplicando el principio de Hamilton, que establece que, de todas las formas de desplazamiento admisibles, la más exacta es aquella que minimiza el funcional lagrangiano, $L$, formado por la diferencia entre energía cinética, $T$, y la energía potencial, $\Pi$; al que se le suma el trabajo de las fuerzas externas, $W_f$ (Milani *et al.*, 2008):

$$\delta \int_{t_1}^{t_2} (T - \Pi + W) \mathrm{d}t = 0 \,. \tag{1.21}$$

Considerando que el cuerpo no se mueve ($\dot{\mathbf{u}} = \mathbf{0}$), la energía cinética es cero, y como la energía potencial, por su parte, se determina como:

$$\Pi = \tfrac{1}{2} \int_\Omega h \boldsymbol{\varepsilon}^{\mathrm{T}} \boldsymbol{\sigma} \, \mathrm{d}\Omega = \tfrac{1}{2} \int_\Omega h \boldsymbol{\varepsilon}^{\mathrm{T}} \mathbf{C} \boldsymbol{\varepsilon} \, \mathrm{d}\Omega \,; \tag{1.22}$$

mientras que el trabajo de las fuerzas externas se calcula por:

$$W = \int_\Omega h \mathbf{u}^{\mathrm{T}} \mathbf{F} \, \mathrm{d}\Omega + \int_\Gamma h \mathbf{u}^{\mathrm{T}} \mathbf{t} \, \mathrm{d}\Gamma \,; \tag{1.23}$$

el principio variacional queda formulado como:

$$\delta L = \delta \left( -\tfrac{1}{2} \int_\Omega h \boldsymbol{\varepsilon}^{\mathrm{T}} \mathbf{C} \boldsymbol{\varepsilon} \, \mathrm{d}\Omega + \int_\Omega h \mathbf{u}^{\mathrm{T}} \mathbf{F} \, \mathrm{d}\Omega + \int_\Gamma h \mathbf{u}^{\mathrm{T}} \mathbf{t} \, \mathrm{d}\Gamma \right) = 0 \,. \tag{1.24}$$

## 1.6. Discretización de dominios planos.

El método de elementos finitos se basa en discretizar el dominio del problema en elementos poligonales de tamaño finito (Fig. 1.4). Como cada elemento tiene $n \geq 3$ nodos, y cada nodo puede realizar dos desplazamientos ($u_x$ y $u_y$), cada elemento posee $2n$ grados de libertad.

Cada uno de los elementos tiene su propio dominio $\Omega_e$ y su propio contorno $\Gamma_e$ (Braess, 2007).

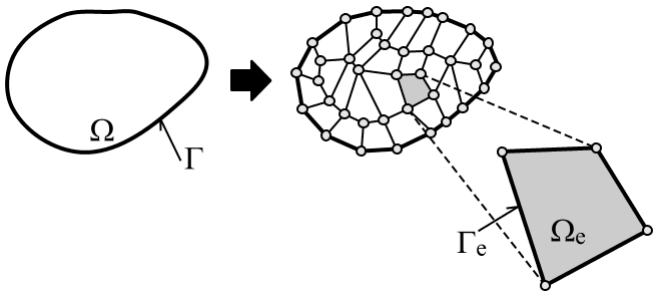



**Figura 1.4.** Discretización del dominio.

Los desplazamientos de cualquier punto del elemento, **u**, pueden calcularse, mediante interpolación, a partir de los desplazamientos nodales, $\mathbf{u}^e$, según la expresión (Askes et al. 2008):

$$\mathbf{u} = \mathbf{N}^e \mathbf{u}^e ; \tag{1.25}$$

donde $\mathbf{N}^e$ es la matriz de funciones de forma $N_i^e(x, y)$, para cada uno de los nodos del elemento:

$$\mathbf{u}^e = [u_{x1}, u_{y1}, ..., u_{xn}, u_{yn}]^T ; \tag{1.26}$$

$$\mathbf{N}^e = \begin{bmatrix} N_1^e & 0 & ... & N_n^e & 0 \\ 0 & N_1^e & ... & 0 & N_n^e \end{bmatrix}. \tag{1.27}$$

Para garantizar la condición de interpolación, cada función de forma, $N_i^e(x, y)$, toma valor uno en el nodo *i*-ésimo y cero, en los restantes nodos (Shi *et al.*, 2007).

Sustituyendo los desplazamientos (1.25) en la expresión (1.14), se obtiene la relación:

$$\boldsymbol{\varepsilon} = \mathbf{D}\mathbf{N}^e \mathbf{u}^e ;$$

que puede simplificarse, definiendo la matriz de deformación-desplazamientos, **B**, como:

$$\mathbf{B} = \mathbf{D}\mathbf{N}^{e}; \tag{1.28}$$

para quedar de la forma:

$$\boldsymbol{\varepsilon} = \mathbf{B}\mathbf{u}^{e}. \tag{1.29}$$

Con lo anterior, el campo de tensiones del elemento puede escribirse en función de los desplazamientos nodales (Felippa, 2004):

$$\boldsymbol{\sigma} = \mathbf{C}\mathbf{B}\mathbf{u}^{e}. \tag{1.30}$$

Para obtener la formulación del problema de elasticidad bidimensional, en elementos finitos, se sustituyen las expresiones (1.29) y (1.30) en la ecuación del principio variacional (1.24), que queda formulada como:

$$\delta\left(\tfrac{1}{2}\int_{\Omega^e} h\mathbf{u}^{eT}\mathbf{B}^{T}\mathbf{C}\mathbf{B}\mathbf{u}^{e}\,d\Omega^{e} + \int_{\Omega^e} h\mathbf{u}^{eT}\mathbf{N}^{eT}\mathbf{F}\,d\Omega^{e} + \int_{\Gamma^e} h\mathbf{u}^{eT}\mathbf{N}^{eT}\mathbf{t}\,d\Gamma^{e}\right) = 0.$$

En el mismo, se definen:

$$\mathbf{K}^{e} = \int_{\Omega^e} h\mathbf{B}^{T}\mathbf{C}\mathbf{B}\,d\Omega^{e}; \tag{1.31}$$

$$\mathbf{f}^{e} = \int_{\Omega^e} h\mathbf{N}^{T}\mathbf{F}\,d\Omega^{e} + \int_{\Gamma^e} h\mathbf{N}^{T}\mathbf{t}\,d\Gamma^{e}; \tag{1.32}$$

con lo cual la expresión del principio variacional queda:

$$\delta\left(-\tfrac{1}{2}\mathbf{u}^{eT}\mathbf{K}^{e}\mathbf{u}^{e} + \mathbf{u}^{eT}\mathbf{f}^{e}\right) = 0;$$

de lo cual resulta, con las derivadas correspondientes, que:

$$\mathbf{K}^{e}\mathbf{u}^{e} - \mathbf{f}^{e} = 0; \tag{1.33}$$

que constituye la expresión fundamental del método de elementos finitos para el problema bidimensional de elasticidad lineal, denominándose $\mathbf{K}^e$ matriz de rigidez y $\mathbf{f}^e$, vector de términos independientes.

## 1.7. Elementos isoparamétricos.

### 1.7.1. Representación isoparamétrica.

Se denomina representación isoparamétrica a aquella en la cual se utilizan las mismas funciones de forma para describir las relaciones entre las posiciones y los desplazamientos de un punto arbitrario del elemento, con las posiciones y los desplazamientos nodales (Liu y Quek, 2003):

$$\begin{bmatrix} 1 \\ x \\ y \\ u_x \\ u_y \end{bmatrix} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ u_{x1} & u_{x2} & \dots & u_{xn} \\ u_{y1} & u_{y2} & \dots & u_{yn} \end{bmatrix} \begin{bmatrix} N_1^e \\ N_2^e \\ \vdots \\ N_n^e \end{bmatrix}. \tag{1.34}$$

La elección de las funciones de forma se realiza de tal forma que cumpla los requerimientos de interpolación, en dependencia del tipo de elemento (Gal y Levy, 2006).

### 1.7.2. Elemento triangular lineal.

El elemento más simple para los dominios bidimensionales es el elemento triangular lineal (T3), que tiene tres nodos en las esquinas (Fig. 1.5) (Felippa, 2004).

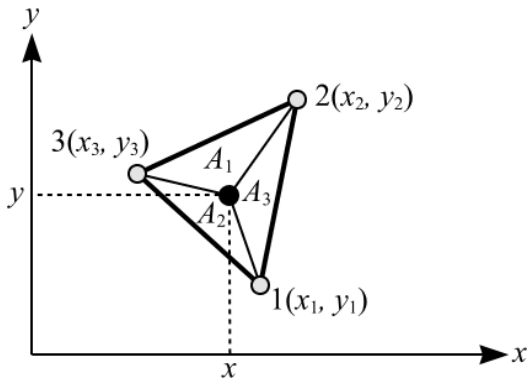



**Figura 1.5.** Elemento triangular lineal (T3).

Para los elementos triangulares, se eligen como coordenadas paramétricas (naturales), para cada nodo, la relación entre el área del triángulo formado por el punto arbitrario ($x$, $y$) y los otros dos nodos, $A_i$, y el área de todo el elemento, $A$:

$$\zeta_i = \frac{A_i}{A} = \frac{1}{2A}(x_j y_k + x_k y + x y_j - x_j y - x_k y_j - x y_k)\,; \tag{1.35}$$

donde $i$, $j$, $k$ es una permutación cíclica de los números 1, 2, 3; y:

$$A = \frac{1}{2}\begin{vmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{vmatrix} = \frac{1}{2A}(x_2 y_3 + x_3 y_1 + x_1 y_2 - x_2 y_1 - x_3 y_2 - x_1 y_3)\,. \tag{1.36}$$

En el caso particular del elemento triangular lineal, las funciones de forma se establecen iguales a estas coordenadas locales:

$$N_1^{\mathrm{e}} = \zeta_1; \quad N_2^{\mathrm{e}} = \zeta_2; \quad N_3^{\mathrm{e}} = \zeta_3\,. \tag{1.37}$$

A pesar de su simplicidad, el elemento triangular lineal no se recomienda para aplicaciones de análisis estructural (Felippa, 2004).

### 1.7.3. Elemento triangular cuadrático.

El elemento triangular cuadrático (T6), posee seis nodos, tres en los vértices y tres en el punto medio de cada lado (Fig. 1.6) ():

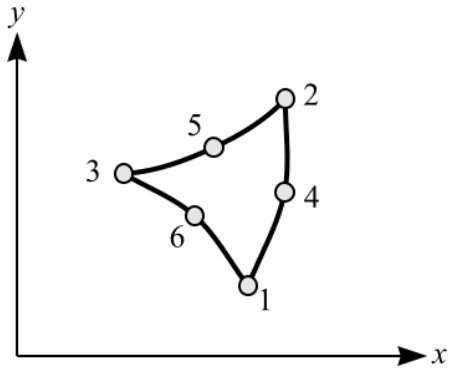



**Figura 1.6.** Elemento triangular cuadrático (T6).

Las funciones de forma, para cada nodo del elemento triangular cuadrático tendrán las expresiones:

$$\begin{aligned} &N_1^e = (2\zeta_1 - 1)\zeta_1; \quad N_2^e = (2\zeta_2 - 1)\zeta_2; \quad N_3^e = (2\zeta_3 - 1)\zeta_3; \\ &N_4^e = \zeta_1\zeta_2; \qquad\quad N_5^e = \zeta_2\zeta_3; \qquad\quad N_6^e = \zeta_3\zeta_1. \end{aligned} \tag{1.38}$$

Nótese que este elemento puede tener aristas curvas, dada la naturaleza cuadrática de la función de forma.

### 1.7.4. Elemento triangular cúbico.

El elemento triangular cúbico (T10), tiene diez nodos, de los cuales tres están en los vértices, seis distribuidos regularmente en los lados y uno en el centroide del triángulo (ver Fig. 1.7).

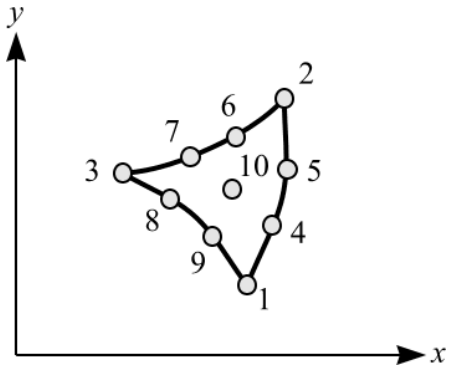



Figura 1.7. Elemento triangular cúbico (T10).

