UNIVERSIDAD NACIONAL DE CAJAMARCA

FACULTAD DE INGENIERIA

ESCUELA ACADEMICO PROFESIONAL DE INGENIERIA CIVIL

TESIS

ANALISIS ESTRUCTURAL POR EL METODO DE ELEMENTOS FINITOS ASISTIDO POR COMPUTADORA (VIGAS-PORTICOS, PLACAS, SOLIDOS DE REVOLUCION)

PARA OPTAR EL TÍTULO PROFESIONAL DE
INGENIERO CIVIL

PRESENTADO POR EL BACHILLER:
CHRISTIAN GONZALO SALCEDO MALAVER

ASESOR:
ING.MARCO MENDOZA LINARES

CAJAMARCA PERU-2014

SECCIÓN PRELIMINAR.

- **DEDICATORIA:**

Dedico esta tesis a mis padres ya que a ellos les debo la vida y mis estudios y lo que puedo ser más adelante.

■ AGRADECIMIENTOS:

* A mi madre a la cual le debo la vida, lo que soy y lo que pretendo ser, por que siempre estuvo en mis aciertos y mis fracasos y me enseño la constancia de la vida con su ejemplo.

* A mi padre al que le debo su ejemplo y su coraje para enfrentar los problemas,el que me demostró que los fracasos de la vida son solo retos para algo mas grande.

* A mis profesores que con su esfuerzo y buen ánimo me aconsejaron y me enseñaron en todos estos años.

▪ RESUMEN

La presente tesis trata en forma objetiva del análisis de estructuras con el método de elementos finitos, lo cual se empezará primero con el aprendizaje del método y después con la aplicación a diferentes sistemas tanto sencillos como complejos,para lo cual se desarrollará un software en python en el sistema windows para ejemplos teóricos y simples, el objetivo principal de la tesis es demostrar que los resultados obtenidos por el método de elementos finitos en este caso representado con el script programado llamado FEMAX son los más cercanos posibles a los resultados aceptados por la comunidad de ingeniería,ya que estos fueron comparados con el software aceptados en forma estandard como el SAP 2000 V14,en el ámbito comparativo se vio que los resultados obtenidos con el script programado en este caso por objetivos de la tesis FEMAX fueron satisfactorios obteniendo un parecido de 99.8 % por lo que se da como resultados aceptables y que se ha llegado a un caso satisfactorio de la presente tesis.

Índice general

Índice de figuras

Índice de cuadros

Planteamiento del Problema.

Capítulo 1

Problema de Investigación.

1.1. Ubicación.

La investigación por tener carácter teórico no se menciona el lugar de la investigación,pero por motivos de formalismos,se propondrá el sitio de estudios a la Universidad Nacional de Cajamarca.

1.2. Población.

No se menciona por el carácter teórico de la tesis.

1.3. Introducción y Planteamiento del Problema.

1.3.1. Planteamiento del Problema.

Aplicar el método de elementos finitos al campo del análisis estructural con la finalidad de diseñar un software para el cálculo de estructuras planas;así como el aprendizaje del método para sistemas estructurales como vigas-pórticos, placas y sólidos de revolución.

1.3.2. Formulación del Problema.

En que medida se ajusta el método de elementos finitos al estudio del análisis estructural,y por que los software comerciales trabajan con dicho método?

1.3.3. Elementos Continuos y Discretos.

Las limitaciones de la mente humana son tales que no pueden captar el comportamiento que el complejo mundo que lo rodea en una sola operación global.Por ello,una forma natural de proceder de ingenieros consiste en separar los sistemas en sus componentes individuales,o "Elementos",cuyo comportamiento pueden conocerse sin dificultad,y a continuación reconstruir el sistema original para estudiarlo a partir de dichos componentes.

En muchos casos se obtiene un modelo adecuado usando un numero finito de componentes bien definidos.A tales problemas los denominaremos discretos. En otros,las subdivisión prosigue indefinidamente y el problema solo puede definirse haciendo uso de la ficción matemática infinitesimal.

Ello nos conduce a ecuaciones diferenciales o expresiones equivalentes con un numero infinito de elementos implicados. a tales sistemas los llamaremos continuos.

Su campo de aplicación de este método es muy amplio,y en consecuencia es una herramienta importante en las nuevas formulas de analizar y simular fenómenos estructurales,así como también en el campo de la hidráulica y la geotecnia. [1, pag.4] [1]

1.3.4. Justificación.

- Los avances tecnológicos obligan el conocimiento cabal de los métodos más precisos para el cálculo en la ingeniería, donde el error cada día se vuelve

[1] [1]⟶Pérez Villar Luis Alberto,Tesis:"Análisis de Estructuras por el Método de Elementos Finitos A.Computadora"

mínimo por lo tanto es necesario el conocimiento analítico y computacional de teorías de cálculo más precisas y exactas.

- Difundir el uso del método de los elementos finitos.

1.3.5. Alcances.

Al ser una tesis de carácter teórico a lo que queremos llegar es primero, al conocimiento formal del método,así como demostrar la fiabilidad del método programado en la computadora y tratar de demostrar de por que los programas comerciales trabajan con el método de elementos finitos.

1.3.6. Hipótesis.

Se comprobará que el método de elementos finitos es un método confiable para el análisis y cálculo de elementos estructurales,por eso es que los software comerciales trabajan con dicho método.

1.3.7. Variables.

Las variables de la investigación son muchas y seria muy apresurado al decidir que variable es mas importante o cual seria las variables principales de la investigación por lo tanto este item queda para observarse en el marco teórico de la tesis.

1.3.8. Limitaciones.

- El programa femax de cálculo de elementos planos, tiene como limitación el cálculo de elementos como placas y sólidos de revolución las cuales serán resueltos en forma aparte por medio scripts generados en el lenguaje de programación python.

- el programa femax tiene como problema el sistema de instalación ya que su ejecución cuenta con librerías especiales la cual complica el sistema de instalado o de generación de un archivo.exe.

1.4. Breve Historia de los Elementos Finitos.

Los Conceptos de discretización numérica para resolver problemas de Ciencia e ingeniería son la base para la formulación del método de elemento finito.La aproximación geométrica mas antigua lleva a las pirámides egipcias de 5000 años.Por otro lado la aproximación numérica podría registrarse históricamente en China,Egipto y Grecia.

Los registros muestran que los chinos calcularon el valor aproximado de π en el primer siglo de nuestra era,con un valor de 3.1547 siendo usado para calcular el volumen de un cilindro.

En el segundo siglo E.C el astrónomo Chang Heng aproximo el valor de π como 3.1466 $\frac{142}{45}$,en la Dinastía oriental de Jihn(265-317 E.C) en su comentario de matemáticas uso un polígono regular inscrito en una circunferencia para poder aproximar π la cual hallo un valor de 3.1416 (3927/1250);es interesante notar que el uso un polígono de 3072 lados,es decir elementos finitos. De acuerdo con el manuscrito Ahmes, se muestra que para 1500 A.C. , los Egipcios usaban como valor de $\pi = 3{,}1416$. Un papiro de tiempos mas tempranos, ahora en Moscú, indica que los egipcios usaron la fórmula para el volumen de una pirámide y el área de un círculo de manera aproximada en 1800 A. C.. Arquímedes uso el concepto de elementos finitos para calcular volúmenes.
En el contexto estructural, las soluciones tanto en elasticidad como en análisis es-

tructural tuvieron un inicio del Método del Elemento Finito con Timoshenko, pero si se considera que el análisis de marcos establece el inicio del método del Elemento Finito, entonces los pioneros fueron Castigliano, Mhor y Maxwell, entre otros, en el periodo 1850-1875.

En 1915, Maney de los Estados Unidos de Norteamérica, presentó el método pendiente-deformación, expresando los momentos en términos de desplazamientos lineales y angulares en los nodos de la estructura, lo cual es una de las formulaciones para plantear el método de las rigideces y un desarrollo similar, fue planteado por Ostenfeld en Dinamarca. En el año 1929, Hardy Cross hizo público un método para analizar marcos basado en distribuciones angulares, el cual se utilizó por los siguientes 35 años.

En forma paralela a los primeros trabajos sobre análisis de estructuras reticulares, se resolvieron problemas de mecánica del medio continuo usando una analogía con estructuras formadas por barras diagonales para generar mallas con elementos triangulares. A principios de los años cuarenta Courant propuso funciones de interpolación polinomiales por secciones para formular sub regiones triangulares como un caso especial del método variacional de Rayleigh-Ritz, que obtiene soluciones aproximadas. Actualmente, el método del elemento finito es utilizado con la ayuda de las computadoras, lo cual ha contribuido a su desarrollo al mismo ritmo que las computadoras. Las publicaciones clásicas por Argyris y Kelsey a mediados de los 50-as , hicieron surgir los conceptos de análisis de marcos discretizando no solo en nodos sino además en puntos intermedios de las barras y análisis de un continuo, lo que marcó un crecimiento explosivo en el método del elemento finito.

Basándose en el planteamiento estático del elemento finito, se han ampliado las

aplicaciones que incluyen diversos efectos físicos y vibraciones en el Análisis dinámico, pandeo y post-pandeo, no linealidades en la geometría y en el material, efectos térmicos, interacción entre fluidos y estructuras, aero elasticidad, interacción acústica-estructura, teoría de la fractura, estructuras laminadas, propagación de oleaje, dinámica estructural, respuesta dinámica aleatoria, y muchas más aplicaciones. Como una consecuencia de tantos campos de estudio, el uso de los programas de computadora orientados a cada caso, se han convertido en una práctica en los sitios involucrados en el análisis estructural. [1, pag.5]

1.5. Aplicaciones del Método.

Existen gran número de estructuras que su paso de revisión tanto como post y pre proceso de diseño,son sometidos a rigurosos procesos de evaluación tales como para verificación de cortantes y flectores y saber si el diseño dado esta correcto.

1.5.1. Aplicaciones al Análisis Estructural.

En programas como el sap el staad pro tanto en el análisis de Int-estructura-estructura y análisis de suelo-estructura como el safe o el plaxis que son de uso para el cálculo por elementos finitos.

1.5.2. Aplicación en Mecánica de Fluidos.

El método de elementos finitos también puede ser usado en el análisis de mecánica de fluidos librerías en python como ECOASTER o el Fenics o programas como Abaqus,Nastram nos pueden dar una idea de como este elemento físico se mueve e interacciona con el mundo aplicando soluciones particulares a la formulación de de la ecuación general de los fluidos Navier y Stokes,idealizaciones de presas así como simulaciones de comportamiento de las turbulencias de los fluidos en turbinas hidráulicas.

1.5.3. Objetivos.

- **Objetivos Generales.**

  Análisis Estructural por el método de elementos finitos asistido por computadora(Vigas-Pórticos,Placas,Sólidos de Revolución).

- **Objetivos Específicos.**

  Crear un pequeño software en lenguaje de programación python,así como la comprobación de los resultados obtenidos con programas comerciales en este caso el sap 2000 v14.