Las funciones de forma, en este tipo de elemento, para cada uno de los nodos, tendrán las expresiones:

$$
\begin{aligned}
&N_1^e = \frac{1}{2}(3\zeta_1 - 1)(3\zeta_1 - 2)\zeta_1; \quad && N_2^e = \frac{1}{2}(3\zeta_2 - 1)(3\zeta_2 - 2)\zeta_2; \\
&N_3^e = \frac{1}{2}(3\zeta_3 - 1)(3\zeta_3 - 2)\zeta_3; \quad && N_4^e = \frac{9}{2}\zeta_1\zeta_2(3\zeta_1 - 1); \\
&N_5^e = \frac{9}{2}\zeta_2\zeta_1(3\zeta_2 - 1); && N_6^e = \frac{9}{2}\zeta_2\zeta_3(3\zeta_2 - 1); \\
&N_7^e = \frac{9}{2}\zeta_3\zeta_2(3\zeta_3 - 1); && N_8^e = \frac{9}{2}\zeta_3\zeta_1(3\zeta_3 - 1); \\
&N_9^e = \frac{9}{2}\zeta_1\zeta_3(3\zeta_1 - 1); && N_{10}^e = 27\zeta_i\zeta_j\zeta_k.
\end{aligned}
\tag{1.39}
$$

Al igual que el elemento triangular cuadrático (y con mayor razón), el cúbico puede representar elementos con aristas curvas.

### 1.7.5. Elemento cuadrangular lineal.

En los elementos cuadrangulares, la parametrización se realiza de un modo diferente. En este caso, se hace corresponder cada punto del cuadrilátero, con un punto de un cuadrado definido sobre la región [-1, 1; -1, 1], y se utilizan las coordenadas ($\xi$, $\eta$) de este cuadrado, como coordenadas paramétricas del cuadrilátero (Fig. 1.8).

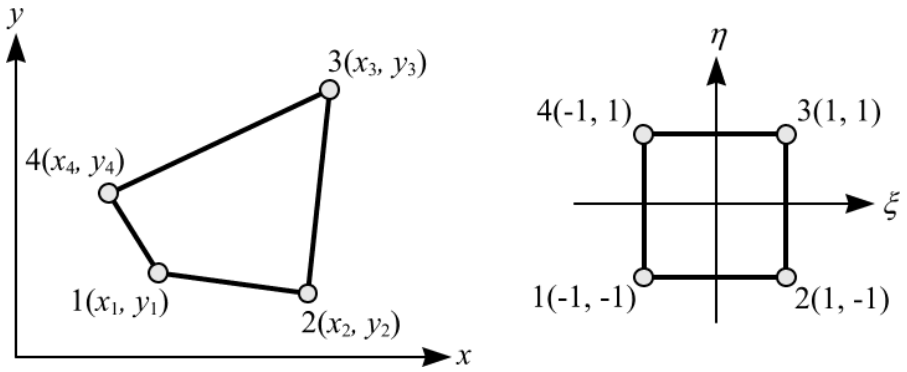



**Figura 1.8.** Elemento cuadrangular lineal (Q4).

El elemento cuadrangular lineal (Q4), tiene cuatro nodos, situados en los vértices del cuadrilátero. Las funciones de forma, para cada uno de estos nodos es:

$$N_1^e = \frac{1}{4}(1-\xi)(1-\eta); \quad N_2^e = \frac{1}{4}(1+\xi)(1-\eta);$$
$$N_3^e = \frac{1}{4}(1+\xi)(1+\eta); \quad N_4^e = \frac{1}{4}(1-\xi)(1+\eta). \tag{1.40}$$

## 1.7.6. Elemento cuadrangular bicuadrático.

El elemento cuadrangular bicuadrático (Q9) tiene nueve nodos, cuatro de ellos en los vértices, otros cuatro en los puntos medios de cada lado y el noveno, en el centroide del cuadrilátero (Fig. 9) (Bastian, 2003).

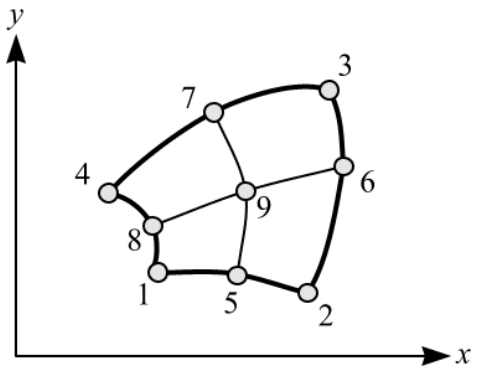



**Figura 1.9.** Elemento cuadrangular bicuadrático (Q9).

Al igual que el elemento triangular cuadrático, el cuadrangular bicuadrático puede tener aristas curvas. Las funciones de forma, para este elemento, tendrán las expresiones:

$$\begin{aligned}
&N_1^e = \frac{1}{4}(1-\xi)(1-\eta)\xi\eta; \qquad N_2^e = \frac{1}{4}(1+\xi)(1-\eta)\xi\eta;\\
&N_3^e = \frac{1}{4}(1+\xi)(1+\eta)\xi\eta; \qquad N_4^e = \frac{1}{4}(1-\xi)(1+\eta)\xi\eta;\\
&N_5^e = -\frac{1}{2}(1-\xi^2)(1-\eta)\eta; \quad N_6^e = \frac{1}{2}(1-\xi)(1-\eta^2)\xi;\\
&N_7^e = -\frac{1}{2}(1-\xi^2)(1+\eta)\eta; \quad N_8^e = \frac{1}{2}(1+\xi)(1-\eta^2)\xi;\\
&N_9^e = -(1-\xi^2)(1-\eta^2).
\end{aligned} \tag{1.41}$$

### 1.7.7. Elementos cuadrangulares de orden superior.

Para los elementos cuadráticos de orden superior, las funciones de forma se calculan multiplicando los polinomios de Lagrange, ajustados para las coordenadas $\xi$ y $\eta$. (Liu y Quek, 2003):

$$L_{p,q}^n = \frac{\sum_{i=1(i\neq p)}^{n}(\xi-\xi_i)}{\sum_{i=1(i\neq p)}^{n}(\xi_p-\xi_i)} \frac{\sum_{i=1(i\neq q)}^{n}(\eta-\eta_i)}{\sum_{i=1(i\neq q)}^{n}(\eta_q-\eta_i)}. \tag{1.42}$$

## 1.8. Integración numérica.

Para determinar el valor numérico de la matriz de rigidez, $\mathbf{K}^e$, de cada elemento, se sustituyen, en la expresión (1.31), los valores de las derivadas parciales con respecto a las coordenadas físicas $\partial N_i^e/\partial x$ y $\partial N_i^e/\partial y$, por las de las coordenadas paramétricas $\partial N_i^e/\partial\xi$ y $\partial N_i^e/\partial\eta$. Para ello, se utiliza la relación (Felippa, 2004):

$$\begin{bmatrix} \partial N_i^{\mathrm{e}} / \partial x \\ \partial N_i^{\mathrm{e}} / \partial y \end{bmatrix} = \mathbf{J}^{-1} \begin{bmatrix} \partial N_i^{\mathrm{e}} / \partial \xi \\ \partial N_i^{\mathrm{e}} / \partial \eta \end{bmatrix}; \tag{1.43}$$

donde **J** es la matriz jacobiana de transformación de coordenadas, que se calcula mediante la expresión:

$$\mathbf{J} = \begin{bmatrix} \partial N_1^{\mathrm{e}} / \partial \xi & \dots & \partial N_n^{\mathrm{e}} / \partial \xi \\ \partial N_1^{\mathrm{e}} / \partial \eta & \dots & \partial N_n^{\mathrm{e}} / \partial \eta \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix}. \tag{1.44}$$

Por lo tanto, la expresión de cálculo de la matriz de rigidez resulta:

$$\mathbf{K}^{\mathrm{e}} = \int_{-1}^{1}\int_{-1}^{1} h\mathbf{B}^{\mathrm{T}}\mathbf{C}\mathbf{B}J \,\mathrm{d}\xi \,\mathrm{d}\eta \,. \tag{1.45}$$

Dado que la obtención de la solución analítica de la integral anterior suele ser computacionalmente costosa, se prefiere resolverla aplicando el método de integración de Gauss, en cual se basa en sustituir la integral por una sumatoria de valores de la función multiplicados por coeficientes (Liu y Quek, 2003):

$$\int_{-1}^{1}\int_{-1}^{1} f(\xi,\eta)\,\mathrm{d}\xi \,\mathrm{d}\eta \approx \sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j f(\xi_i,\eta_j)\,. \tag{1.46}$$

La integración numérica puede llevar con diferentes órdenes de precisión, *n*. Según aumenta este valor, mejora la precisión de los resultados, pero se incrementa el costo computacional del proceso. En la tabla siguiente se muestran los valores de los factores $w_i$ y $\xi_i$ para la integración numérica.

**Tabla 1.1.** Coeficientes de integración numérica.

| $n$ | $i$ | $w_i$ | $\xi i$ |
|---|---|---|---|
| 2 | 1 | 1 | –0,577349 |
| | 2 | 1 | 0,577349 |
| 3 | 1 | 0,555556 | –0,774596 |
| | 2 | 0,888889 | 0 |
| | 3 | 0,555556 | 0,774596 |
| 4 | 1 | 0,347855 | −0,861136 |
| | 2 | 0,652145 | −0,339981 |
| | 3 | 0,652145 | 0,339981 |
| | 4 | 0,347855 | 0,861136 |
| 5 | 1 | 0,236927 | −0,906180 |
| | 2 | 0,478629 | −0,538469 |
| | 3 | 0,568889 | 0 |
| | 4 | 0,478629 | 0,538469 |
| | 5 | 0,236927 | 0,906180 |
| 6 | 1 | 0,171324 | −0,932470 |
| | 2 | 0,360762 | −0,661209 |
| | 3 | 0,467914 | −0,238619 |
| | 4 | 0,467914 | 0,932470 |
| | 5 | 0,360762 | 0,661209 |
| | 6 | 0,171324 | 0,238619 |

## 1.9. Mallado.

La desventaja más importante del análisis de FEM es la necesidad de describir una configuración geométrica en un elemento finito de malla legítimo; generalmente requiere de largas horas en el CPU y una extensa interacción con el usuario (Bastian y Li, 2003).

A pesar del gran número de métodos desarrollados para mallados en las últimas décadas, los algoritmos todavía tienden a ser complejos y toman mucho tiempo de CPU aun para configuraciones simples (El-Hamalawi 2004).

En general, hay dos tipos de mallados computacionales: estructurados y no estructurados (Troyani et al.; 2005). Las mallas estructuradas son producidas sobre una estructura de bloque predeterminada y los nodos son enumerados en una secuencia pre-ordenada. Para una malla estructurada, todos los nodos interiores tienen un número igual de elementos adyacentes. Las mallas no estructuradas por el contrario, son generadas sobre el dominio entero y/o bloques predeterminados y sus nodos son enumerados arbitrariamente. Una malla no estructurada permite que cualquier número de elementos coincida en un mismo nodo (Jilani *et al.*, 2009). Mientras las mallas tanto estructuradas como no estructuradas pueden tomar formas regulares e irregulares, el último a menudo produce formas de elemento que son más versátiles, por lo cual su aplicación computacional es más sencilla (Bellenger y Coorevits, 2005).

En los últimos años, se han llevado a cabo grandes avances en la creación automática de mallas no estructuradas, en particular, las generaciones de mallas triangulares y tetraédricas. Las técnicas de avance frontal, los métodos de Octree y los métodos basados en los algoritmos de Voronoi y Delaunay son algunos de los enfoques bien estudiados en la creación de mallados no estructurados (Tezduyar y Sathe, 2004). El método de Octree es conceptualmente sencillo pero no es completamente satisfactorio para construir una malla con la variación anisótropa del tamaño de los elementos. Por otro lado, tanto el método

de avance frontal como el de Delaunay parecen adaptarse bien a este propósito (Li *et al.*, 2004).

Los algoritmos de generación de malla para modelos sólidos tridimensionales generalmente producen mallas de elementos tetraédricos o hexaédricos, o una combinación de ellos. Muchos resultados de simulación numérica muestran que la malla de elemento hexaédrico es mejor que la malla de elemento tetraédrico porque puede incrementar la exactitud de análisis y reducir la cantidad total de elementos. Desafortunadamente, la generación de una malla completamente formada por elementos hexaédricos es algorítmicamente mucho más compleja que lo de elementos tetraédricos (Lee *et al.*, 2003).

### 1.10. Post-procesamiento.

Se denomina post-procesamiento, al cálculo y la representación de parámetros del problema (Vaz *et al.*, 2009), que se lleva a cabo después de haber calculado las deformaciones, en un problema resuelto por elementos finitos (Gal y Levy, 2006). Entre los parámetros que más comúnmente se calculan son las componentes de la tensión, para lo cual se emplean la ecuación (1.10) y algún criterio de la tensión equivalente, como el de Von Misses (Auricchio y da Veiga, 2003):

$$\tilde{\sigma} = \sqrt{(\sigma_{11} - \sigma_{22})^2 + (\sigma_{22} - \sigma_{33})^2 + (\sigma_{33} - \sigma_{11})^2 + 6(\sigma_{12}^2 + \sigma_{23}^2 + \sigma_{31}^2)} \qquad (1.47)$$

## 1.11. Software para análisis por elementos finitos.

Hay una gran variedad de software disponible para el análisis estructural por elementos finitos. En general se pueden clasificar en tres grandes grupos:

- Software de código abierto: Son aquellos en los que el usuario tiene acceso al código de programación. Dan un gran control sobre el programa y permiten conocer a fondo las técnicas y algoritmos utilizados. Adicionalmente, pueden ser modificados y ampliados. Su principal desventaja es la pobre interfaz de usuario que presentan. Dentro de ellos se destacan el CalculiX, el JFEM, el OOFEM y el Z88 (Munjiza, 2007).
- Software propietario: Son aquellos que se distribuyen bajo una licencia comercial. En ellos, el acceso al código es limitado o, en ocasiones, completamente restringido, lo cual obstaculiza el control y la comprensión del usuario de los cálculos realizados. Algunos de ellos tienen interfaces de usuarios muy amigables. Son muy conocidos el Abacus, el Algor, el Ansys y el Cosmos DesignStar (Barth *et al.*, 2005). En algunas aplicaciones, como la simulación de procesos de maquinado, estos paquetes no siempre alcanzan una aplicación satisfactoria (Umbrello, 2008).
- Bibliotecas: Son conjuntos de subrutinas agrupadas que permiten, a través del uso de un lenguaje de programación, implementar los algoritmos del método de electos finitos para

diversos problemas. Son fácilmente utilizables, y muy convenientes para la docencia y la investigación científica. Su desventaja más señalada es que casi siempre tienen limitaciones en sus implementaciones y una muy pobre documentación (Karris, 2007).

## 1.12. Aplicaciones del método de elementos finitos.

El método de elementos finitos tiene una multitud de aplicaciones en la ingeniería mecánica. Como ejemplos, pueden citarse la modelación de procesos de maquinado (Dixit y Dixit, 2008; Pramanik et al., 2007), análisis de procesos de soldadura (Palmonella *et al.*, 2005) y conformado (Ou y Armstrong, 2006), el diseño de estructuras metálicas (Almeida y Awruch, 2009; Otto y Denier, 2005; Gattulli et al., 2004) y el análisis de materiales compuestos (Niezgoda y Derewońko, 2009).

## 1.13. Conclusiones parciales del Capítulo.

Como resultado del análisis crítico de la bibliografía se ha arribado a las siguientes conclusiones:

1. El método de elementos finitos es de gran importancia para la docencia y las investigaciones de la ingeniería mecánica.
2. El software disponible presenta limitaciones para su aplicación en problemas de propósito específico, tales como la modelación de procesos de maquinado.

3. Se dispone de toda la información necesaria para implementar, en un lenguaje de programación, una biblioteca de funciones para aplicar el método de elementos finitos para problemas de elasticidad lineal.

# CAPÍTULO 2. DESCRIPCIÓN DE LA APLICACIÓN.

En este Capítulo se hace una descripción de la biblioteca desarrollada, destacando los algoritmos utilizados en cada una de las funciones más importantes y presentando los códigos desarrollados.

## 2.1. Algoritmo general.

Para la solución general del problema, se ha seguido el algoritmo siguiente (ver Fig. 2.1).

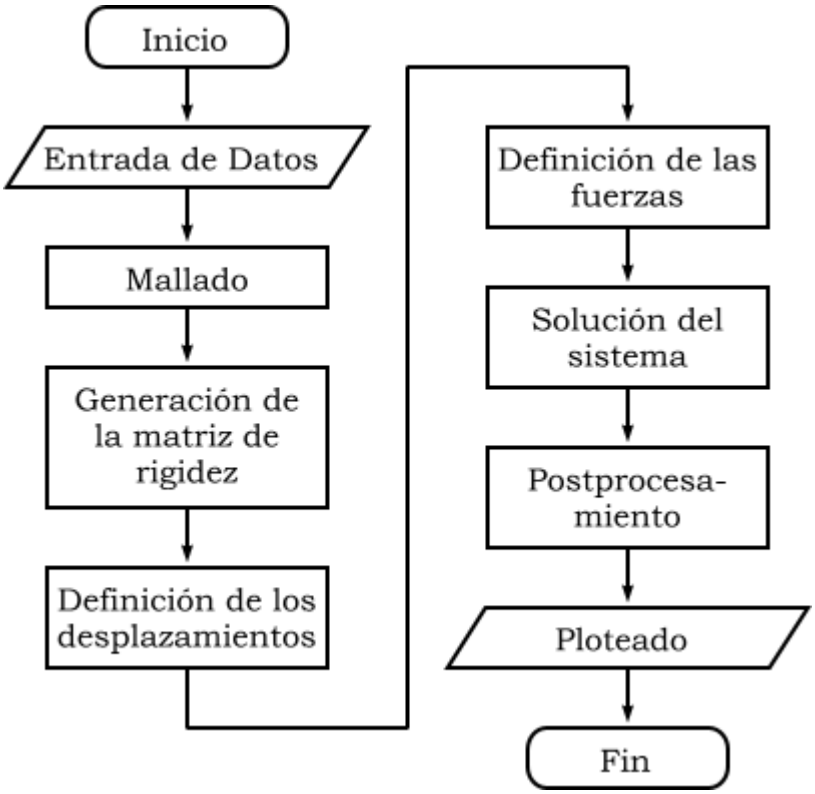



**Figura 2.1**. Algoritmo general.

La implementación de dicho algoritmo se realiza en un archivo de lotes de MATLAB (archivo *.m*), en el cual se implementa el código correspondiente.

En los epígrafes siguientes, se ofrece una descripción detallada de cada una de las secciones del algoritmo.

## 2.2. Entrada de datos.

La entrada de datos es una de las secciones más simples, pero, a la vez una de las más importantes del programa, ya que cualquier error en ella repercutiría negativamente en el desempeño del programa.

El programa requiere, para su correcta ejecución, los siguientes datos:

- *Módulo de elasticidad* (módulo de Young) del material. Debe ser introducido en Pa ($N/m^2$). Se asigna a la variable **e**.
- *Coeficiente de Poisson* del material. Adimensional. Se asigna a la variable **n**.
- *Ancho* (espesor) de la pieza. Se introduce en metros y se asigna a la variable **t**.
- *Geometría de la pieza*. Es ésta una de las variables más complejas a introducir. Se asigna a la variable **G**, la cual es una matriz que contiene una columna por cada uno de los elementos simples que forman la geometría de la pieza (círculos, polilíneas cerradas, rectángulos y elipses). A su vez, cada una de las columnas, tiene cierta estructura que contiene los valores de la geometría del elemento en sus filas, tal como se muestra en la Tabla 2.1.

**Tabla 2.1.** Formato de la matriz de geometría.

| *Tipo de elemento* | *Fila 1* | *Filas restantes* | |
|---|---|---|---|
| | | *Filas* | *Descripción* |
| Círculo | 1 | 2 | Abscisa del centro del círculo |
| | | 3 | Ordenada del centro del círculo |
| | | 4 | Radio del círculo |
| Polilínea | 2 | 2 | Cantidad de vértices, $N$ |
| | | 3 … 2+$N$ | Abscisas de los vértices |
| | | 3+$N$ … 2+2$N$ | Ordenadas de los vértices |
| Rectángulo | 3 | 2 | Cantidad de vértices, $N$ |
| | | 3 … 2+$N$ | Abscisas de los vértices |
| | | 3+$N$ … 2+2$N$ | Ordenadas de los vértices |
| Elipse | 4 | 1 | Abscisa del centro del círculo |
| | | 2 | Ordenada del centro del círculo |
| | | 3 | Longitud del eje mayor |
| | | 4 | Longitud del eje menor |
| | | 5 | Ángulo de rotación |

Los valores de las coordenadas de la pieza se deben introducir en una unidad tal que los valores de las coordenadas no sean muy pequeños (menores que 1), para garantizar el funcionamiento del mallado (que se explicará más adelante). Una vez realizado éste, las coordenadas de los nodos se convierten a metros, para garantizar la compatibilidad de las unidades.

En la Fig. 2.2 se ofrece un ejemplo de matriz de geometría simple, formada por un único elemento (polilínea).

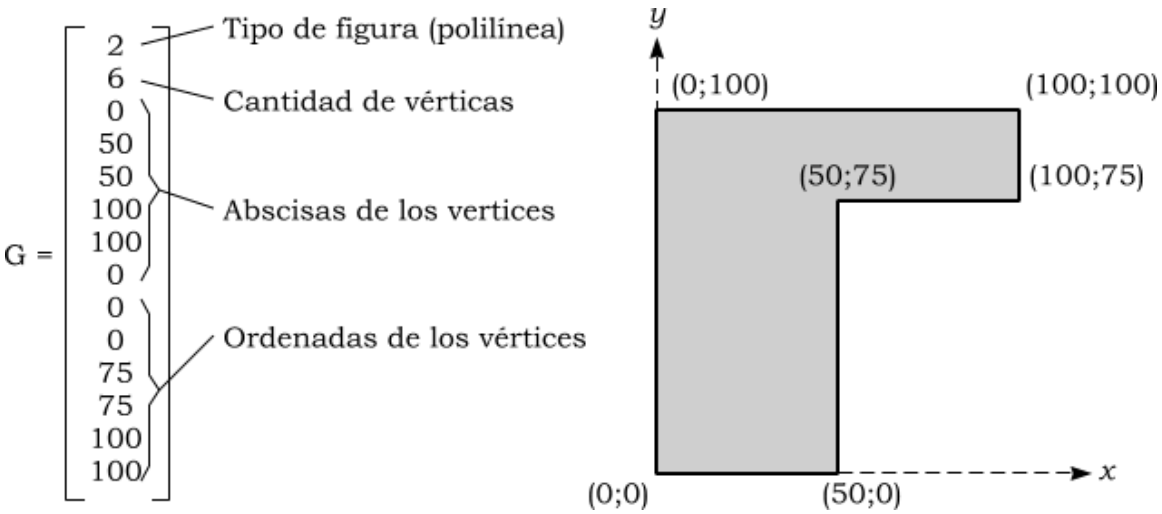



**Figura 2.2.** Ejemplo de matriz de geometría simple.

Se pueden construir geometrías más complejas, formadas por varios elementos simples, los cuales se combinan con las operaciones booleanas (adición, +; intersección, *; y sustracción –).

A partir de la geometría de la región, se genera la llamada *matriz de geometría descompuesta*, lo cual se lleva a cabo a través de la función **decsg**, perteneciente al *toolbox* PDE, de Matlab. Ésta función requiere como parámetros la matriz de geometría, una cadena de caracteres expresando las operaciones a realizar con los diferentes elementos para obtener la figura deseada, **F**, y una matriz de caracteres cuyas columnas sean los nombres asignados, **N**.

Si la región está formada por un solo elemento (como la mostrada en la Fig. 2.2), los dos últimos parámetros se omiten, y sólo es necesario pasar la geometría de la región.

En la Fig. 2.3 se muestra una región compleja que puede formarse a través de dos rectángulos (R1 y R2) y dos círculos (C1 y C2). La fórmula de la operación necesaria para obtener la región sería (C1+R1+R2)–C2.

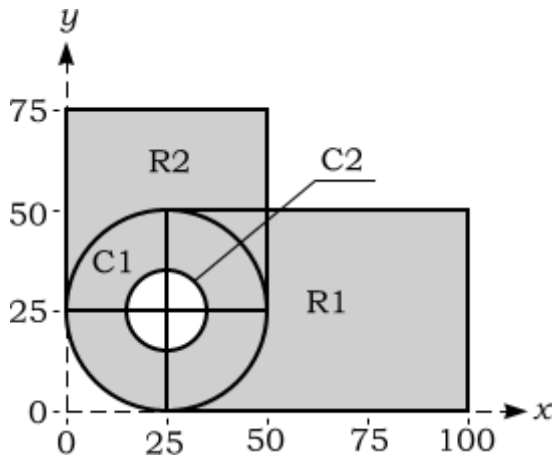



**Figura 2.3.** Ejemplo de región compleja.

El código para generar la matriz de geometría descompuesta, para la región mostrada en la Fig. 2.3 será:

**Código 2.1.** Generación de la geometría descompuesta una región (Fig. 2.3).

```
G = [1 25 25 25 0 0 0 0 0 0; ...
     3 4 25 100 100 25 0 0 50 50; ...
     3 4 0 50 50 0 25 25 75 75; ...
     1 25 25 5 0 0 0 0 0 0]';
F = '(C1+R1+R2)-C2';
N = strvcat('C1','R1','R2','C2').';
GD = decsg(G,F,N);
```

### 2.3. Mallado.

El mallado juega un papel fundamental en una buena implementación del método de los elementos finitos. Para ejecutar el mallado, se emplean

la función **intmesh**, del *toolbox* PDE, la cual emplea el algoritmo de triangulación de Delaunay.