MARCO TEÓRICO

Capítulo 2

MÉTODO DE ELEMENTOS FINITOS

Las recetas para deducir las características de un elemento finito de un continuo, serán presentadas bajo una forma matemática más detallada. Es conveniente obtener los resultados de una forma general aplicable a cualquier situación, pero para evitar la introducción de conceptos más complicados se ilustraran las expresiones generales con un ejemplo. [2, Pag.21]

$$u = Na \longrightarrow \sum_i N_i x a_i = [N_1, N_2 \dots N_n] \left\{ \begin{array}{c} a_1 \\ a_2 \\ \vdots \\ a_n \end{array} \right\} \tag{2.1}$$

Un elemento típico e, se define por sus nodos$i, j, m \cdots etc$ y por su contorno formado por líneas rectas. Aproximemos los desplazamientos u de cualquier punto del elemento mediante un vector u, como en la formula anterior. [2, Pag.22]
En el caso particular de tensión plana.

$$u = \left\{ \begin{array}{c} u(x,y) \\ v(x,y) \end{array} \right\} \tag{2.2}$$

representa los movimientos horizontales y verticales de un punto cualquiera del elemento.

$$a_i = \left\{ \begin{array}{c} u_i \\ v_i \end{array} \right\} \tag{2.3}$$

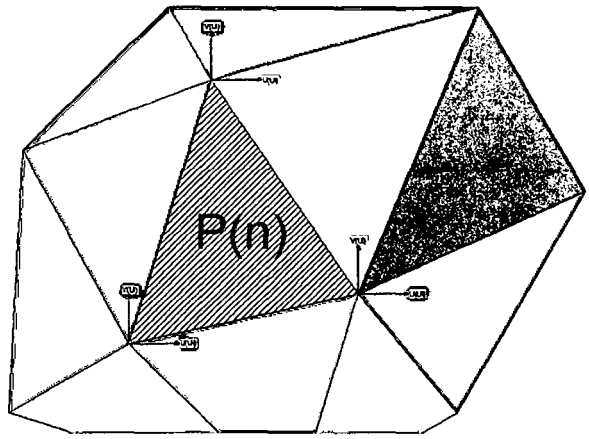


Figura 2.1: Análisis de un sistema continuo.

Los correspondientes desplazamientos de un nodo i.

Las funciones N_i, N_j, N_m han de escogerse de manera que al sustituir de la ecuación,las coordenadas de los nodos se obtengan los correspondientes desplazamientos nodales. [2, pag.24] [1]

2.1. Deformaciones.

Una vez conocidos los desplazamientos para todos los puntos del elemento,pueden determinarse las deformaciones en cualquier punto.Éstas darán siempre resultado en relación que podrá escribirse como sigue en forma matricial.

$$\epsilon = Su \tag{2.4}$$

$$\epsilon = Ba \tag{2.5}$$

$$B = SN \tag{2.6}$$

Donde S es un operador lineal apropiado en los casos de la tensión plana las deformaciones se expresan en función de los desplazamientos mediante las conocidas

[1] Bibliográfica [2]⟶ O.CZienkiewichz-RTaylor,Método de Elementos Finitos(Formulación Básica,Problemas Lineales).

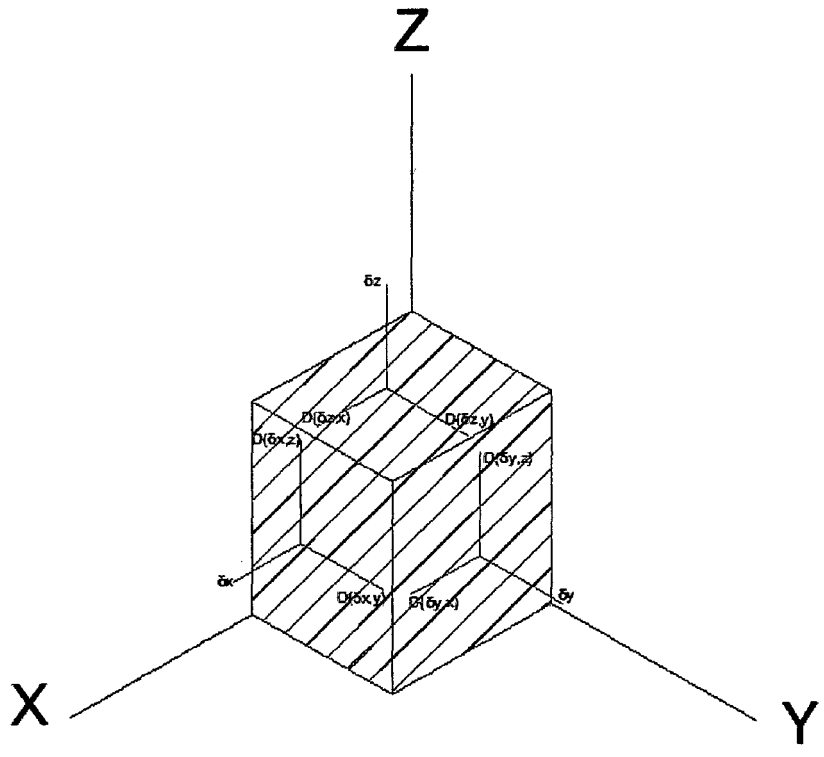


Figura 2.2: Estado general de Esfuerzos

relaciones que definen al operador S.

$$\epsilon = \left\{ \begin{array}{c} \epsilon_x \\ \epsilon_y \\ \epsilon_z \end{array} \right\} = \left[\begin{array}{cc} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{array} \right] \left\{ \begin{array}{c} u \\ v \end{array} \right\} \tag{2.7}$$

determinadas ya las funciones de forma es fácil obtener la matriz B.

2.2. Tensiones.

Conociendo el contorno del material o elemento puede estar sujeto a deformaciones iniciales,tales como las debidas a cambios de temperatura etc.Deberán diferenciarse entre Esfuerzos iniciales y Esfuerzos σ_0 que muy bien podrían medirse,pero

cuya predicción seria imposible sin un conocimiento completo de la historia del material.estas tensiones pueden sencillamente añadirse a las ecuaciones generales asumiendo un comportamiento elástico lineal. [3, Pag 27]

$$\sigma_i = D(\epsilon - \epsilon_0) + \sigma_0 \tag{2.8}$$

$$\sigma_i = \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} \tag{2.9}$$

$$\begin{Bmatrix} \epsilon_x \\ \epsilon_y \\ \tau_{xy} \end{Bmatrix} = \begin{Bmatrix} \frac{1}{E}\sigma_x - \frac{v}{E}\sigma_y \\ \frac{1}{E}\sigma_y - \frac{v}{E}\sigma_x \\ \frac{1}{G}\gamma_{xy} \end{Bmatrix} = \begin{bmatrix} \frac{1}{E} & -\frac{v}{E} & 0 \\ -\frac{v}{E} & \frac{1}{E} & 0 \\ 0 & 0 & \frac{1}{G} \end{bmatrix} \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{Bmatrix} \tag{2.10}$$

2.3. Fuerzas nodales equivalentes

Son fuerzas que están en la misma dirección de los desplazamientos de posición a_i correspondientemente y deben ordenarse en dirección apropiada.
Las fuerzas distribuidas b son por definición las que actúan por unidad de volumen en dirección correspondientes a las de desplazamientos u de ese punto. [2, Pag 30]

$$q_i = \begin{Bmatrix} q_1^e \\ q_2^e \\ \vdots \\ q_n^e \end{Bmatrix} \tag{2.11}$$

$$b_i = \begin{Bmatrix} b_x^e \\ \\ b_y^e \end{Bmatrix} \tag{2.12}$$

Para mejor entendimiento se debe dar desplazamientos arbitrarios o virtuales a los nodos para darle mas sentido físico. e igualando el trabajo exterior con el interior.

$$\int \int_V \int (\delta \epsilon_i \sigma) dV + \int_l (\delta u b) dV = \sum_i (\delta(u) X_i) \tag{2.13}$$

$$\int \int_V \int ([a^e]^T B^T D B a) dV) + \int_V ([a^e]^T (N_i^T b) dV)) = [a^e]^T q_i \tag{2.14}$$

$$K_i a_i + f_i = q_i \tag{2.15}$$

$$K_i = \int_V (B^T D B) dV \tag{2.16}$$

$$f_i = \int_V (N^T b) dV \tag{2.17}$$

[2]

Capítulo 3

ANÁLISIS DE VIGAS.

El análisis de vigas para el estudio de elementos finitos,se hace para entender en forma elemental las diferentes tipos de discretizaciones a elementos de simples a complejos,ya que para poder entender los análisis y cálculo de placas y sólidos de revolución es necesario entender los diferentes items del estudio de vigas tanto sometidas a fuerzas axiales como a flexiones que pueden ser tratadas por teorías muy conocidas como Bernully y Timoshenko sobre todo entendimiento de condiciones de contorno para pasar a estadios mas complejos. [2, Pag.22]

3.1. Vigas Sometidas a Fuerza Axial.

[1] Una vez deducida las ecuaciones principales tanto de Tensión como de deformación,rigideces y fuerzas nodales,aplicaremos esas formulas generales al caso de vigas sometidas a fuerza axial con la cual empezaríamos los casos de aplicación y análisis de FEM. [3, Pag.28]

$$\sigma_i = E\epsilon = E\frac{du}{dx} \tag{3.1}$$

con E que es el módulo de elasticidad de la barra;en la configuración de equilibrio de la barra,las tensiones las fuerzas exteriores satisfacen el **(Principio de Trabajos**

[1]Bibliografia 3⟶ H.PartII:Eugenio Oñate,Cálculo de Estructuras por el Método de Elementos finitos,MGGraw-Hill,(1998)

Virtuales) y en este caso en particular el principio quedaría expresado de la siguiente manera.

$$\int\int\int_V (\delta\epsilon\sigma)dV = \int_0^l (\delta ub)dx + \sum_{i=1}^{p}(\delta u_i X_i) \tag{3.2}$$

$$\int_0^l \delta\epsilon EA\frac{du}{dx} = \int_0^l \delta ubdx + \sum_{i=1}^{p}\delta u_i X_i \tag{3.3}$$

Bueno usando la aproximación de una deformación u_i la cual estaría aproximada a una ecuación lineal $u_i = u_0 + a_1 x$,como ya se vio en el capitulo anterior lo que tiene es hallar las funciones de forma. [3, Pag.28]

$$u_i = N_1 a_1 + N_2 a_2 \tag{3.4}$$

(3.5)

Analizando la generalización de las deformaciones de la ecuación lineal tendríamos.

$$u_1 = a_0 + a_1 x_1 \tag{3.6}$$

$$u_2 = a_0 + a_1 x_2 \tag{3.7}$$

Solucionando el problema de los casos tanto de a_0 y a_1 se propondría:

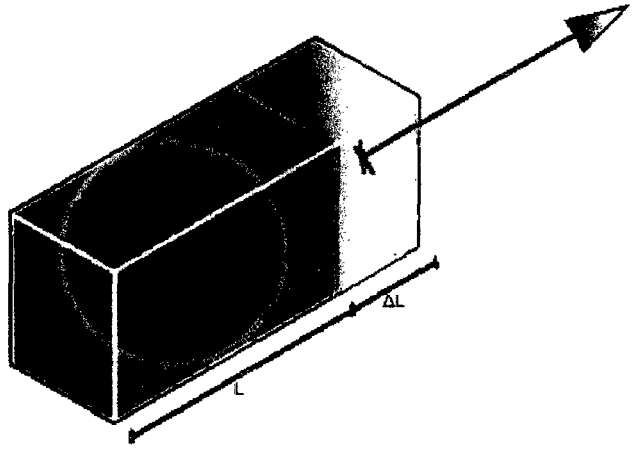


Figura 3.1: funciones de forma de viga C_0

$$a_0 = \frac{u_1 - u_2}{x_1 - x_2} \tag{3.8}$$

$$a_1 = \frac{x_2 u_1 - x_1 u_2}{x_2 x_1} \tag{3.9}$$

Para una mejor trabajabilidad de los dos casos hay que tomar en cuenta las secciones anteriores donde tendríamos $N_i ... N_m$ los cuales son las funciones de forma entonces sabiendo que $x_2 - x_1 = l_e$,se tendría que: [3, pag.28]

$$\int_{x_1}^{x_2} (\frac{dN_1}{dx}\delta u_1) AE (\frac{dN_2}{dx}\delta u_2) - \int_{x_1}^{x_2} [N_1 \delta u_1 + N_2 \delta u_2] = \delta u_1 X_1 + \delta u_2 X_2 \tag{3.10}$$

Agrupando términos tanto en δu_1 como $\delta_2 u_2$,se obtendría la formula siguiente:

$$\int_{x_1}^{x_2} (\frac{dN_1}{dx}(EA)\frac{dN_1}{dx}u_1 + \frac{dN_1}{dx}(EA)\frac{dN_2}{dx}u_2)dx - \int_{x_1}^{x_2} N_1 b dx - X_1 = 0 \tag{3.11}$$

$$\int_{x_1}^{x_2} (\frac{dN_2}{dx}(EA)\frac{dN_2}{dx}u_1 + \frac{dN_2}{dx}(EA)\frac{dN_1}{dx}u_2)dx - \int_{x_1}^{x_2} N_1 b dx - X_2 = 0 \tag{3.12}$$

Acomodando con un arreglo matricial:

$$\int_{x_1}^{x_2} \begin{bmatrix} (\frac{dN_1}{dx}(EA)\frac{dN_1}{dx}) & (\frac{dN_1}{dx}(EA)\frac{dN_2}{dx}) \\ (\frac{dN_2}{dx}(EA)\frac{dN_2}{dx}) & (\frac{dN_2}{dx}(EA)\frac{dN_1}{dx}) \end{bmatrix} dx \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} - \int_{x_1}^{x_2} \begin{Bmatrix} N_1 \\ N_2 \end{Bmatrix} b dx = \begin{Bmatrix} X_1 \\ X_2 \end{Bmatrix} \tag{3.13}$$

con la cual obtenemos que para este caso de vigas sometidas a tension tendríamos las siguientes formulas:

$$k_i a_i + f_i = q_i \tag{3.14}$$

$$K_i = \int_{x_1}^{x_2} \begin{bmatrix} (\frac{dN_1}{dx}(EA)\frac{dN_1}{dx}) & (\frac{dN_1}{dx}(EA)\frac{dN_2}{dx}) \\ (\frac{dN_2}{dx}(EA)\frac{dN_2}{dx}) & (\frac{dN_2}{dx}(EA)\frac{dN_1}{dx}) \end{bmatrix} dx = \int_{x_1}^{x_2} (B_i^T D B i) dx \tag{3.15}$$

$$f_i = \int_{x_1}^{x_2} \begin{Bmatrix} N_1 \\ N_2 \end{Bmatrix} b dx \tag{3.16}$$

En elementos finitos se hablan de coordenadas naturales,pues se analizan en espacio natural la mayoría de sus geometrías,llevando esta solución ah espacio natural. [3, Pag.34]

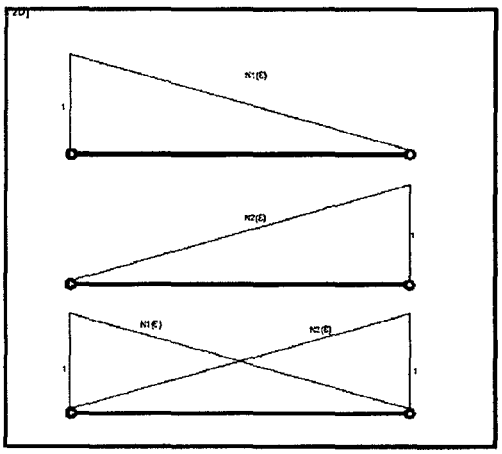


Figura 3.2: funciones de forma de viga C_0

Por interpolación del polinomio de lagrange se hace un análisis de interpretación geométrica.

$$N_i = \frac{(\xi - \xi_1)(\xi - \xi_2)\dots(\xi - \xi_n)}{(\xi_i - \xi_1)(\xi_i - \xi_2)\dots(\xi_i - \xi_n)} \tag{3.17}$$

$$N_i = \prod_{i=1}^{n}(\frac{\xi - \xi_j}{\xi_i - \xi_j}) \tag{3.18}$$

Resolviendo la formulación por el polinomio de lagrange para el caso de dos nodos quedaría de la siguiente manera.

$$N_1 = (\frac{1-\xi}{2}) \tag{3.19}$$

$$N_2 = (\frac{1+\xi}{2}) \tag{3.20}$$

Con esto ya tendríamos la posibilidad de calcular tanto el K_i y f_i de la cual obtendríamos los parámetros ya dados en la parte anterior.

$$x_2 - x_1 = l_e \tag{3.21}$$

$$\xi_s = \frac{2(x - x_1)}{(x_2 - x_1)} - 1 \tag{3.22}$$

$$\frac{d\xi}{dx} = \frac{2}{l_e} \tag{3.23}$$

Se pondría en acción de las funciones B_i y D_i con ello empezaríamos a formular la Matriz de Rigidez y de fuerza.

$$B_i = [\frac{\partial N_1}{\partial \xi} * \frac{2}{l_e}, \frac{\partial N_2}{\partial \xi} * \frac{2}{l_e}] \tag{3.24}$$

$$K_I = \int_{-1}^{1} (\begin{bmatrix} \frac{\partial N_1}{\partial \xi} * \frac{2}{l_e} \\ \\ \frac{\partial N_2}{\partial \xi} * \frac{2}{l_e} \end{bmatrix} (AE)[\frac{\partial N_1}{\partial \xi} * \frac{2}{l_e}, \frac{\partial N_2}{\partial \xi} * \frac{2}{l_e}])(\frac{l_e}{2}) d\xi \tag{3.25}$$

$$K_I = \frac{AE}{L} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \tag{3.26}$$

$$f^e = \frac{bl_e}{2} \begin{Bmatrix} 1 \\ 1 \end{Bmatrix} \tag{3.27}$$

Recordando esto seria la formula clásica de matriz de rigidez sometido a tensión pura.Pero en elementos finitos se puede aproximar más a la solución real del sistema apoyándonos en mas nodos con la cual tendríamos mas cercanía de la solución correcta del problema físico.Por eso resolveremos el mismo problema solo que esta vez lo haremos con 3 nodos. [3, Pag.35]

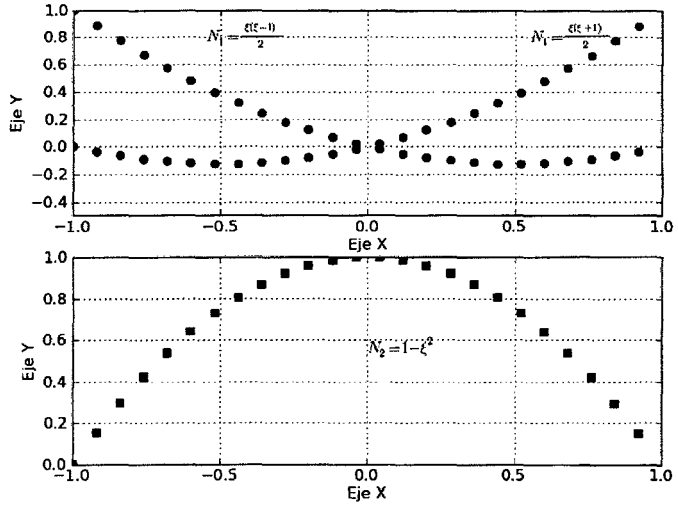


Figura 3.3: funciones del Sistema Natural de tres nodos.

Como vemos en la figura el elemento de tres nodos se analizaría de la siguiente manera:

$$L_{(2,1)} = \frac{(\xi - \xi_2)(\xi - \xi_3)}{(\xi_1 - \xi_2)(\xi_1 - \xi_3)} = \frac{\xi(\xi - 1)}{2} \tag{3.28}$$

$$L_{(2,2)} = \frac{(\xi - \xi_1)(\xi - \xi_3)}{(\xi_2 - \xi_1)(\xi_2 - \xi_3)} = \frac{\xi(\xi + 1)}{2} \tag{3.29}$$

$$L_{(2,3)} = \frac{(\xi - \xi_1)(\xi - \xi_2)}{(\xi_3 - \xi_1)(\xi_3 - \xi_2)} = (1 - \xi^2) \tag{3.30}$$

Ahora resolviendo el caso de ξ_i para el caso de solución del caso general de los tres nodos:

$$x = N_1 x_1 + N_2 x_2 + N_3 x_3 \tag{3.31}$$

$$\frac{dx}{d\xi} = \frac{dN_1}{d\xi} x_1 + \frac{dN_2}{d\xi} x_2 + \frac{dN_3}{d\xi} x_3 \tag{3.32}$$

$$\frac{dx}{d\xi} = \frac{2\xi - 1}{2} x_1 + \frac{2\xi + 1}{2} x_2 - 2\xi x_3 \tag{3.33}$$

$$\frac{d\xi}{dx} = \frac{2}{2\xi(x_1 + x_2 - 2x_3) + l_e} \tag{3.34}$$

$$\frac{d\xi}{dx} = \frac{2}{l_e} \tag{3.35}$$

Usando las Ecuaciones EC(2.16) y EC(2.17) de Matriz de Rigidez para poder solucionar la aproximación en el caso de tres nodos la cual quedaría de la siguiente manera:

$$\int_{-1}^{1} \left(\begin{Bmatrix} (\xi - \frac{1}{2})\frac{2}{l_e} \\ (-2\xi)\frac{2}{l_e} \\ (\xi + \frac{1}{2})(\frac{2}{l_e}) \end{Bmatrix} AE\{(\xi - \frac{1}{2})\frac{2}{l_e}, (-2\xi)\frac{2}{l_e}, (\xi + \frac{1}{2})(\frac{2}{l_e})\}(\frac{l_e}{2}) \right) d\xi \tag{3.36}$$

Como estamos asumiendo que el elemento es isotrópico y es un sistema Elástico

lineal donde A,E,son constantes Resolviendo el sistema anterior. [3, Pag.45]

$$K_e = (\frac{EA}{6}) \begin{bmatrix} 14 & -16 & 2 \\ -16 & 32 & -16 \\ 2 & -16 & 14 \end{bmatrix} \tag{3.37}$$

$$f_i = \int_{-1}^{1} \begin{pmatrix} \frac{\xi(\xi-1)}{2} \\ 1-\xi^2 \\ \frac{\xi(\xi+1)}{2} \end{pmatrix} b\frac{l_e}{2} d\xi \tag{3.38}$$

$$f_i = \begin{pmatrix} 1 \\ 4 \\ 1 \end{pmatrix} b\frac{l_e}{6} \tag{3.39}$$

3.2. Análisis de Flexión de Vigas.

EL clásico problemas de vigas puede ser resuelto con el método tradicionales de Resistencia de materiales sin embargo resolver el problema con el método sofisticado del MEF es de gran interés didáctico pues en este particular problema cada nodo puede ser trabajado con dos variables y puede servir como conceptos primarios para estudio de placas y láminas.