Esta función requiere como parámetro la matriz de geometría descompuesta de la región a mallar. Adicionalmente, se le pueden pasar otros parámetros, como pares nombre-valor, que regulan las características de la malla (ver Tabla 2.2).

**Tabla 2.2.** Parámetros opcionales de la función **initmesh**.

| *Nombre* | *Tipo de dato* | *Valor predeter-minado* | *Descripción* |
|---|---|---|---|
| *Hmax* | *Numérico* | *Estimado* | *Valor máximo de la arista de los elementos del mallado* |
| *Hgrad* | *Numérico* | *1.3* | *Valor de la razón de deformación de los elementos* |
| *Box* | *on/off* | *off* | *Preserva el rectángulo que contiene la región* |
| *Init* | *on/off* | *off* | *Realiza la triangulación a partir de las aristas* |
| *Jiggle* | *off/mean/min* | *mean* | *Realiza el ajuste de la malla* |
| *JiggleIter* | *Numérico* | *10* | *Cantidad de iteraciones del ajuste* |

En concordancia con lo anterior, para generar la malla de la región mostrada en la Fig. 2.3, se utilizará el siguiente código.

**Código 2.2.** Generación de la malla de la región mostrada en la Fig. 2.3.

```
[P,A,T] = initmesh(GD, 'hmax', 5, 'jiggle', 'off');
pdemesh(P,A,T);
axis equal;
```

Nótese que se ha establecido el valor del parámetro **hmax** en 5, y se desactivó el ajuste de malla (**jiggle**); el resto de los parámetros mantiene sus valores predeterminados.

La función **initmesh** devuelve las características geométricas del mallado, contenida en tres matrices: la matriz de puntos, **P**; la matriz de aristas, **A**; la matriz de elementos, **T**.

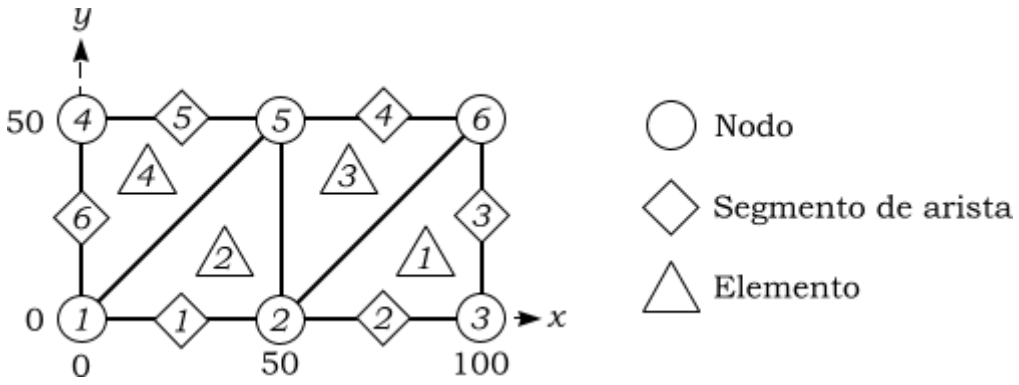



**Figura 2.4.** Ejemplo de malla.

La matriz de puntos, **P**, está formada por dos filas, que contienen los valores de las abscisas y las ordenadas, respectivamente, de los puntos correspondientes a los nodos de la malla. La cantidad de columnas corresponde a la cantidad de nodos. Por ejemplo, la matriz de puntos de la malla mostrada en la Fig. 2.4, tendrá la forma:

$$P = \begin{bmatrix} 0 & 50 & 100 & 0 & 50 & 100 \\ 0 & 0 & 0 & 50 & 50 & 50 \end{bmatrix} \tag{2.1}$$

Por su parte, la matriz de aristas, **A**, está formada por siete filas que contienen las propiedades de los segmentos del contorno. Las dos primeras filas contienen los índices del punto inicial y el punto final del segmento; la tercera y la cuarta los valores inicial y final del parámetro (el factor entre la posición del extremo y del extremo de la arista); la quinta, es el número de la arista a la que pertenece el segmento; la sexta y la séptima son los números de los subdominios que tiene el segmento a

la “izquierda” y a la “derecha”. La cantidad de columnas corresponde con la cantidad de segmentos de aristas. Para la malla de la Fig. 2.4, la matriz de aristas tomará el valor:

$$A = \begin{bmatrix} 1 & 2 & 3 & 6 & 5 & 4 \\ 2 & 3 & 6 & 5 & 4 & 1 \\ 0.0 & 0.5 & 0.0 & 0.0 & 0.5 & 0.0 \\ 0.5 & 1.0 & 1.0 & 0.5 & 1.0 & 1.0 \\ 1 & 1 & 2 & 3 & 3 & 4 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.2}$$

Finalmente, la matriz de elementos, **T**, contiene cuatro filas. Las tres primeras con los números de los nodos del elemento; y la cuarta con el número del subdominio al cual pertenece el elemento. La cantidad de columnas coincide con la cantidad de elementos. Para la malla de la Fig. 2.4, la matriz de elementos tomará el valor:

$$T = \begin{bmatrix} 2 & 1 & 2 & 1 \\ 3 & 2 & 6 & 5 \\ 6 & 5 & 5 & 4 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{2.3}$$

Otra opción para el mallado, es la función **poimesh**, también del *toolbox* PDE, de MATLAB. La misma permite realizar mallados regulares en regiones rectangulares. Requiere como parámetro la matriz de geometría descompuesta de la región, y la cantidad de divisiones en los ejes *x* y *y*. A continuación, se muestra el código para generar la malla regular de la Fig. 2.4:

**Código 2.4.** Generación de la malla regular de la Fig. 2.4.

```
G = [3 4 0 100 100 0 0 0 50 50]';
GD = decsg(G);
[P,A,T] = poimesh(GD,2,1);
pdemesh(P,A,T);
axis equal;
echo off;
```

Tanto en el código anterior, como en el 2.3, se utilizó la función de Matlab, **pdemesh**, que permite visualizar una malla.

### 2.4. Generación de la matriz de rigidez.

La generación de la matriz es el corazón del método de elementos finitos. Se basa en calcular la matriz de rigidez de cada uno de los elementos y, luego, adicionarla a la matriz de rigidez global. En la Fig. 2.5 se muestra el algoritmo global de este proceso. A continuación se explican cada uno de los pasos individuales.

## 2.4.1. Cálculo de la matriz de elasticidad.

El primer paso, es determinar la matriz de elasticidad, E, para el material trabajado, en dependencia del estado tensional o deformacional considerado. Para ello se utilizan las ecuaciones (1.3) y (1.4). Con éste objeto, se implementó la función **elastmat**, cuyo código se muestra a continuación (Código 2.5).

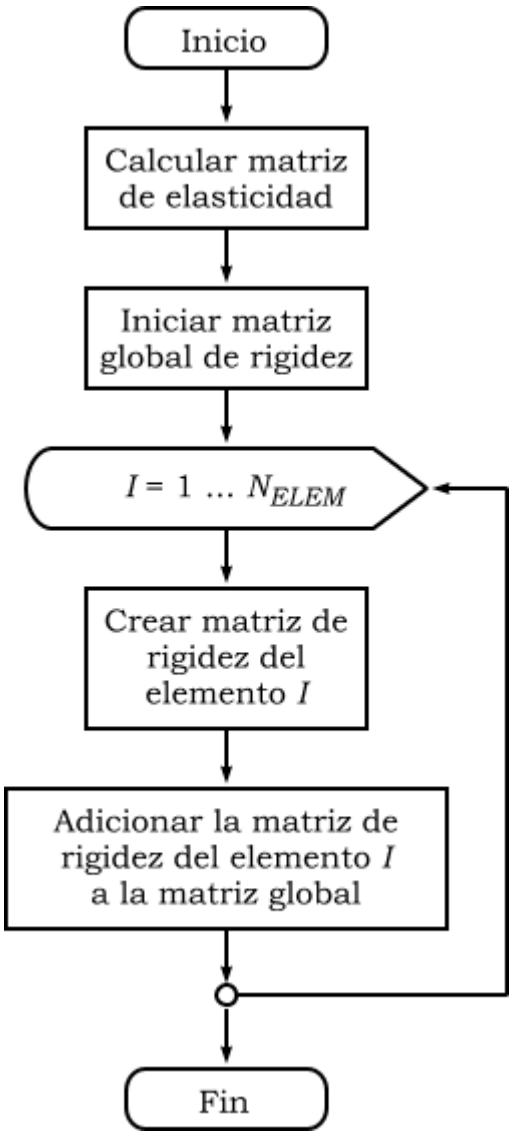



**Figura 2.5.** Algoritmo de generación de la matriz de rigidez.

Esta función requiere como parámetros el módulo de Young y el coeficiente de Poisson del material, así como un valor numérico entero indicando el estado tensional/deformacional del sistema, que será 1 para estado tensional plano y 2 para estado deformacional plano.

La función **elastmat** devuelve una matriz de ($2N \times 2N$) elementos que es la propia matriz de rigidez, donde $N$ es la cantidad de nodos del elemento.

**Código 2.5.** Implementación de la función **elastmat**.

```
function E = elastmat(e, n, s);
% ELASTMAT Calcula la matriz de elasticidad del material
%    Devuelve:
%      E  (matriz 3x3) Matriz de elasticidad.
%    Requiere:
%      e  (escalar) Módulo de Young's.
%      n  (escalar) Coeficiente de Poisson's.
%      s  (entero)  Estado (1 = tensional plano;
%                           2 = deformacional plano).
if (s == 1)
  E = (e./(1-n.^2)).*[1 n 0; n 1 0; 0 0 (1-n)./2];
elseif (s == 2)
  E = (e./((1+n).*(1-2.*n))).*[1-n n 0; n 1-n 0; ...
                               0 0 (1-2.*n)./2];
else
  error('Estado tensional/deformacional desconocido).');
end
```

### 2.4.2. Generación de la matriz de rigidez global.

Tal como se explicó, en el Capítulo 1, la matriz de rigidez global está formada por ($2N$x$2N$) elementos, donde $N$ es la cantidad de nodos de la malla.

Para generar la matriz de rigidez global, vacía, se implementó la función **initsm**, que se lista en el Código 2.6. Esta función requiere como parámetros la matriz de nodos y devuelve la matriz global de rigidez, con todos sus elementos con valor a creo.

Como se puede observar, la matriz de rigidez global se crea como matriz dispersa (*sparse*) ya que, al ser muchos de sus elementos iguales a cero, esto permite un ahorro sustancial de memoria.

**Código 2.6**. Implementación de la función **initsm**.

```
function K = initsm(P);
% INITSM Genera la matriz de rigidez global vacía.
%    Devuelve:
%      K  (matriz 2Nx@n) Matriz de rigidez global vacía.
%    Requiere:
%      P  (2xN) Matriz de puntos (nodos) de la malla.

K = sparse(zeros(size(P,2)*2));
```

### 2.4.3. Cálculo de la matriz de rigidez de un elemento.

Para calcular la matriz de rigidez de un elemento (de tipo T3), se emplea la ecuación (1.10). A este fin, se implementó la función **elemsm**, cuyo código de muestra a continuación.

**Código 2.7.** Implementación de la función **elemsm**.

```
function k = elemsm(P,Tc,E,t);
% ELEMST Calcula la matriz de rigidez para un elemento
%    Devuelve:
%      k  (matriz 6x6) Matriz de rigidez.
%    Requiere:
%      P  (matriz 2xN) Matriz de coordenadas de los nodos.
%      Tr (vector 4x1) Número de los vértices del elemento.
%      E  (matriz 3x3) Matriz de elasticidad.
%      t  (escalar) Espesor del cuerpo.

y12 = P(2,Tc(1)) - P(2,Tc(2));
y23 = P(2,Tc(2)) - P(2,Tc(3));
y31 = P(2,Tc(3)) - P(2,Tc(1));
x21 = P(1,Tc(2)) - P(1,Tc(1));
x32 = P(1,Tc(3)) - P(1,Tc(2));
x13 = P(1,Tc(1)) - P(1,Tc(3));
A = det([P(1,Tc(1)) P(1,Tc(2)) P(1,Tc(3)); ...
         P(2,Tc(1)) P(2,Tc(2)) P(2,Tc(3)); ...
         1          1          1]./2;
B = [y23 0 y31 0 y12 0; 0 x32 0 x13 0 x21; ...
     x32 y23 x13 y31 x21 y12]./(2.*A);
k = t.*A.*(B.'*E*B);
```

Esta función, requiere como parámetros la matriz de nodos, **P**; la columna de la matriz de elementos, **T**, correspondiente al elemento en cuestión. También requiere la matriz de elasticidad del material, **E**; y el espesor de la figura calculada, **t**.