Entre estos principios y conceptos bsicos existen dos formas planteadas como estudio generalizado de las mismas como son el Estudio de vigas por el método Viga Bernoulli y el estudio de vigas por el método Viga Timoshenko. [3, Pag.101]

3.2.1. Teoría de Flexión de Viga de Euler y Bernoulli.

Consideremos una viga de longitud L,sección transversal de área A y modulo de inercia I sobre la cual actúan una serie de cargas verticales(flechas) y momentos contenidos en el plano XZ.

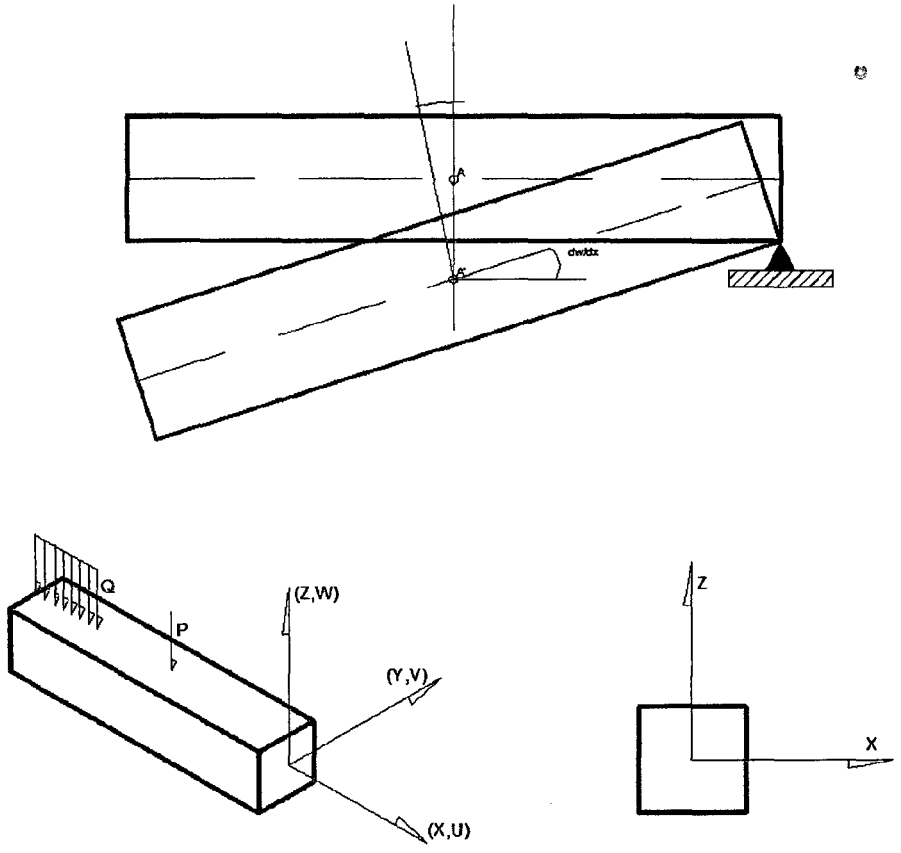


Figura 3.4: Viga Convencional Euler-Bernulli

$$\theta = \frac{dw}{dx} \tag{3.40}$$

$$\epsilon = \theta y = y\frac{d^2w}{dx^2} \tag{3.41}$$

$$\sigma = Ey\frac{d^2w}{dx^2} \tag{3.42}$$

$$dF_i = (Ey\frac{d^2w}{dx^2})dA \tag{3.43}$$

$$dM_i = (E\frac{d^2w}{dx^2})y^2dA \tag{3.44}$$

$$M_i = EI\frac{d^2w}{dx^2} \tag{3.45}$$

3.2.2. Discretización de elementos finitos de dos nodos.

La Incógnita fundamental del problema es la flecha w.No obstante debido a que en la expresión del trabajo virtual interno aparecen segundas derivadas de w, se deben usar elementos continuos de clase C_1(la variable y su primera derivada han de ser continuas) para evitar singularidades en el cálculo de las integrales.Esta condición se puede interpretar físicamente de manera sencilla teniendo en cuenta $\frac{dw}{dx}$,coincide con la pendiente de la deformada del eje de la viga.Por tanto,dicha derivada debe ser continua para garantizar que la deformada del eje de la viga,de tal manera dicha derivada debe ser continua para garantizar que la deformada del eje describa una curva suave.

El elemento mas sencillo de la viga clase C_1 es el unidimensional de dos nodos la cual estaría dada de la siguiente manera: [3, Pag.104]

$$w = \alpha_0 + \alpha_1\xi + \alpha_2\xi^2 + \alpha_3\xi^3 \tag{3.46}$$

$$w^{'} = \alpha_1 + 2\alpha_2\xi + 3\alpha_3\xi^2 \tag{3.47}$$

Por las condiciones de contorno la ecuación de deformación quedaría sintetizada de la siguiente manera:

$$w = N_1w_1 + \dot{N}_1(\frac{dw}{dx})_1 + N_2w_2 + \dot{N}_2(\frac{dw}{dx})_2 \tag{3.48}$$

Analizaremos para cada caso las Condiciones frontera con la cual sacamos el siguiente cuadro comparativo :

H_1	$\dot{H}_1$	H_2	$\dot{H}_2$	H_3	$\dot{H}_3$	H_4	$\dot{H}_4$
1	0	0	1	0	0	0	0
0	0	0	0	1	0	0	1

Cuadro 3.1: Valores por cada caso de discretización de la Barra.

Empezaremos a resolver las ecuaciones formadas por las condiciones de contorno.

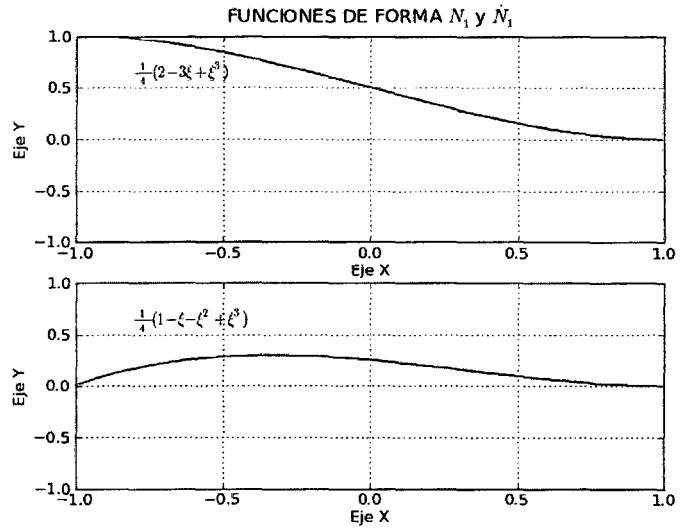


Figura 3.5: Funciones de forma N_1 y $\overline{N_1}$

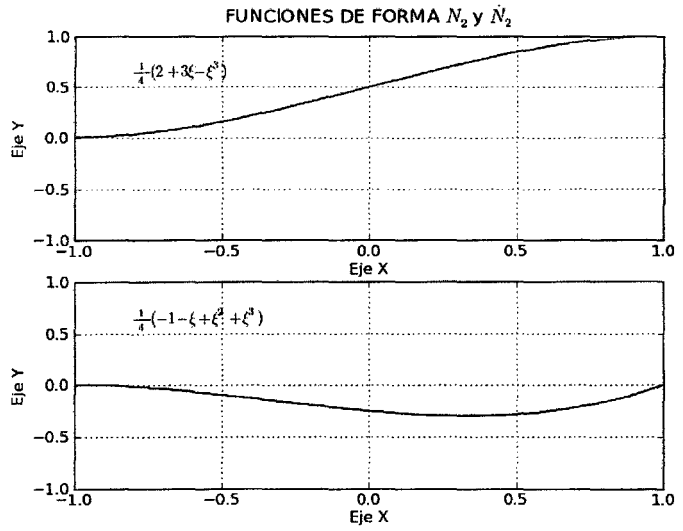


Figura 3.6: Funciones de forma N_2 y $\overline{N_2}$

$$\begin{bmatrix} 1 & -1 & 1 & -1 \\ 0 & 1 & -2 & 3 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \end{bmatrix} \begin{Bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{Bmatrix} \tag{3.49}$$

$$\begin{bmatrix} 1 & -1 & 1 & -1 \\ 0 & 1 & -2 & 3 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \end{bmatrix} \begin{Bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{Bmatrix} \tag{3.50}$$

$$\begin{bmatrix} 1 & -1 & 1 & -1 \\ 0 & 1 & -2 & 3 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \end{bmatrix} \begin{Bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{Bmatrix} \tag{3.51}$$

$$\begin{bmatrix} 1 & -1 & 1 & -1 \\ 0 & 1 & -2 & 3 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \end{bmatrix} \begin{Bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{Bmatrix} \tag{3.52}$$

Resolviendo las ecuaciones matriciales anteriores se obtendrán los coeficientes de la formula general

$H_i = \alpha_0 + \alpha_1\xi + \alpha_2\xi^2 + \alpha_3\xi^3$

con la cual tendríamos los coeficientes para cada caso de las condiciones de Contorno.

$$\begin{Bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{Bmatrix} = \begin{Bmatrix} \frac{1}{2} \\ -\frac{3}{4} \\ 0 \\ \frac{1}{4} \end{Bmatrix} \tag{3.53}$$

$$\begin{Bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{Bmatrix} = \begin{Bmatrix} \frac{1}{4} \\ -\frac{1}{4} \\ -\frac{1}{4} \\ \frac{1}{4} \end{Bmatrix} \tag{3.54}$$

$$\begin{Bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{Bmatrix} = \begin{Bmatrix} \frac{1}{2} \\ \frac{3}{4} \\ 0 \\ -\frac{1}{4} \end{Bmatrix} \tag{3.55}$$

$$\begin{Bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{Bmatrix} = \begin{Bmatrix} -\frac{1}{4} \\ -\frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \end{Bmatrix} \tag{3.56}$$

Con esto obtendríamos las funciones de forma para el caso de flexión de **Viga Bernoulli**.

$$N_1 = \frac{1}{4}(2 - 3\xi + \xi^3) \tag{3.57}$$

$$\overline{N_1} = \frac{1}{4}(1 - \xi - \xi^2 + \xi^3) \tag{3.58}$$

$$N_2 = \frac{1}{4}(2 + 3\xi - \xi^3) \tag{3.59}$$

$$\overline{N_2} = \frac{1}{4}(-1 - \xi + \xi^2 + \xi^3) \tag{3.60}$$

3.2.3. Matriz de Rigidez Viga Bernoulli.

Con todos los datos anteriores empezaremos con la propuesta de encontrar la Matriz de rigidez para el caso de viga Bernoulli, aplicaremos la formula $\int_V B_i^T D B_i dv$ con la cual veremos las particularidades flectores y complementaríamos con el sistema Axial de fuerzas para completar la matriz de rigidez de una viga.

Para ello haremos uso de la discretización de dos nodos para los ejes x en la cual

estableceríamos lo siguiente: [3, Pag.103,104]

$$x = \frac{1-\xi}{2}x_1 + \frac{1+\xi}{2}x_2 \tag{3.61}$$

Empezaremos a resolver el paso de un sistema natural al cartesiano de la siguiente manera:

$$\frac{dx}{d\xi} = \frac{l_e}{2} \tag{3.62}$$

Reemplazando en la ecuación(2.16)para lo cual ya hallamos las funciones de forma y el traslado de un sistema por medio del Jacobiano anterior.

$$K_i = \int_{-1}^{1} \frac{4}{l_e^2} \begin{Bmatrix} \frac{dN_1}{d\xi} \\ \frac{d\overline{N_1}}{d\xi} \\ \frac{dN_2}{d\xi} \\ \frac{d\overline{N_2}}{d\xi} \end{Bmatrix} EI\{\frac{dN_1}{d\xi}, \frac{d\overline{N_1}}{d\xi}, \frac{dN_2}{d\xi}, \frac{d\overline{N_2}}{d\xi}\}\frac{l_e}{2}d\xi \tag{3.63}$$

$$K_i = \frac{EI}{l_e^2}\begin{bmatrix} 12 & 6l_e & -12 & 6l_e \\ 6l_e & 4l_e^2 & -6l_e & 2l_e^2 \\ -12 & -6l_e & 12 & -6l_e \\ 6l_e & 2l_e^2 & -6l_e & 4l_e^2 \end{bmatrix} \tag{3.64}$$

Como se pueden ver es la matriz de rigidez aprendida en el curso de Análisis Estructural ;estaríamos estableciendo ya en este momento la matriz de $K_{(6x6)}$ para lo cual nos agenciaríamos de la Ec(3.26)y también la ecuación EC(3.64).[3, Pag.107]

$$K_T = \begin{bmatrix} \frac{AE}{l_e} & 0 & 0 & -\frac{AE}{l_e} & 0 & 0 \\ 0 & 12\frac{EI}{l_e^3} & 6\frac{EI}{l_e^2} & 0 & -12\frac{EI}{l_e^3} & 6\frac{EI}{l_e^2} \\ 0 & 6\frac{EI}{l_e^2} & 4\frac{EI}{l_e} & 0 & -6\frac{EI}{l_e^2} & 2\frac{EI}{l_e} \\ -\frac{AE}{l_e} & 0 & 0 & \frac{AE}{l_e} & 0 & 0 \\ 0 & -12\frac{EI}{l_e^3} & -6\frac{EI}{l_e^2} & 0 & 12\frac{EI}{l_e^3} & -6\frac{EI}{l_e^2} \\ 0 & 6\frac{EI}{l_e^2} & 2\frac{EI}{l_e} & 0 & -6\frac{EI}{l_e^2} & 4\frac{EI}{l_e} \end{bmatrix} \tag{3.65}$$

Para hallar las fuerzas en las vigas en forma general usaremos la ecuación general de $K_i a_i + f^e = q$ la cual f^e es igual a $\int_v N_i^T q dv$ que en este caso seria de la siguiente manera: [3, Pag.107]

$$\int_{-1}^{1} \begin{Bmatrix} N_1 \\ \overline{N_1} \\ N_2 \\ \overline{N_2} \end{Bmatrix} q dx = \begin{Bmatrix} -\frac{ql}{2} \\ -\frac{ql^2}{12} \\ -\frac{ql}{2} \\ \frac{ql^2}{12} \end{Bmatrix} = \begin{Bmatrix} V_1 \\ M_1 \\ V_2 \\ M_2 \end{Bmatrix} \tag{3.66}$$

3.2.4. Teoría de Flexión de Viga de Timoshenko

La teoría de vigas de Timoshenko comparte hipótesis de la teoría clásica.Por contrapartida,la nueva hipótesis establece que **Las secciones planas normales al eje de viga antes de la deformación,permanecen planas pero no necesariamente normales al eje después de la deformación**.

Esta hipótesis representa una mayor aproximación a la deformación real de la sección

transversal en vigas de gran canto.A medida que la relación longitud/canto disminuye,las secciones transversales dejan de conservarse planas después de la deformación. [3, Pag.119]

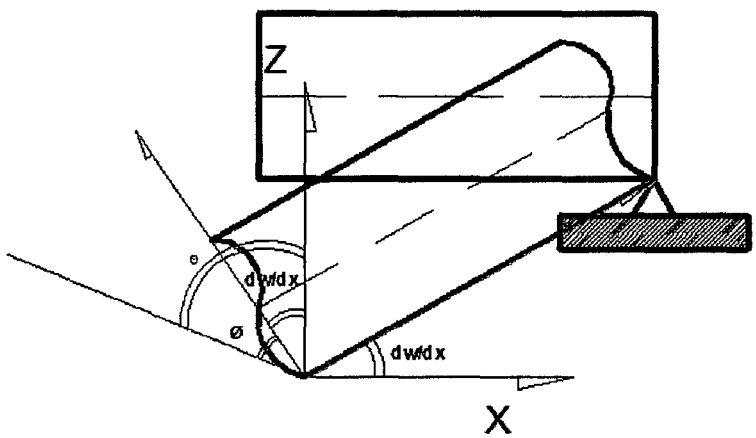


Figura 3.7: Viga con giro Adicional ϕ

$$\theta = \phi + \frac{dw}{dx} \tag{3.67}$$

$$\epsilon_x = \frac{du}{dx} = -z\frac{d\theta}{dx} \tag{3.68}$$

$$\gamma_{xz} = \frac{dw}{dx} + \frac{du}{dz} = \frac{dw}{dx} - \theta = -\phi \tag{3.69}$$

Por consiguiente la teoría de Timoshenko equivale a considerar el efecto de la deformación por cortante transversal,coincidiendo la magnitud de dicha deformación adicional de la norma ϕ los dos esfuerzos tanto de σ_x y σ_{xz} se relacionan con las correspondientes deformaciones por: [3, Pag.121]

$$\sigma_x = E\epsilon_x = -zE\frac{d\theta}{dx} \tag{3.70}$$

$$\sigma_{xz} = G(\frac{dw}{dx} - \theta)M_i = EI\frac{d\theta}{dx} \tag{3.71}$$

$$Q_i = GA(\frac{dw}{dx} - \theta) = GA\gamma_{xz} \tag{3.72}$$

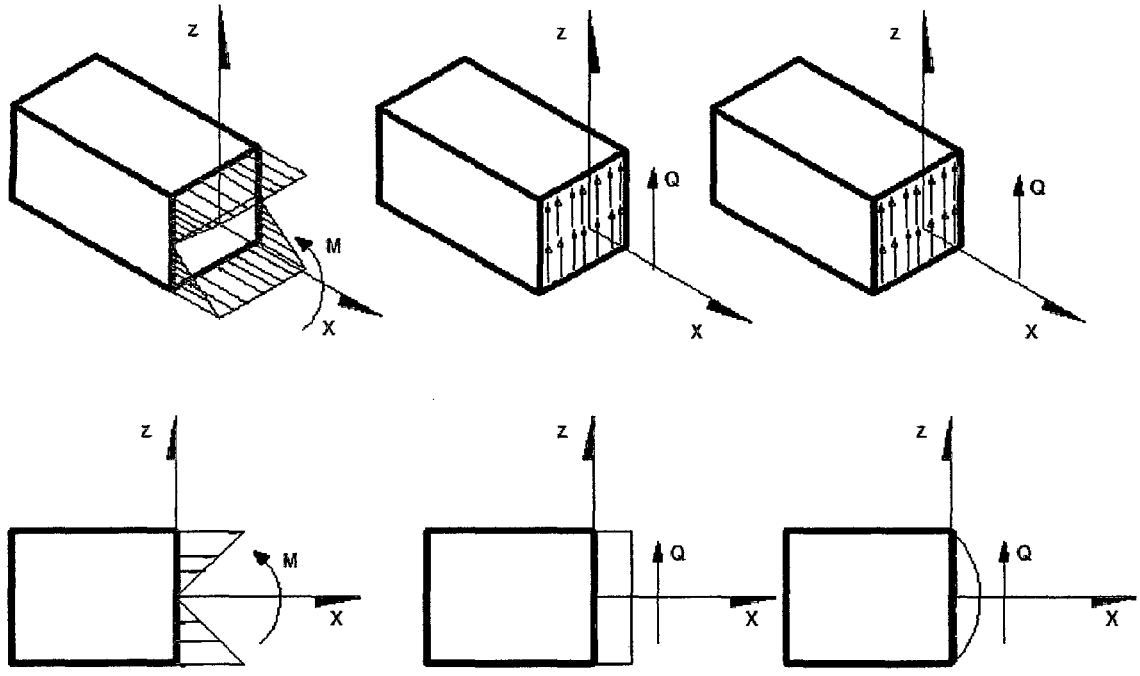


Figura 3.8: Análisis de Vigas tanto en Momento como en Cortante.