Los valores **y12**, **y32**, **y31**, **x21**, **x32** y **x13**, se calculan de acuerdo con las expresiones (1.6). El área del triángulo, **A**, se calcula según (1.7). La matriz de deformación-desplazamiento, **B**, se determina por (1.8). Finalmente, la matriz de rigidez del elemento, **k**, se computa por la expresión (1.10).

### 2.4.4. Cálculo de la matriz de rigidez global.

Para el cálculo de la matriz de rigidez global se creó la función **calcsm**, que implementa el lazo mostrado en la Fig. 2.5. Esta función se divide en dos partes.

La primera consiste en calcular un vector, **P**, que contiene las posiciones de los desplazamientos en *x* y en *y*, de los nodos del elemento. Esto se realiza según el algoritmo mostrado en la Fig. 2.7 Por ejemplo, para un elemento formado por los nodos [1 3 4], el vector de posiciones será:

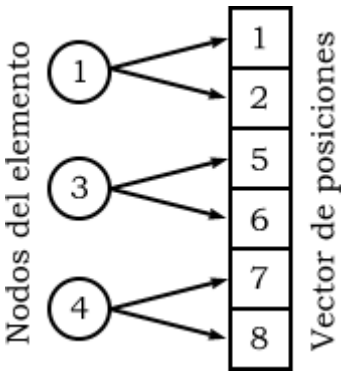



**Figura 2.6.** Ejemplo de relación entre el vector de los nodos del elemento, y el vector de posiciones.

Una vez que esté determinado el vector de posiciones, **P**, se adicionan los valores de la matriz de rigidez del elemento a la matriz global, a través de un lazo doble, tal como se muestra en la Fig. 2.9.

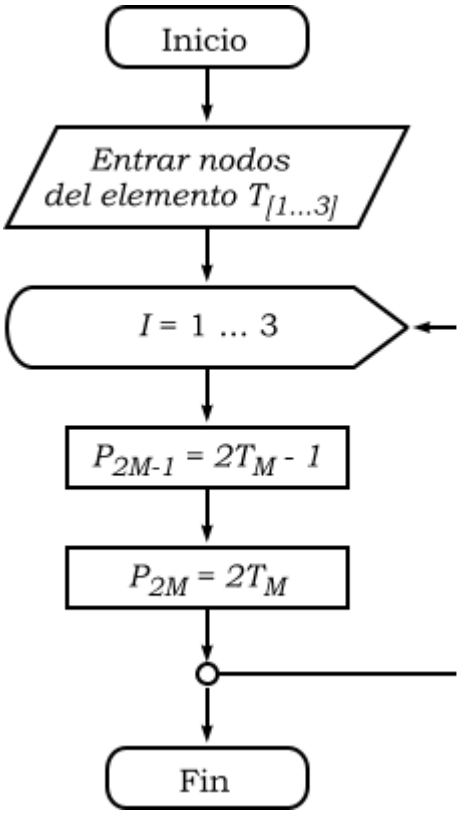



**Figura 2.7.** Algoritmo de cálculo de las posiciones de los elementos en la matriz global.

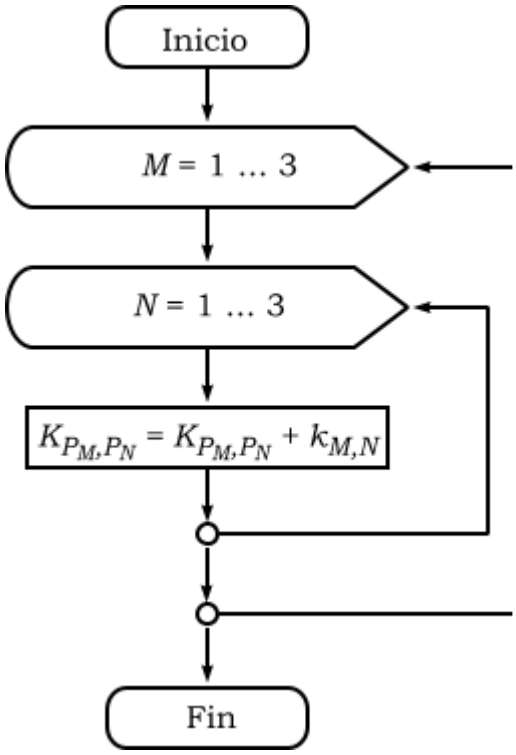



**Figura 2.8.** Algoritmo para la adición de la matriz de rigidez de un elemento a la matriz global.

El código de la función, se muestra en el listado siguiente:

**Código 2.7.** Implementación de la función **calcsm**.

```
function K = calcsm(K,P,T,E,t);
% CALCSM Calcula la matriz de rigidez para un elemento
%    Devuelve:
%      K  (matriz 2Nx2N) Matriz de rigidez global.
%    Requiere:
%      K  (matriz 2Nx2N) Matriz de rigidez global.
%      P  (matriz 2xN) Matriz de coordenadas de los nodos.
%      T  (matriz 4xN) Matriz de elementos.
%      E  (matriz 3x3) Matriz de Elasticidad.
%      t  (scalar) Body thickness.

for I = 1 : size(T,2),
  k = elemsm(P,T(:,I),E,t);
  for M = 1 : 3,
    P(2.*M - 1) = 2.*T(M,I) - 1;
    P(2.*M) = 2.*T(M,I);
  end
  for M = 1 : 6,
    for N = 1 : 6,
      K(P(M),P(N)) = K(P(M),P(N)) + k(M,N);
    end
  end
end
```

## 2.5. Definición de las Cargas.

Las fuerzas y los desplazamientos se almacenarán en los vectores **F** y **U**, respectivamente. Ambos son vectores de una columna y $2N$ filas, donde $N$ es la cantidad de nodos de la malla.

Para inicializar estos vectores se implementó la función **initlc**, cuyo código se muestra a continuación. Los valores del vector de fuerza se inicializan en 0, mientras que los de desplazamiento, en NaN.

**Código 2.8.** Implementación de la función **initlc**.

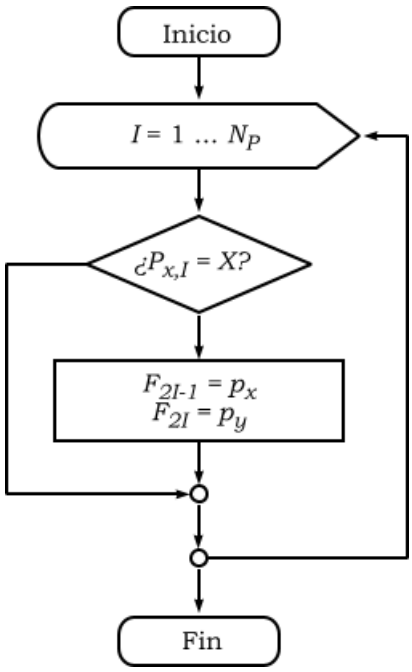
```
function [F, U] = initlc(P);
% INITLC Inicializa los vectores de fuerza y
%        desplazamientos
%    Devuelve:
%      F  (vector 1x2N) Vector de fuerza.
%      U  (vector 1x2N) Vector de desplazamientos.
%    Requiere:
%      P  (matriz 2xN) Matriz de coordenadas de los nodos.

F = zeros(1,size(P,2).*2).';
U = NaN.*ones(1,size(P,2).*2).';
```

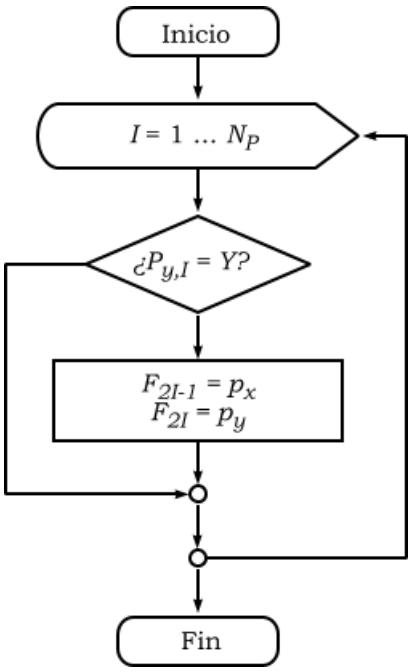
Para establecer las cargas que actúan sobre el sistema, se han implementado las funciones **sethload**, para cargas en caras horizontales; **setvload**, para cargas en superficies verticales; y **setpload**, para cargas puntuales. Los algoritmos correspondientes se muestran en la Fig. 2.9.

Para las cargas en aristas verticales y horizontales, previamente, hay que contar (utilizando un ciclo) la cantidad de nodos que hay en la arista, para dividir la carga aplicada entre la cantidad de nodos.

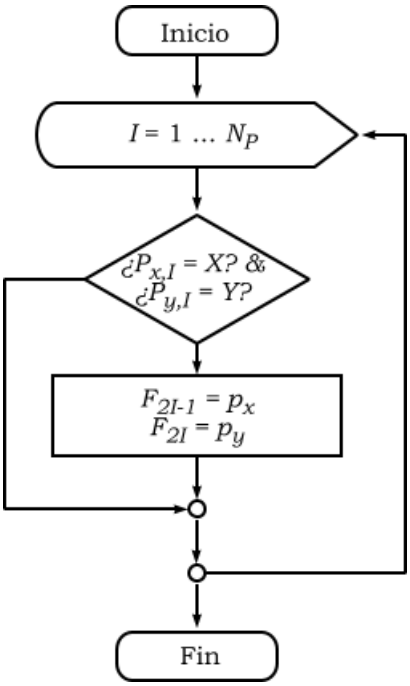
Los códigos de las funciones respectivas se muestran en los listados 2.9a) – c).




a) Función **setvload**




b) Función **sethload**




*c) Función **setpload***

**Figura 2.9.** Implementación de las funciones de establecimiento de cargas.

**Código 2.9a.** Implementación de la función **setvload**.

```
function F = setvload(F,P,x,p);
% SETVLOAD Establece cargas sobre una superficie vertical.
%    Devuelve:
%      F  (vector 1x2N) Vector de fuerza.
%    Requiere:
%      F  (vector 1x2N) Vector de fuerza.
%      P  (matriz 2xN) Matriz de coordenadas de los nodos.
%      x  (escalar) Abscisa de la superficie.
%      p  (vector 1x2) Componentes (x,y) de la fuerza.

c = 0;
for I = 1 : size(P,2),
  if (P(1,I) == x)
    c = c + 1;
  end
end
for I = 1 : size(P,2),
  if (P(1,I) == x)
    F(2.*I - 1) = p(1);
    F(2.*I) = p(2);
  end
end
```

**Código 2.9b.** Implementación de la función **sethload**.

```
function F = sethload(F,P,y,p);
% SETHLOAD Establece cargas sobre una superficie
%          horizontal.
%    Devuelve:
%      F  (vector 1x2N) Vector de fuerza.
%    Requiere:
%      F  (vector 1x2N) Vector de fuerza.
%      P  (matriz 2xN) Matriz de coordenadas de los nodos.
%      y  (escalar) Ordenada de la superficie.
%      p  (vector 1x2) Componentes (x,y) de la fuerza.

c = 0;
for I = 1 : size(P,2),
  if (P(1,I) == x)
    c = c + 1;
  end
end
```

**Código 2.9b.** Implementación de la función **sethload** (cont.).

```
for I = 1 : size(P,2),
  if (P(2,I) == y)
    F(2.*I - 1) = p(1);
    F(2.*I) = p(2);
  end
end
```

**Código 2.9c.** Implementación de la función **setpload**.

```
function F = setpload(F,P,r,p);
% SETVLOAD Establece cargas sobre un punto.
%    Devuelve:
%      F  (vector 1x2N) Vector de fuerza.
%    Requiere:
%      F  (vector 1x2N) Vector de fuerza.
%      P  (matriz 2xN) Matriz de coordenadas de los nodos.
%      r  (vector 1x2) Compenentes (x,y) del punto.
%      p  (vector 1x2) Componentes (x,y) de la fuerza.

for I = 1 : size(P,2),
  if ((P(1,I) == r(1)) & (P(2,I) == r(2)))
    F(2.*I - 1) = p(1);
    F(2.*I) = p(2);
  end
end
```

### 2.6. Definición de las Restricciones de Desplazamientos.

Para el establecimiento de las restricciones de desplazamientos se implementaron las funciones **setvcons**, para las superficies horizontales; **sethcons**, para las superficies verticales; y **setpcons**, para restricciones puntuales. Estas funciones, además de asignar el valor cero a los desplazamientos de los nodos restringidos, también asigna valor NaN (desconocido), a las cargas sobre dichos nodos. Dada la similitud de los algoritmos con los del establecimiento de cargas, se omiten. Los códigos correspondientes se muestran en los listados 2.10a – c.

**Código 2.10 a.** Implementación de la función **setvcons**.

```
function [F,U] = setvcons(F,U,P,x);
% SETVCONS Establece restricciones sobre una sup. vertical.
%    Devuelve:
%      F  (vector 1x2N) Vector de fuerza.
%      U  (vector 1x2N) Vector de desplazamientos.
%    Requiere:
%      F  (vector 1x2N) Vector de fuerza.
%      U  (vector 1x2N) Vector de desplazamientos.
%      P  (matriz 2xN) Matriz de coordenadas de los nodos.
%      x  (escalar) Abscisa de la superficie.

for I = 1 : size(P,2),
  if (P(1,I) == x)
    F(2.*I - 1) = NaN;
    F(2.*I) = NaN;
    U(2.*I - 1) = 0;
    U(2.*I) = 0;
  end
end
```

**Código 2.10 b.** Implementación de la función **sethcons**.

```
function [F,U] = sethcons(F,U,P,y);
% SETHCONS Establece restricciones sobre una sup. horiz.
%    Devuelve:
%      F  (vector 1x2N) Vector de fuerza.
%      U  (vector 1x2N) Vector de desplazamientos.
%    Requiere:
%      F  (vector 1x2N) Vector de fuerza.
%      U  (vector 1x2N) Vector de desplazamientos.
%      P  (matriz 2xN) Matriz de coordenadas de los nodos.
%      y  (escalar) Ordenada de la superficie.

for I = 1 : size(P,2),
  if (P(2,I) == y)
    F(2.*I - 1) = NaN;
    F(2.*I) = NaN;
    U(2.*I - 1) = 0;
    U(2.*I) = 0;
  end
end
```

**Código 2.10 c.** Implementación de la función **setpcons**.

```
function [F,U] = setpcons(F,U,P,x);
% SETPCONS Establece restricciones sobre un punto.
%    Devuelve:
%      F  (vector 1x2N) Vector de fuerza.
%      U  (vector 1x2N) Vector de desplazamientos.
%    Requiere:
%      F  (vector 1x2N) Vector de fuerza.
%      U  (vector 1x2N) Vector de desplazamientos.
%      P  (matriz 2xN) Matriz de coordenadas de los nodos.
%      r  (vector 1x2) Componentes (x,y) del punto.

for I = 1 : size(P,2),
  if ((P(1,I) == r(1)) & (P(2,I) == r(2)))
    F(2.*I - 1) = NaN;
    F(2.*I) = NaN;
    U(2.*I - 1) = 0;
    U(2.*I) = 0;
  end
end
```

## 2.7. Solución del Sistema de Ecuaciones.

### 2.7.1 – Simplificación del sistema de ecuaciones.

El primer paso de la solución del sistema de ecuaciones es su simplificación, eliminando las columnas y filas correspondientes a los desplazamientos nulos. Para ello se implementó la función **simpleq**, cuyo código es:

**Código 2.11.** Implementación de la función **simpleq**.

```
function [A1,x1,b1] = simpleq(A,x,b);
% SIMPLEQ Simplifica el sistema de ecuaciones.
%    Devuelve:
%      A1 (matriz NxN) Matriz de coeficientes.
%      x1 (vector 1xN) Vector de variables.
%      b1 (vector 1xN) Vector de términos independientes.
%    Requiere:
%      A  (matriz NxN) Matriz de coeficientes.
%      x  (vector 1xN) Vector de variables.
%      b  (vector 1xN) Vector de términos independientes.

I = 1;
while (I <= size(x,1))
  if (x(I) == 0)
    A(:,I) = [];
    A(I,:) = [];
    x(I) = [];
    b(I) = [];
  else
    I = I + 1;
  end
end
A1 = A;
x1 = x;
b1 = b;
```

## 2.7.2 – Solución del sistema de ecuaciones.

El sistema de ecuaciones lineales obtenido, se resuelve multiplicando la inversa de la matriz de rigidez simplificada por el vector de fuerzas simplificadas.

Seguidamente, se procede a reintroducir los valores del vector de desplazamientos simplificado, en el vector de desplazamientos original y, con este se calcula el vector de fuerzas completo.

Estas operaciones se implementaron en la función **solveq**, cuyo código se muestra a continuación:

**Código 2.12.** Implementación de la función **solveq**.

```
function [F,U] = solveq(K,F,U,K1,F1,U1);
% SIMPLEQ Simplifica el sistema de ecuaciones.
%    Devuelve:
%      F  (vector 1xN) Vector de fuerzas.
%      U  (vector 1xN) Vector de desplazamientos.
%    Requiere:
%      K  (matriz NxN) Matriz de rigidez.
%      F  (vector 1xN) Vector de fuerzas.
%      U  (vector 1xN) Vector de desplazamientos.
%      K1 (matriz NxN) Matriz de rigidez simplificada.
%      F1 (vector 1xN) Vector de fuerzas simplificado.
%      U1 (vector 1xN) Vector de desplazamientos simpl.

U1 = inv(K1)*F1;
C = 1;
for I = 1 : size(U,1),
  if (isnan(U(I)))
    U(I) = U1(C);
    C = C + 1;
  end
end
F = K*U;
```

## 2.8. Postprocesamiento.

El postprocesamiento tiene como objetivo fundamental calcular las tensiones de cada elemento a partir de los desplazamientos de sus nodos. Esto se implemento en la función **postproc**.

Esta función está formada por un ciclo que recorre todos los elementos de la malla. Para cada uno de los elementos, se calcula la matriz deformación-desplazamiento, y con ella se determinan las componentes de la deformación.

Seguidamente, con la deformación, se calculan las componentes de las tensiones, utilizando la matriz de elasticidad. La tensión equivalente de Von Misses se calcula según la expresión:

$$\tilde{\sigma} = \sqrt{(\sigma_{11} + \sigma_{22})^2 - 3(\sigma_{11}\sigma_{22} - \sigma_{12}^2)} \qquad (2.4)$$

que se deriva de la (1.13), para el caso de estados planos. El código de la función se muestra a continuación.

**Código 2.13.** Implementación de la función **postproc**.

```
function [S,Q,Qe] = postproc(U,T,P,E);
% POSTPROC Postprocesamiento.
%    Devuelve:
%      S  (matriz 3xN) Componentes de la deformación.
%      Q  (matriz 3xN) Componentes de la tensión.
%      Qe (vector 1xN) Tensión equivalente (Von Misses).
%    Requiere:
%      U  (vector 1xN) Vector de desplazamientos.
%      T  (matriz 4xN) Matriz de elementos.
%      P  (matriz 2xN) Matriz de coordenadas de los nodos.
%      E  (matriz 3x3) Matriz de elasticidad.

S = zeros(size(T,2),3);
Q = zeros(size(T,2),3);
Qe = zeros(size(T,2),1);

for I = 1 : size(T,2),
  y12 = P(2,T(1,I)) - P(2,T(2,I));
  y23 = P(2,T(2,I)) - P(2,T(3,I));
  y31 = P(2,T(3,I)) - P(2,T(1,I));
  x21 = P(1,T(2,I)) - P(1,T(1,I));
  x32 = P(1,T(3,I)) - P(1,T(2,I));
  x13 = P(1,T(1,I)) - P(1,T(3,I));
  A = det([P(1,T(1,I)) P(1,T(2,I)) P(1,T(3,I)); ...
           P(2,T(1,I)) P(2,T(2,I)) P(2,T(3,I)); ...
           1           1           1]./2;
  B = [y23 0 y31 0 y12 0; 0 x32 0 x13 0 x21; ...
       x32 y23 x13 y31 x21 y12]./(2.*A);
  u = [U(T(1,I).*2-1), U(T(1,I).*2), U(T(2,I).*2-1), ...
       U(T(2,I).*2), U(T(3,I).*2-1), U(T(3,I).*2)].';
  S(I,:) = (B*u).';
  Q(I,:) = (E*(S(I,:)).').';
  Qe(I) = ((Q(I,1) + Q(I,2)).^2 - 3.*(Q(I,1).*Q(I,2) - ...
            Q(I,3).^2)).^(1./2);
end
```

## 2.9. Ploteado de los Resultados.

Finalmente, se implementó la función **plotvm**, para mostrar el gráfico de tensiones equivalente (de Von Misses), cuyo código se muestra a continuación.

**Código 2.14.** Implementación de la función **plotvm**.

```
function plotvm(P,T,Q,Qmin,Qmax);
% PLOTVM Plotea las tensiones equivalentes de los elementos
%    Requiere:
%      P    (matriz 2xN) Matriz de coord. de los nodos.
%      T    (matriz 4xN) Matriz de elementos.
%      Q    (vector 1xN) Matriz de tensiones.
%      Qmin (escalar) Tensión mínima (opcional).
%      Qmax (escalar) Tensión máxima (opcional).

hold on;
axis equal;
if (nargin < 4)
  Qmin = min(Q);
  Qmax = max(Q);
end

caxis([Qmin Qmax]);
c = colormap;
for I = 1 : size(T,2),
  N1 = P(:,T(1,I)).';
  N2 = P(:,T(2,I)).';
  N3 = P(:,T(3,I)).';
  H = patch([N1(1) N2(1) N3(1)], ...
            [N1(2) N2(2) N3(2)], Q(I));
  n = round((size(c,1) - 1).*(Q(I) - Qmin)./ ...
                              (Qmax - Qmin)) + 1;
  set(H, 'EdgeColor', c(n,:));
end
colorbar;
```

## 2.10. Conclusiones parciales del capítulo.

A través del desarrollo de este capítulo, se ha podido arribar a las siguientes conclusiones parciales.

1. Se ha implementado una biblioteca para la aplicación del método de elementos finitos, que permite resolver problemas de elasticidad lineal en dominios bidimensionales complejos.
2. La biblioteca implementa elementos triangulares lineales, pero dada la organización y la programación de las subrutinas que la componen, puede ser extendida, en un futuro, a otros tipos de elementos, con facilidad.
3. La biblioteca cuenta con capacidades numéricas y gráficas para la visualización de los resultados, lo que permite su análisis e interpretación por parte del usuario.

# CAPÍTULO 3. ESTUDIOS DE CASO.

Para ilustrar el uso de la biblioteca desarrollada, así como para verificar su correcto funcionamiento, se implementaron varios ejemplos, como estudios de casos.

A continuación, se exponen dichos ejemplos, comparando sus resultados con los obtenidos por otros programas de FEM.

## 3.1. Ejemplo 1: Viga sometida a flexión.

El primer ejemplo, corresponde a una viga de acero AISI 1020, cuyas propiedades mecánicas son: módulo de Young $E = 2 \cdot 10^{11}$ Pa y coeficiente de Poisson $\nu = 0{,}29$. Las dimensiones de la viga son: sección transversal de (25 x 25) mm y longitud 100 mm. Está empotrada en un extremo y tiene una fuerza de 2000 N, dirigida verticalmente hacia abajo, aplicada en el otro extremo.

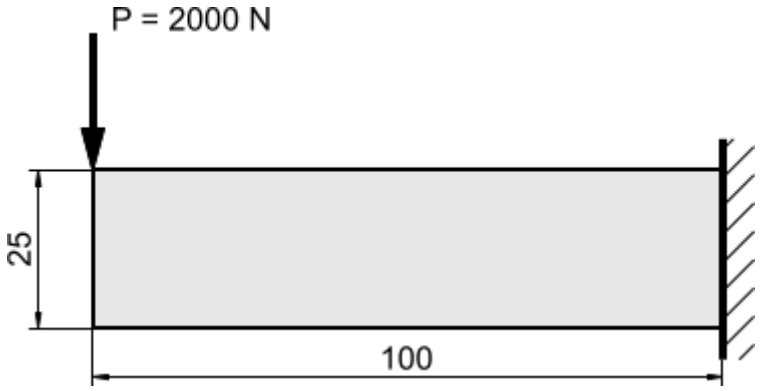



**Figura 3.1.** Ejemplo No. 1 (viga empotrada).

Para la solución de este ejemplo, se implementó el siguiente código:

**Código 3.1.** Ejemplo No. 1 (viga empotrada).

```
echo on;
% Ejemplo No. 1 (viga emprotrada sometida a flexión
%                con mallado regular)

% Datos
e = 2e11;  % Pa
n = 0.29;
t = 0.025; % m

% Definición de la geometría
G = [3 4 0 100 100 0 0 0 25 25]';
GD = decsg(G);

% Mallado
[P,A,T] = poimesh(GD,30,10);
P = P./1e3;

% Generación de la matriz de rigidez
E = elastmat(e,n,2);
K = initsm(P);
K = calcsm(K,P,T,E,t);

% Definición de las cargas y las restricciones
[F,U] = initlc(P);
F = setpload(F,P,[0 0.025],[0 -2000]);
[F,U] = setvcons(F,U,P,0.1);

% Solución del sistema de ecuaciones
[K1,U1,F1] = simpleq(K,U,F);
[F,U] = solveq(K,F,U,K1,F1,U1);

% Postprocesamiento
[S,Q,Qe] = postproc(U,T,P,E);

% Ploteado
plotvm(P,T,Qe);

echo off;
```

Nótese que se emplea la función **poimesh**, para generar un mallado regular, ya que la superficie es rectangular. La carga se

establece con la función **setpload**, sobre el punto (0; 0,025) y la restricción con la función **setvcons**, sobre la superficie x = 0,1.