Ahora aplicando el principio de trabajo virtuales pero con la condición de hacerlo lineal o constante al elemento cortante de la Viga Timosheko $\tau_{xz} = \alpha G \gamma_{xz}$,donde α es la cortante de forma o de distorsión $A^* = \alpha A$.

$$\int \int_V \int (\delta \epsilon_x + \delta \gamma_{xz} \tau_{xz}) dV = - \int_l \delta q dx + \sum_{i=1}^{n} \delta (\frac{dw}{dx})_i M_i + \sum_{i=1}^{n} \delta w_i Z_i \qquad (3.73)$$

Ahora simplificando la expresión con los datos anteriores deducidos anteriormente quedaría de la siguiente manera: [3, Pag.123]

$$\int_l [\delta (\frac{d\theta}{dx}) EI (\frac{d\theta}{dx}) + \delta (\frac{dw}{dx} - \theta) GA^* (\frac{dw}{dx})] dx \qquad (3.74)$$

3.2.5. Elementos finitos para flexión de vigas de Timoshenko.

La flecha de la viga Timoshenko esta en función en este caso de θ y w variables independientes de continuidad C_0 por lo tanto se pueden interpolar por separado

cada una de ellas por:

$$w(\xi) = N_1(\xi)w_1 + N_2(\xi)w_2 \tag{3.75}$$

$$\theta(\xi) = N_1(\xi)\theta_1 + N_2(\xi)\theta_2 \tag{3.76}$$

Donde w_1,θ_1 y w_2,θ_2 son flechas y giros de los nodos 1 y 2 del elemento.

$$\chi = \frac{d\theta}{dx} = (\frac{\delta N_1(\xi)}{\delta\xi}\theta_1 + \frac{\delta N_2(\xi)}{\delta\xi}\theta_2)\frac{d\xi}{dx} \tag{3.77}$$

$$\gamma_{xy} = \frac{dw}{dx} - \theta = \frac{d\theta}{dx} = (\frac{\delta N_1(\xi)}{\delta\xi}w_1 + \frac{\delta N_2(\xi)}{\delta\xi}w_2)\frac{d\xi}{dx} - N_1\theta_1 - N_2\theta_2 \tag{3.78}$$

Utilizando una formulación isoparametrica idéntica a la empleada para el elemento barra de dos nodos se obtiene que $\frac{d\xi}{dx} = \frac{2}{l_e}$ las ecuaciones Ec.3.79 y Ec.3.78 pueden ser escritas de la siguiente manera:

$$\chi = B_f a \tag{3.79}$$

$$\gamma_{xz} = B_c a \tag{3.80}$$

$$B_f = [0, \frac{2}{l_e}\frac{\delta N_1}{\delta\xi}, 0, \frac{2}{l_e}\frac{\delta N_2}{\delta\xi}] \tag{3.81}$$

$$B_c = [\frac{2}{l_e}\frac{\delta N_1}{\delta\xi}, -N_1, \frac{2}{l_e}\frac{\delta N_2}{\delta\xi}, -N_2] \tag{3.82}$$

$$a_i = [w_1, \theta_1, w_2, \theta_2] \tag{3.83}$$

Resolviendo las derivadas de los vectores se tendría:

$$B_f = [0, -\frac{1}{l_e}, 0, \frac{1}{l_e}] \tag{3.84}$$

$$B_c = [-\frac{1}{l_e}, \frac{\xi - 1}{2}, \frac{1}{l_e}, \frac{-\xi - 1}{2}] \tag{3.85}$$

Con las formulas anteriores calcularemos la Matriz de Rigidez general de la Viga según teoría Timoshenko. para eso volveremos a resolver o simplificar la ecuación Ec(3.74) que según las formulas deducidas en las ecuaciones 3.79 y 3.80 reemplazando en la formula 2.13 o el Principio de Trabajo Virtual.

$$[\delta a^e]^T \int_{l^e} [B_f^T(EI)B_f + B_c^T(GA)B_c]dx(a^e) = -[\delta a^e]^T \int_{l_e} \overline{N}^T(q)dx + [\delta a^e]^T q_e \tag{3.86}$$

tras Simplificación de la formula queda de la siguiente manera :

$$[K_f^e + K_c^e]a^e - f^e = q_e \tag{3.87}$$

Deduciendo de la Primera parte de la formula anterior quedaría:

$$K_f^e = \int_{l_e} B_f^T (EI) B_f dx \tag{3.88}$$

$$K_c^e = \int_{l_e} B_c^T (GA^*) B_c dx \tag{3.89}$$

El vector de fuerzas nodales equivalentes debidas a las cargas repartidas q; y

$$f^e = -\int_{l_e} \overline{N}^T q dx \tag{3.90}$$

$$\overline{N} = [N_1, 0, N_2, 0] \tag{3.91}$$

El vector de Fuerzas nodales de equilibrio que permite ensamblar las contribuciones de los distintos elementos de la matriz de rigidez y en el vector de las fuerzas globales.

Todas las integrales anteriores pueden transformarse sobre el dominio normalizado del elemento.Así,teniendo en cuenta que $dx = \frac{l_e}{2} d\xi$,las ecuaciones Ec(3.88,3.89,3.90) se escribirían como: [3, Pag 124-126]

$$K_f = \int_{-1}^{1} B_f^T (EI) B_f \frac{l_e}{2} d\xi \tag{3.92}$$

$$K_g = \int_{-1}^{1} B_g^T (EI) B_g \frac{l_e}{2} d\xi \tag{3.93}$$

$$f^e = -\int_{-1}^{1} \overline{N}^T \frac{l_e}{2} d\xi \tag{3.94}$$

3.2.6. Matriz de Rigidez Total V.Timoshenko y Efecto de Bloqueo.

Como en las anteriores ecuaciones se deduce que solo debe existir un solo punto de integración,ya que todos los términos del integrando de K_f exige un solo punto de integración,ya que todos los términos del integrando son contantes así pues tras

operar las operaciones obtenemos.

$$K_f = (\frac{EI}{l_e}) \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \tag{3.95}$$

Por otra parte la integración exacta de la matriz de rigidez de cortante precisa dos puntos de integración por parecer en el integrando K_c términos de segundo grado ξ ,obteniéndose:

$$K_c = (\frac{GA^*}{l}) \begin{bmatrix} 1 & \frac{l_e}{2} & -1 & \frac{l_e}{2} \\ \frac{l_e}{2} & \frac{l_e^2}{3} & -\frac{l_e}{2} & \frac{l_e}{2} \\ -1 & -\frac{l_e}{2} & 1 & -\frac{l_e}{2} \\ \frac{l_e}{2} & \frac{l_e}{2} & -\frac{l_e}{2} & \frac{l_e^2}{3} \end{bmatrix} \tag{3.96}$$

Ahora sumamos las matrices y obtendremos una matriz general para lo cual la matriz quedará de la siguiente manera y hallando a la vez un sistema de voladizo y resolviendo con el sistema de V.Timoshenko. [3, Pag]

$$\begin{bmatrix} \frac{GA^*}{l} & \frac{GA^*}{2} & -\frac{GA^*}{l} & \frac{GA^*}{2} \\ \frac{GA^*}{2} & (\frac{GA^*}{3}l + \frac{EI}{l}) & -\frac{GA^*}{2} & (\frac{GA^*}{3}l - \frac{EI}{l}) \\ -\frac{GA^*}{l} & -\frac{GA^*}{2} & \frac{GA^*}{l} & -\frac{GA^*}{2} \\ \frac{GA^*}{2} & (\frac{GA^*}{3}l - \frac{EI}{l}) & -\frac{GA^*}{2} & (\frac{GA^*}{3}l + \frac{EI}{l}) \end{bmatrix} \begin{Bmatrix} w_1 \\ \theta_1 \\ w_2 \\ \theta_2 \end{Bmatrix} = \begin{Bmatrix} V_1 \\ M_1 \\ P \\ 0 \end{Bmatrix} \tag{3.97}$$

Por las condiciones de contorno tendríamos que $w_1 = 0$ y que $\theta_1 = 0$ con ello analizaríamos las soluciones para la deformación w_2 para su respectiva comparación con el sistema V.Bernoulli.

$$\begin{bmatrix} \frac{GA^*}{l} & -\frac{GA^*}{2} \\ -\frac{GA^*}{2} & (\frac{GA^*}{3}l + \frac{EI}{l}) \end{bmatrix} \begin{Bmatrix} w_2 \\ \theta_2 \end{Bmatrix} = \begin{Bmatrix} P \\ 0 \end{Bmatrix} \tag{3.98}$$

Resolviendo y desarrollando la matriz el problema quedaría de la siguiente manera:

$$\begin{Bmatrix} w_2 \\ \theta_2 \end{Bmatrix} = \frac{\gamma}{\gamma+1} \begin{bmatrix} (\frac{GA^*}{3}l + \frac{EI}{l}) & \frac{l^2}{EI} \\ \frac{l^2}{EI} & \frac{l}{EI} \end{bmatrix} \begin{Bmatrix} P \\ 0 \end{Bmatrix} \tag{3.99}$$

donde $\gamma = \frac{12EI}{GA^*l^2}$ la flecha o la deformación vertical estaría dada por:

$$w_2 = \frac{\gamma}{\gamma+1}(\frac{l}{GA^*} + \frac{l^3}{3EI})Pw_2^{'} = \frac{l^3}{3EI}P \tag{3.100}$$

Analizando el cociente entre la solución sin cortante y la solución con cortante, quedaría de la siguiente manera:

$$\varphi = \frac{w_2}{w_2^{'}} = \frac{3(4\lambda^2+3)}{4\lambda^2(\lambda^2+3)} \tag{3.101}$$

Lógicamente el valor de φ debería tender a la unidad a medida que λ o esbeltez aumente pero vemos que según lo que hemos sacado es totalmente incorrecto eso quiere decir que el ***que la viga de Timoshenko de dos nodos es incapaz de reproducir en el límite la teoría clásica de vigas*** así a medida que la longitud aumenta se produce un fenómeno de sobre rigidez curiosamente cuando llega a a tener mayor importancia hasta llegar ah bloquear la solución. [3] Para tratar de llegar a una exactitud lógica es visto que se tiene que sub-evaluar es decir integrarlo solo en un solo

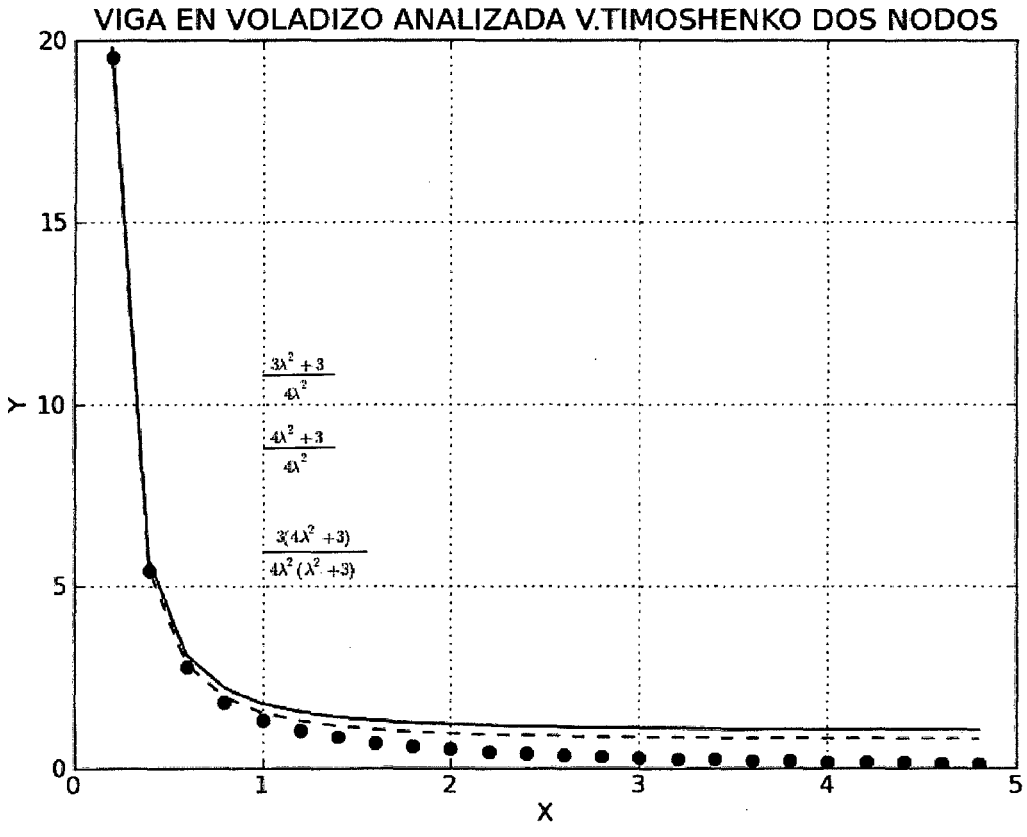


Figura 3.9: Análisis de Viga en Voladizo V.Timoshenko.