El gráfico de los resultados se muestra en la Fig. 3.2. La tensión máxima, según el criterio de Von Misses, es de 87,5 MPa (se exceptúa la tensión de los nodos donde está aplicada la fuerza).

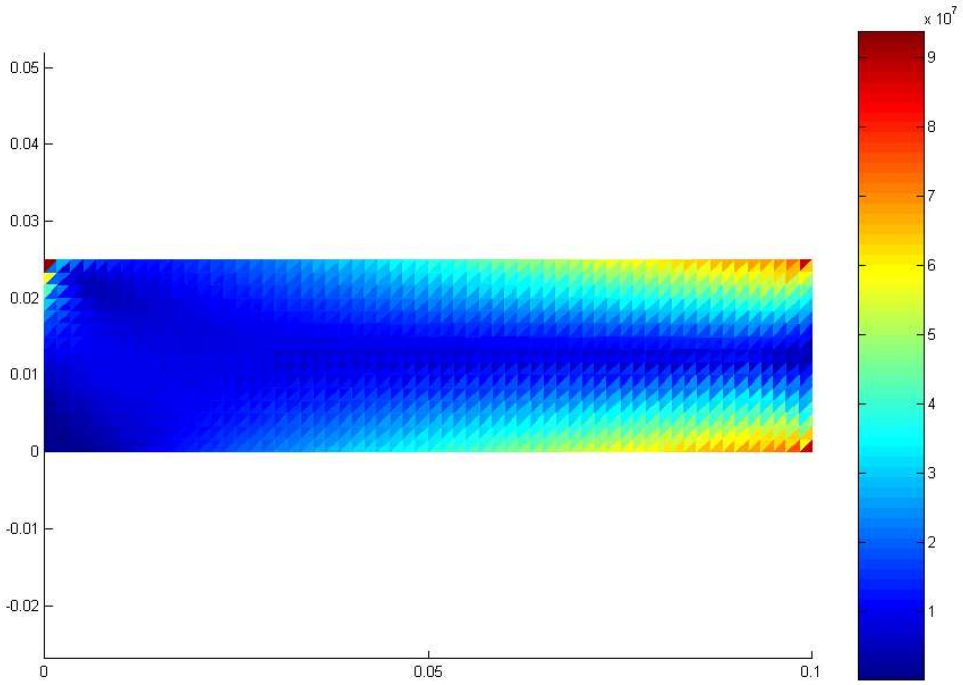



**Figura 3.2.** Gráfico de tensiones del Ej. 1.

Para comprobar los resultados, se modeló el mismo problema en el programa Cosmos DesingStar. El valor máximo de las tensiones de Von Misses fue de 84,4 MPa , lo que significa un 3,67 % de diferencia entre ambos modelos.

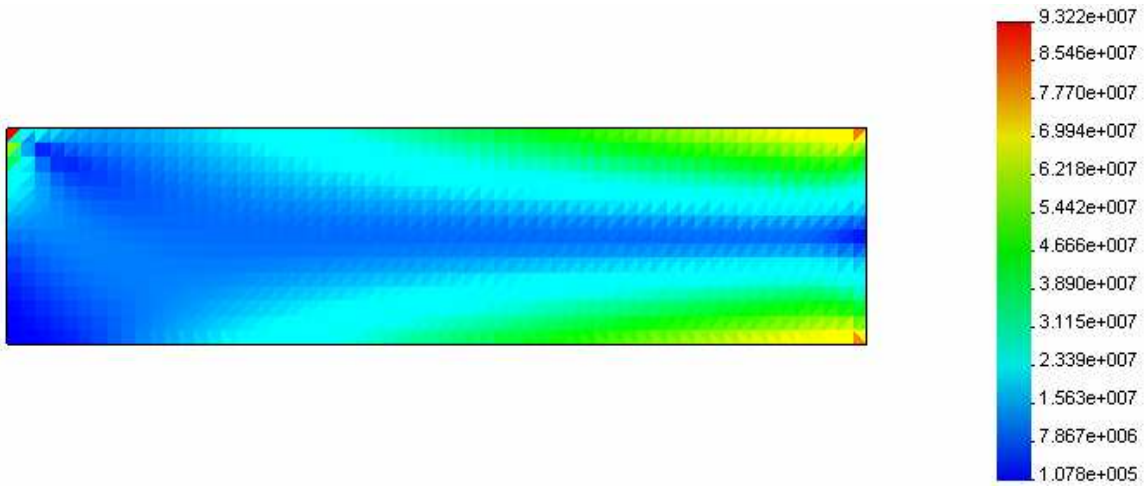



*Figura 3.3 – Gráfico de las tensiones del Ej. 1 obtenidas en Cosmos.*

### 3.2. Ejemplo No. 2 – Barra en L.

El segundo ejemplo es una viga en forma de L, empotrada en su extremo inferior y con una fuerza horizontal de 500 N, aplicada a su otro extremo. El material de la viga es aluminio AISI 2014, con un módulo de Young, $E = 7{,}3 \cdot 10^{10}$ MPa y un coeficiente de Poisson, $\nu = 0{,}33$; t = 25 mm

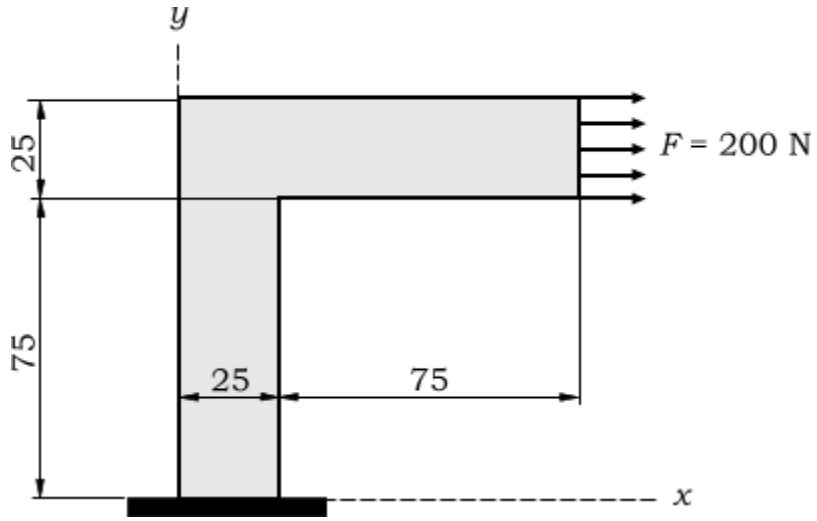



**Figura 3.4.** Ejemplo 2: Viga en L.

Para la solución de este ejemplo, se implementó el siguiente código:

**Código 3.2.** Ejemplo No. 2 (viga en L).

```
echo on;
% Ejemplo No. 2 - Viga en L

% Datos
e = 7.3e10; % Pa
n = 0.33;
t = 0.025; % m

% Definición de la geometría
G = [2 6 0 25 25 100 100 0 0 0 75 75 100 100]';
GD = decsg(G);

% Mallado
[P,A,T] = initmesh(GD,'hmax',3);
P = P./1e3;

% Generación de la matriz de rigidez
E = elastmat(e,n,2);
K = initsm(P);
K = calcsm(K,P,T,E,t);

% Definición de las cargas y las restricciones
[F,U] = initlc(P);
F = setvload(F,P,0.1,[-500 0]);
[F,U] = sethcons(F,U,P,0);

% Solución del sistema de ecuaciones
[K1,U1,F1] = simpleq(K,U,F);
[F,U] = solveq(K,F,U,K1,F1,U1);

% Postprocesamiento
[S,Q,Qe] = postproc(U,T,P,E);

% Ploteado
plotvm(P,T,Qe);

echo off;
```

La distribución de tensiones obtenidas, se muestra en la figura 3.5. La tensión de Von Misses fue de 17,9 MPa.

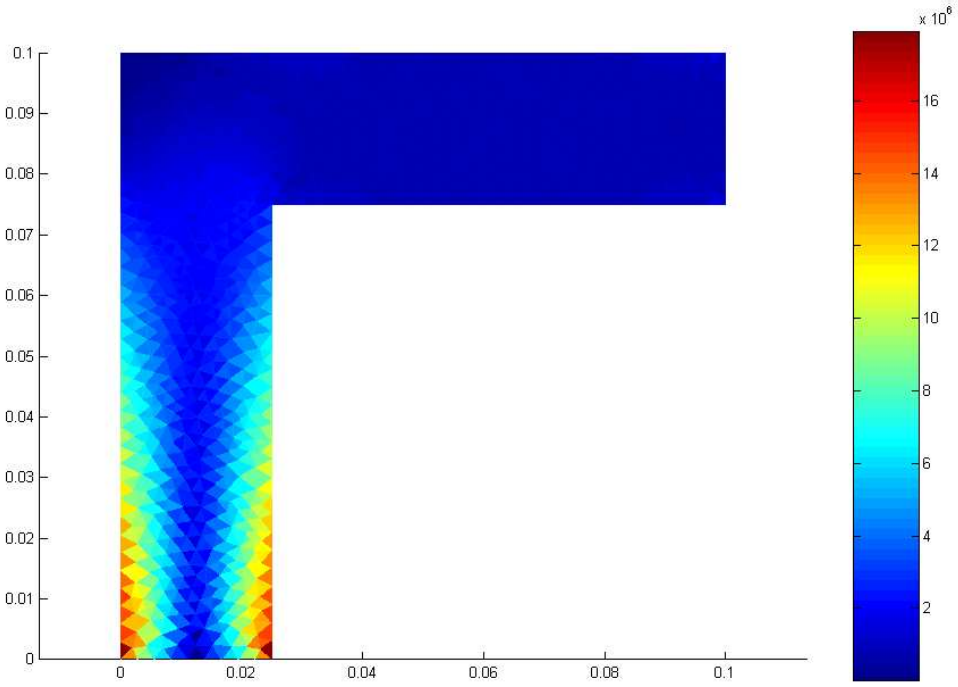



**Figura 3.5.** Gráfico de tensiones del Ej. 2.

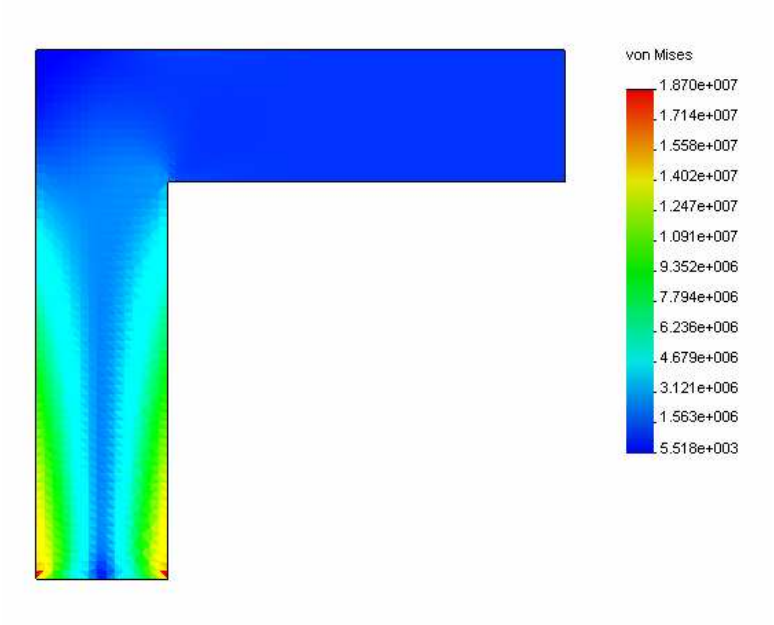



**Figura 3.6.** Gráfico de las tensiones del Ej. 2 obtenidas en Cosmos.

El mismo problema resuelto en Cosmos DesignStar, arrojó una tensión máxima de 18,7 MPa, representando una diferencia de 4,28%. La distribución de tensiones obtenida se muestra en la Fig. 3.6.

### 3.3. Conclusiones parciales del capítulo.

A través de los estudios de casos presentados en el capítulo, se ha podido arribar a las siguientes conclusiones:

1. El funcionamiento de la biblioteca es estable y fiable, sin que se hayan detectado errores en su utilización.
2. El costo computacional del uso de la librería, en los casos estudiados, es similar al de los paquetes profesionales.
3. Los resultados calculados a través del uso de la biblioteca, coinciden, con un error razonable, con los reportados por otras aplicaciones de elementos finitos.

# CONCLUSIONES.

Como resultado de lo expuesto en el presente trabajo, se enuncian las siguientes conclusiones:

1. Se implementó una librería de funciones de MATLAB para resolver problemas de elasticidad lineal a través del método de elementos finitos.
2. La librería implementada utiliza elementos de tipo triangular lineal, pudiendo ser incrementada en el futuro con otros tipos de elementos.
3. A través de los estudios de casos, la librería desarrollada ha mostrado ser eficaz, computacionalmente eficiente y precisa en sus resultados.

## RECOMENDACIONES.

Como resultado del trabajo se recomienda lo siguiente:

1. Incrementar la librería con otros tipos de elementos, en especial con los elementos triangulares cuadrático y cúbico y con los cuadrangulares lineal, bicuadrático y bicúbico.
2. Implementar métodos de mallado más efectivos, que tengan en cuenta el mallado adaptativo y la importación de modelos desde sistemas de CADD.
3. Analizar la posible ampliación de la librería para el análisis de problemas tridimensionales y no lineales.

## BIBLIOGRAFÍA.


1. Almeida, F.S.; Awruch, A.M. (2009). "Design optimization of composite laminated structures using genetic algorithms and finite element analysis". *Composite Structures* 88 (3) pp. 443-454.
2. Askes, H.; Morata, I.; Aifantis, E.C. (2008). "Finite element analysis with staggered gradient elasticity". *Computers and Structures* 86 (11-12) pp. 1266-1279.
3. Auricchio, F.; da Veiga, L.B. (2003). "On a new integration scheme for von-Mises plasticity with linear hardening". *International Journal for Numerical Methods in Engineering* 56 (10) pp. 1375–1396.
4. Avril, S.; Pierron, F. (2007). "General framework for the identification of constitutive parameters from full-field measurements in linear elasticity". *International Journal of Solids and Structures* 44 (14-15) pp. 4978-5002.
5. Barth, T.J.; Griebel, M.; Keyes, D.E.; Nieminen, R.M.; Roose, D.; Schlick, T. (Eds.) (2005). *Design of adaptive finite element software: The finite element toolbox ALBERTA*. Berlín (Alemania): Springer.
6. Bastian, M.; Li, B.Q. (2003). "An efficient automatic mesh generator for quadrilateral elements implemented using C++". *Finite Elements in Analysis and Design* 39 (9) pp.905-930.

7. Braess, D. (2007). *Finite element: theory, fast solvers, and applications in elasticity theory*, Cambridge (UK): Cambridge University Press, ISBN 9780511279102.
8. Dixit, P.M.; Dixit, U.S. (2008). *Modeling of metal forming and machining processes by finite element and soft computing methods*. London (UK): Springer-Verlag.
9. Duan, H.-Y.; Lin, Q. (2005). "Mixed finite elements of least-squares type for elasticity". Computer Methods in Applied Mechanics and Engineering 194 (9-11) pp. 1093-1112.
10. El-Hamalawi, A. (2004). "A 2D combined advancing front-Delaunay mesh generation scheme". *Finite Elements in Analysis and Design* 40, pp. 967-989.
11. Bellenger, E.; Coorevits, P. (2005). "Adaptive mesh refinement for the control of cost and quality in finite element analysis". *Finite Elements in Analysis and Design* 41 (15) pp. 1413-1440.
12. Felippa, C.A. (2004). *Introduction to finite element methods* [on-line]. Boulder, CO (USA): University of Colorado, available in: <*http://www.colorado.edu*>.
13. Fernández-Méndez, S.; Huerta, A. (2004). "Imposing essential boundary conditions in mesh-free methods". *Computer Methods in Applied Mechanics and Engineering* 193 (12-14) pp. 1257-1275.

14. Gal, E.; Levy, R. (2006). "Geometrically nonlinear analysis of shell structures using a flat triangular shell finite element". *Archives of Computational Methods in Engineering* 13 (3) pp. 331-388.
15. Gattulli, V.; Martinelli, L.; Perotti, F.; Vestroni, F. (2004). "Nonlinear oscillations of cables under harmonic loading using analytical and finite element models". *Computer Methods in Applied Mechanics and Engineering* 193 (1-2) pp. 69-85.
16. Jadamba, B.; Khan, A.A.; Raciti, F. (2008). On the inverse problem of identifying Lamé coefficients in linear elasticity. *Computers & Mathematics with Applications* 56 (2) pp. 431-443.
17. Karris, S.T. (2007). *Numerical analysis using MATLAB and Excel*. 3rd. Ed. New York (USA): Orchard Publications. ISBN: 978-1-934404-04-1.
18. Koenraad, J. (2007). *Computational material engineering*. Oxford (UK): Elsevier, ISBN 978-0-12-369468-3.
19. Lee, K.Y.; Kim, I.I.; Cho, D.Y.; Kim, T.W. (2003). "An algorithm for automatic 2D quadrilateral mesh generation with line constraints". *Computer-Aided Design* 35 (12) pp.1055-1068.
20. Li, L.X., Kunimatsu, S., Han, X.P., Xu, S.Q. (2004). "The analysis of interpolation precision of quadrilateral elements". *Finite Elements in Analysis and Design* 41 pp. 91-108.
21. Liu, G.R.; Quek, S.S. (2003). *The finite element method: a practical course*. Oxford (UK): Elsevier. ISBN 0-7506-5866-5.

22. Ma, C.; Wang, D.; Wang, Y. (2009). "The finite element analysis of a fractional-step method for the time-dependent linear elasticity equations". *Nonlinear Analysis: Real World Applications* 10 (2) pp. 1210-1219.

23. Milani, R.; Quarteroni, A.; Rozza, G. (2008). "Reduced basis method for linear elasticity problems with many parameters". Computer Methods in Applied Mechanics and Engineering 197 (51-52) pp. 4812-4829

24. Merlini, T; Morandini, M. (2005). "The helicoidal modeling in computational finite elasticity. Part III: Finite element approximation for non-polar media". International Journal of Solids and Structures 42 (24-25) pp. 6475-6513.

25. Meyers, M.A.; Chawla, K.K. (2008). *Mechanical behavior of materials*. 2nd Ed. Cambridge (UK): Cambridge University Press. ISBN 978-0-511-45557-5.

26. Munjiza, A. (2007). *Development of open source FEM/DEM simulation tools* [on-line]. Toronto (Canadá): University of Toronto. Disponible en la Internet en: <http://mediacast.ic.utoronto.ca/20071107-LASS/msh.htm>. Consultado: 12-oct-2009.

27. Niezgoda, T.; Derewońko, A. (2009). "Multiscale composite FEM modeling". *Procedia Engineering* 1 (1) pp. 209-212.

28. Ou, H.; Armstrong, C.G. (2006). "Evaluating the effect of press and die elasticity in forging of aerofoil sections using finite element

simulation". *Finite Elements in Analysis and Design* 42 (10) pp. 856-867.

29. Otto, S.R.; Denier, J.P. (2005). *An introduction to programming and numerical methods in MATLAB*. Londres (UK): Springer-Verlag. ISBN-1852339195.
30. Öztorun, N.K. (2006). "A rectangular finite element formulation". *Finite Elements in Analysis and Design* 42 (12) pp. 1031-1052.
31. Palmonella, M.; Friswell, M.I.; Mottershead, J.E.; Lees, A.W. (2005). "Finite element models of spot welds in structural dynamics: review and updating". *Computers and Structures* 83 (8-9) pp 648-661.
32. Park, K.S.; Van Tyne, C.J.; Moon, Y.H. (2007). "Process analysis of multistage forging by using finite element method". *Journal of Materials Processing Technology*, 187-188 (12) pp. 586-590.
33. Pramanik, A.; Zhang, L.C.; Arsecularatne, J.A. (2007). "An FEM investigation into the behavior of metal matrix composites: Tool–particle interaction during orthogonal cutting". *International Journal of Machine Tools and Manufacture* 47 (10) pp.1497-1506.
34. Sadd, M.H. (2005). *Elasticity: theory, applications, and numerics*. Oxford (UK): Elsevier. ISBN 0-12-605811-3.
35. Saka, M.P. (2005). "The theorems of structural variation for rectangular finite elements for plate flexure". *Computers and Structures* 83 (28-30) pp. 2442-2452.

36. Shabana, A.A. (2008). *Computational continuum mechanics*. Cambridge (UK): Cambridge University Press. ISBN 978-0-511-38640-4.

37. Shi, D.; Mao, S.; Chen, S. (2007). "A locking-free anisotropic nonconforming finite element for planar linear elasticity problem". *Acta Mathematica Scientia* 27 (1) pp. 193-202.

38. Tezduyar, T.E.; Sathe, S. (2004). "Enhanced-discretization space–time technique (EDSTT)". *Computer Methods in Applied Mechanics and Engineering*, 193 (15-16) pp.1385-1401.

39. Troyani, N.; Pérez, A.; Baíz, P. (2005). "Effect of finite element mesh orientation on solution accuracy for torsional problems". Finite Elements in Analysis and Design 41 (14) pp. 1377-1383.

40. Umbrello, D. (2008). "Finite element simulation of conventional and high speed machining of Ti6Al4V alloy". Journal of Materials Processing Technology 196 (1-3) pp.79-87.

41. Vaz, M.; Muñoz-Rojas, P.A.; Filippini, G. (2009). "On the accuracy of nodal stress computation in plane elasticity using finite volumes and finite elements". Computers and Structures 87 (17-18) pp. 1044-1057.

42. Whiteley, J.P. (2009). "Discontinuous Galerkin finite element methods for incompressible non-linear elasticity". Computer Methods in Applied Mechanics and Engineering 198 (41-44) pp. 3464-3478.

43. Zienkiewicz, O.C.; Taylor, R.L. (2005). *The element method for solid and structural mechanics*. 6th Ed. Oxford (UK): Elsevier. ISBN 0-7506-6321-9.

44. Jilani, H.; Bahreininejad, A.; Ahmadi, M.T. (2009). "Adaptive finite element mesh triangulation using self-organizing neural networks". Advances in Engineering Software 40 (11) pp. 1097-1103.

# ANEXOS

## Anexo 1. Notación de las magnitudes del problema de elasticidad lineal.

Vector de desplazamientos:

$$\mathbf{u} = [u_1, u_2, u_3]^{\mathrm{T}} \equiv u_i$$

Vector de fuerzas distribuidas por unidad de volumen:

$$\mathbf{F} = [F_1, F_2, F_3]^{\mathrm{T}} \equiv F_i$$

Tensor de deformaciones infinitesimales de Green-Lagrange (segundo orden):

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_{11} & \varepsilon_{12} & \varepsilon_{13} \\ \varepsilon_{21} & \varepsilon_{22} & \varepsilon_{23} \\ \varepsilon_{31} & \varepsilon_{32} & \varepsilon_{33} \end{bmatrix} \equiv \varepsilon_{ij}$$

Segundo tensor de tensiones de Piola-Kichhoff (segundo orden):

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix} \equiv \sigma_{ij}$$

Tensor de elasticidad (cuarto orden):

$$\mathcal{C} \equiv c_{ijkl}$$

Vector de desplazamientos prescritos:

$$\hat{\mathbf{u}} = [\hat{u}_1, \hat{u}_2, \hat{u}_3]^{\mathrm{T}} \equiv \hat{u}_i$$

Vector de presiones superficiales prescritas:

$$\mathbf{t} = [t_1, t_2, t_3]^{\mathrm{T}} \equiv t_i$$

Operador gradiente sobre un vector;

$$\nabla(\bullet) = \begin{bmatrix} \partial(\bullet)_1 / \partial x_1 & \partial(\bullet)_1 / \partial x_2 & \partial(\bullet)_1 / \partial x_3 \\ \partial(\bullet)_2 / \partial x_1 & \partial(\bullet)_2 / \partial x_2 & \partial(\bullet)_2 / \partial x_3 \\ \partial(\bullet)_3 / \partial x_1 & \partial(\bullet)_3 / \partial x_2 & \partial(\bullet)_3 / \partial x_3 \end{bmatrix} \equiv \frac{\partial(\bullet)_i}{\partial x_j} \mathbf{e}_i \mathbf{e}_j$$

Operador divergencia sobre un vector:

$$\nabla \cdot (\bullet) = \frac{\partial(\bullet)_1}{\partial x_1} + \frac{\partial(\bullet)_2}{\partial x_2} + \frac{\partial(\bullet)_3}{\partial x_3} \equiv \frac{\partial(\bullet)_i}{\partial x_i}$$