punto eliminando $\xi = 0$ y así tener una solución que no se distorsione en los limites.

$$B_c = [\frac{-1}{l_e}, \frac{1}{2}, \frac{1}{l_e}, -\frac{1}{2}] \tag{3.102}$$

$$\int_{-1}^{1} \left(\begin{bmatrix} \frac{-1}{l_e} \\ \\ \frac{1}{2} \\ \\ \frac{1}{l_e} \\ \\ \frac{1}{2} \end{bmatrix} GA^*[\frac{-1}{l_e}, \frac{1}{2}, \frac{1}{l_e}, -\frac{1}{2}] \right) \frac{l_e}{2} d\xi \tag{3.103}$$

de la cual la matriz de rigidez K_c quedaría de la siguiente manera es decir la matriz de integración en un solo punto: [3]

$$K_c = (\frac{GA^*}{l}) \begin{bmatrix} 1 & \frac{l_e}{2} & -1 & \frac{l_e}{2} \\ \frac{l_e}{2} & \frac{l_e^2}{4} & -\frac{l_e}{2} & \frac{l_e^2}{4} \\ -1 & \frac{-l_e}{2} & 1 & \frac{-l_e}{2} \\ \frac{l_e}{2} & \frac{l_e^2}{4} & -\frac{l_e}{2} & \frac{l_e^2}{4} \end{bmatrix} \tag{3.104}$$

Resolviendo igual que en el sistema anterior quedaría de la siguiente manera pues la matrices de Rigidez y Flexibilidad después de eliminar los grados de libertad del empotramiento.

$$K = \begin{bmatrix} \frac{GA^*}{l} & -\frac{GA^*}{l} \\ -\frac{GA^*}{l} & (\frac{GA^*}{4}l + \frac{EI}{l}) \end{bmatrix} \tag{3.105}$$

$$F = \begin{bmatrix} (\frac{1}{GA} + \frac{l^3}{4EI}) & -\frac{l^2}{2EI} \\ -\frac{l^2}{2EI} & (\frac{l}{EI} \end{bmatrix} \tag{3.106}$$

La relación entre este valor y el exacto para las vigas esbeltas es:

$$\varphi = \frac{w2}{(w2)^e_{exacta}} = \frac{3\lambda^2 + 3}{4\lambda^2} \tag{3.107}$$

La variación de la nueva función φ con λ se ha representado en la figura Fig3.9 vemos como la solución anterior si se aproxima a la real ya que la otra solución de las ecuaciones anteriores al $\lambda \longrightarrow 0$ es ∞ pero con este sistema subevaluado $\lambda \longrightarrow \infty$ es igual a 0.75.

Pero analizando la anterior solución vemos que existe un bloqueo ya que al acercarse a 0 esto tiende al infinito.

3.3. Programación para Vigas hecho en Python.

Deberíamos empezar esta sección con la pregunta del rigor por ¿que en python?, conociendo otro paquetes de programación como el Visual Basic,Mathcad,Matlab y otros, es que si tomamos en cuentas todos los lenguajes anteriores son privativos es decir si no compras las licencias correspondientes tu script o pequeño programa no sirve y como funcionan como pre compilación pues las licencias y todo eso se vuelven prácticamente insostenibles se esta optando por una programación de software libres pues esto va acompañado a la lógica y a la moralidad de compartir pues en lo personal me parece maligno y egoísta guardar el conocimiento pues es un derecho universal de la educación mundial en ese sentido quiero empezar este bloque diciendo que el mundo necesita una lógica y argumentos diferentes para subsistir sobre todo en países con subdesarrollo como el nuestro la cual merece una oportunidad de cambio no solo moral si no también tecnológico.

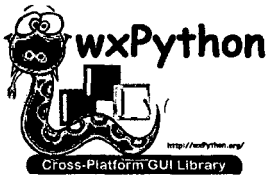


Figura 3.10: Una de las librerías de python.

3.3.1. Que es Python?.

- **Programa Interpretado y de Script.**

  Un lenguaje interpretado o de script es aquel que se ejecuta utilizando un programa intermedio llamado intérprete, en lugar de compilar el código a lenguaje máquina que pueda comprender y ejecutar directamente una computadora (lenguajes compilados).

  La ventaja de los lenguajes compilados es que su ejecución es más rápida. Sin

embargo los lenguajes interpretados son más flexibles y más potables.

Python tiene, no obstante, muchas de las características de los lenguajes compilados, por lo que se podría decir que es semi interpretado. En Python, como en Java y muchos otros lenguajes, el código fuente se traduce a un pseudo código máquina intermedio llamado bytecode la primera vez que se ejecuta, generando archivos .pyc o .pyo (bytecode optimizado), que son los que se ejecutarán en sucesivas ocasiones.

- **Tipado Dinámico.**

  La característica de tipado dinámico se refiere a que no es necesario declarar el tipo de dato que va a contener una determinada variable, sino que su tipo se determinará en tiempo de ejecución según el tipo del valor al que se asigne, y el tipo de esta variable puede cambiar si se le asigna un valor de otro tipo.

- **Fuertemente Tipado.**

  No se permite tratar a una variable como si fuera de un tipo distinto al que tiene, es necesario convertir de forma explícita dicha variable al nuevo tipo previamente. Por ejemplo, si tenemos una variable que contiene un texto (variable de tipo cadena o string) no podremos tratarla como un número . En otros lenguajes el tipo de la variable cambiaría para adaptarse al comportamiento esperado, aunque esto es más propenso a errores.

- **Multiplataforma.**

  El intérprete de Python está disponible en multitud de plataformas (UNIX, Solaris, Linux, DOS, Windows, OS/2, Mac OS, etc.) por lo que si no utilizamos librerías específicas de cada plataforma nuestro programa podrá correr en todos estos sistemas sin grandes cambios.

- **Orientado a Objetos.**

  La orientación a objetos es un paradigma de programación en el que los conceptos del mundo real relevantes para nuestro problema se trasladan a clases y *objetos* en nuestro programa. La ejecución del programa consiste en una serie de interacciones entre los objetos.

  Python también permite la programación imperativa, programación funcional y programación orientada a aspectos.

3.3.2. Librerías Principales de python.

Python dispone de una amplia colección de librerías, que simplifican nuestra tarea a la hora de escribir código. El objetivo de este tema es enseñar cómo funcionan algunas de las librerías, para que los usuarios noveles de este lenguaje tengan una base para utilizar la mayoría de las librerías disponibles, que son muchas, además de proporcionar unos consejos que, evitarán más de un dolor de cabeza o fallos inesperados a la hora de ejecutar un código.

- **Numpy**

  NumPy es el paquete fundamental para la computación científica con Python. Contiene entre otras cosas: un poderoso N-dimensional array de objetos sofisticados (radiodifusión) funciones herramientas para la integración de código C / C + + y Fortran código álgebra lineal útil, la transformada de Fourier, y la capacidad de números aleatorios

  Además de sus usos científicos obvias, NumPy también se puede utilizar como un eficiente multi-dimensional contenedor de datos genéricos. Arbitrarias de tipos de datos pueden ser definidos. Esto permite NumPy para integrar y rápidamente con una amplia variedad de bases de datos.

  Numpy está licenciado bajo la licencia BSD , lo que permite su re utilización con pocas restricciones.

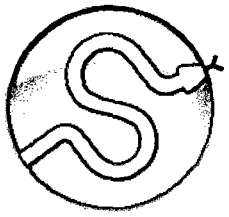


Figura 3.11: Símbolo de Numpy

- **Scipy**

  Depende directamente de numpy es una librería de calculo numérico sirve para establecer todo tipo de calculo numérico y potencialmente grande para establecer datos de todo tipo matemático.

- **Matplotlib**

  matplotlib es una de las biblioteca de trazado 2D que produce figuras de calidad de publicación,en una variedad de formatos impresos y entornos interactivos,a través de plataformas. Matplotlib se puede utilizar en scripts para python, la pitón y la ipython shell, servidores de aplicaciones web.

  Matplotlib trata de hacer las cosas fáciles . Puede generar gráficos, histogramas, espectro de potencia, gráficos de barras, errorcharts, diagramas de dispersión, etc, con sólo unas pocas líneas de código. [5]

3.3.3. Programa hecho en python

Con lo dicho anteriormente vamos ah establecer la primera parte del código fuente del sistema y hacer unos pequeños ejemplos de aplicación de vigas continuas y esto compararlo con programas como el sap.

```
#! /usr/bin/python

#PROGRAMA DE VIGAS_SAMAX PRIMERA PARTE DE TESIS
from numpy import *
from scipy import *
from matrix2d import *
from numpy.linalg import *
from scipy.linalg import *
from FForma import *
from Graficos import*
print (' PROGRAMA HECHA EN PYTHON PONER LOS DATOS DE
SISTEMAS NODALES [0,1.00000000]')

n=input('Colocar el Numero de Vigas :')
nodos=n+1
Cord=[]
for i in range(nodos):
    s=input('Colocar las Coordenadas de los nodos :')
    Cord = Cord + [s]

Cord1=array(Cord)

L=[]
for i in range(nodos):
    for j in range(nodos):
        if j-i==1:
            L = L + [norm(Cord1[j]-Cord1[i])]
#COLOCACION DE PROPIEDADES DE LA BARRA O VIGA
Pt=[]
for i in range(n):
    Pi=input('Colocar las Propiedades [E,b,h]:')
    Pt=Pt+[Pi]

#COEFICIENTES DE ELASTICIDAD DEL SISTEMA
```

```
E=[]
I=[]
for i in range(len(Pt)):
    E=E+[Pt[i][0]]

#ELEMENTOS DE INERCIA DE LOS ELEMENTOS
for i in range(len(Pt)):
    I=I+[Pt[i][1]*(Pt[i][2]**3)/12.0]

#MATRICES DE RIGIDEZ DE LOS ELEMENTOS
K=[]
for i in range(len(Pt)):
    s=Matrix(E[i],I[i],L[i])
    K=K+[s.Constructor()]
#ACOPLAMIENTO DE LA MATRIZ GLOBAL
R1=range(n)

R2=range(1,n+1)

U1=()
for i in range(len(R1)):
    for j in range(len(R2)):
        if i==j:
            U1=U1+(R1[i],R2[j])
U2=np.array([U1[i]+1 for i in range(len(U1))])

s=len(U1)/2
U3=np.reshape(U2,(s,2))
U4=U3*2


O=[]

WQ=array([range(U4[i][0]-2,U4[i][0]) for i in range(len(U4))])
WR=array([range(U4[i][1]-2,U4[i][1]) for i in range(len(U4))])
```

```
for i in range(len(WQ)):
    for j in range(len(WQ)):
        if i==j:
            O = O + [array([WQ[i],WR[j]]).reshape(4)]

U5=array(O)
Ord=transpose(U5)

"""MATRIZ DE ORDENAMIENTO ----->Ord"""
KT=zeros((2*nodos,2*nodos))

#PROCEDIMIENTO DE ENSAMBLAJE DE LA MATRIZ DE RIGIDEZ.
for i in range(4):
    for j in range(4):
        for s in range(len(K)):
            KT[Ord[i][s]][Ord[j][s]] =
            KT[Ord[i][s]][Ord[j][s]]+K[s][i][j]


#COLOCAR LOS GRADOS DE LIBERTAD DE LOS NODOS O APOYOS.
M=[]
for i in range(nodos):
    u=input('Nodos Empotrados [0,0] Nodos Fijos [0,1]
    Nodos Libres [1,1] :')
    M = M + [u]
N=array(M).reshape(2*nodos,1)



#COLOCAR LAS FUERZAS TANTO EN NODOS COMO EN ELEMENTOS DE RECURRENCIA.
F=[]
y=[]
for i in range(n):
    w=input('Colocar el Valor de [1] fuerza Puntual o
```

```
    [2] Fuerza Dist :')
    if w==1:
         """Deducidos en La Tesis"""
         q=input('Colocar [Q_i,Xi] :')
         Q1=Forma(q[1],L[i])
         U=[[q[0]*Q1.N_1(),q[0]*Q1.N_2()],[q[0]*Q1.N_3(),q[0]*Q1.N_4()]]
         y = y + [q]
         F = F + U
    if w==2:
         """Deducidos en La Tesis"""
         s=input('Colocar el valor de qi :')
         B=[[s*L[i]/2.0,s*L[i]**2/12.0],[s*L[i]/2.0,-s*L[i]**2/12.0]]
         F = F + B
         y = y + [[s,0]]

FV=array(y).reshape(n,2)




FI=array(F)

#MATRIZ DE FUERZAS DE LOS ELEMENTOS
#-----------------------------------------------------------
FR=[]
for i in range(1,len(FI)-1,2):
    for j in range(2,len(FI)-1,2):
        if i!=0:
            if j!=0:
                if i!=len(FI)-1:
                    if j!=len(FI)-1:
                        if j-i==1:
                            QW=[array(FI[i])+array(FI[j])]
                            FR = FR + QW
FU=array([array(FI[0])]+FR+[array(FI[len(FI)-1])])
```

```
.reshape(2*nodos,1)
#-------------------------------------------------------------

#CALCULO DE LA MATRIZ DE RIGIDEZ CORREGIDA .

GDL0=[]
for i in range(len(N)):
    if N[i]==0:
        GDL0 = GDL0+[i]

GDT=array(GDL0)
KM1=delete(KT,GDT,axis=0)
KM2=delete(KM1,GDT,axis=1)

FGDL=[]
for i in range(len(N)):
    if N[i]!=0:
       FGDL=FGDL+[FU[i]]
FGDL1=array(FGDL).reshape(len(KM2),1)


#HALLANDO EL VECTOR DE LOS DESPLAZAMIENTOS
#-------------------------------------------------
Desp=dot(inv(KM2),FGDL1)

#-------------------------------------------------
#PROCESO DE SEPARACION PROPIEDADES DE VIGAS
UPP=[]
for i in range(len(M)):
    for j in range(len(M)):
        if j-i==1:
            UPP=UPP+[M[i]+M[j]]
UPP1=array(UPP)
AMY=[]
for i in range(len(N)):
```

```
    if N[i]==1:
        AMY=AMY+[[i]]

AMY1=array(AMY)
for i in range(len(AMY1)):
    N[AMY[i]]=Desp[i]

N2=N.reshape(len(N)/2,2)

POR=[]
for i in range(len(N2)):
    for j in range(len(N2)):
        if j-i==1:
            POR=POR+[([N2[i]]+[N2[j]])]
POW=np.array(POR)

#---------------------------------------------------------------

#PARCIALIZAR LOS X[i] PARA DE TAL MOTIVO
SACAR LAS GRAFICAS DE LOS ELEMENTOS VIGAS.
X=[]
for i in range(n):
    x1=arange(0,L[i],0.1)
    X = X + [x1]
Form=[]
for i in range(n):
    QR=Forma(X[i],L[i])
    Form=Form+[QR]
Resid=[]
for i in range(n):
    QM=Residuo(X[i],L[i],FV[i][0],E[i],I[i])
    Resid = Resid + [QM]


DEF=[]
```

```
for i in range(n):
    QT=Form[i].N_1()*POW[i][0][0]+Form[i].N_2()*POW[i][0][1]+Form[i]
    .N_3()*POW[i][1][0]+Form[i].N_4()*POW[i][1][1]+Resid[i].R_1()
    DEF = DEF +[QT]


DEFC=[]
for i in range(n):
    QE=Form[i].Gr_1()*POW[i][0][0]+Form[i].Gr_2()*POW[i][0][1]+Form[i]
    .Gr_3()*POW[i][1][0]+Form[i].Gr_4()*POW[i][1][1]+Resid[i].R_2()
    DEFC = DEFC + [QE]
Mom=[]
for i in range(n):
    QU=Form[i].M_1()*POW[i][0][0]+Form[i].M_2()*POW[i][0][1]+Form[i]
    .M_3()*POW[i][1][0]+Form[i].M_4()*POW[i][1][1]+Resid[i].R_3()
    Mom = Mom +[QU]

Cort=[]
for i in range(n):
    QV=Form[i].V_1()*POW[i][0][0]+Form[i].V_2()*POW[i][0][1]+Form[i]
    .V_3()*POW[i][1][0]+Form[i].V_4()*POW[i][1][1]+Resid[i].R_4()
    Cort=Cort+[QV]


#-------------------------------------------------------------
#GRAFICAS _TERMINAREMOS YA EL PRIMER Y SEGUNDO CAPITULO ;)
#-------------------------------------------------------------
while True:
    p=input('Elegir Viga :')
    i=p-1
    DefT.plot(X[i],DEF[i],'ro')
    GirT.plot(X[i],-DEFC[i],'ro')
    MomT.plot(X[i],Mom[i],'ro')
    CorT.plot(X[i],-Cort[i],'ro')
    plt.show()
```

```
#-----------------------------------------------------------------------
# CREACION DE TABLAS DE RESULTADOS DE LOS ELEMENTOS
#-----------------------------------------------------------------------
```

Capítulo 4

PROBLEMAS DE ELASTICIDAD BIDIMENSIONAL.

4.1. Introducción.

En este capitulo se presenta la aplicación del método de los elementos finitos al análisis de estructuras en las que se cumplen las hipótesis de la elasticidad bidimensional(tensión o deformación plana).la mayor parte de los conceptos que aparecerán a lo largo del capitulo serán utilizados en el resto de la tesis al tratar otros problemas de estructuras en dos, e incluso tres dimensiones.Por consiguiente,este capitulo puede considerarse, en gran parte,como introductorio a la metodología general de aplicación del método de los elementos finitos a estructuras bi y tridimensionales. [2, Pag 47]

Existe una gran variedad de estructuras de interés practico dentro de la ingeniería en las que se puede hacer uso de las hipótesis de la elasticidad bidimensional.Dichas estructuras se caracterizan por tener una forma aproximada de prisma recto.No obstante,según la proporción que guarden las dimensiones de dicho prisma,y la disposición de las cargas,pueden clasificarse en uno de los tipos siguientes:

- **Problemas de Tensión Plana.**

  Se dice que una estructura prismática esta en estado de tension plana si una

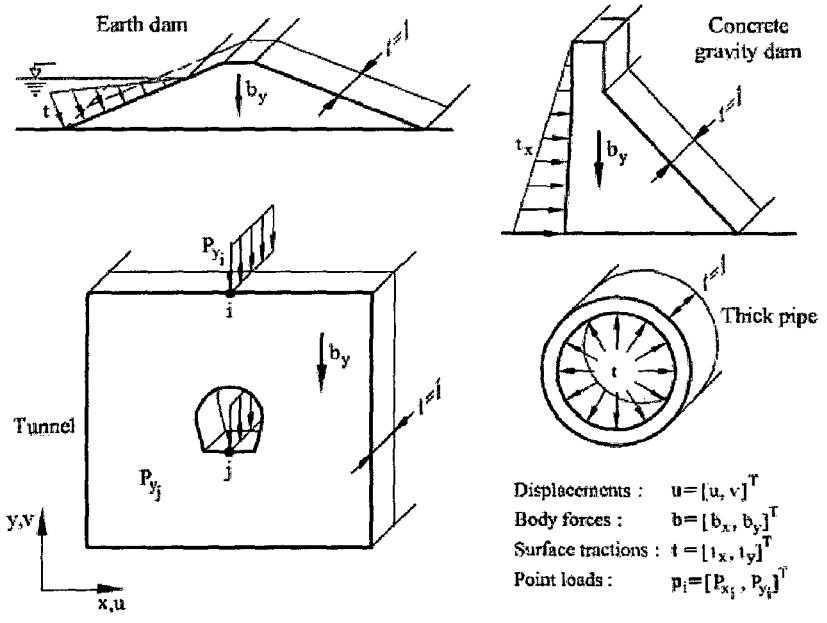


Figura 4.1: Elementos en Tensión Plana y Deformación Plana.

de sus dimensiones (espesor) es mucho menor que las otras dos, y sobre ellas actúan únicamente cargas contenidas en su plano medio.Entre los problemas de estructuras que se incluyen dentro de esta categoría podemos citar los de análisis de vigas de gran canto,placas con cargas en su plano,presas contrafuertes,etc.

- **Problemas de Deformación Plana.**

  Una estructura prismática está en estado de deformación plana si una de sus dimensiones (Longitud) es mucho mayor que las otras dos y sobre ella actúan únicamente cargas uniformemente distribuidas a lo largo de toda su longitud y contenidas en planos ortogonales al eje que une los centros de gravedad de sus distintas secciones transversales, dentro de esta clasificación se pueden incluir entre otros,los problemas de muro de contención,presas de gravedad,tuberías bajo presión interior y diversos problemas de ingeniería del terreno (túneles,análisis de tensiones bajo zapatas,etc) una de las principales ventajas de la teoría bidimensional es que permite el estudio de los problemas de tensión

y deformación plana. [3]

4.2. Teoria de la Elasticidad Bidimensional.

Presentemos los conceptos que hay que conocer de la teoría de elasticidad bidimensional para la utilización del método de elementos finitos.

4.2.1. Campo de Desplazamientos.

Las características geométricas y de cargas de una estructura en estado de tension o deformación plana permiten establecer la hipótesis de que todas las secciones perpendiculares al eje prismático z se deforman en su plano y de manera idéntica .Por consiguiente,basta con conocer el comportamiento de cualquiera de dichas secciones.Así ,consideramos una | genérica contenida en el plano $x - y$ de cualquiera de las figuras().El campo de desplazamientos de la sección esta perfectamente definido si se conocen los desplazamientos en las direcciones x y y de todos sus puntos.El vector de desplazamientos de un punto se define,por tanto,como [2, Pag 47]

$$u(x,y) = \left\{ \begin{array}{c} u(x,y) \\ \\ v(x,y) \end{array} \right\} \tag{4.1}$$

Donde $u(x,y)$ y $v(x,y)$ son los desplazamientos del punto en direcciones de los ejes x , y,respectivamente.

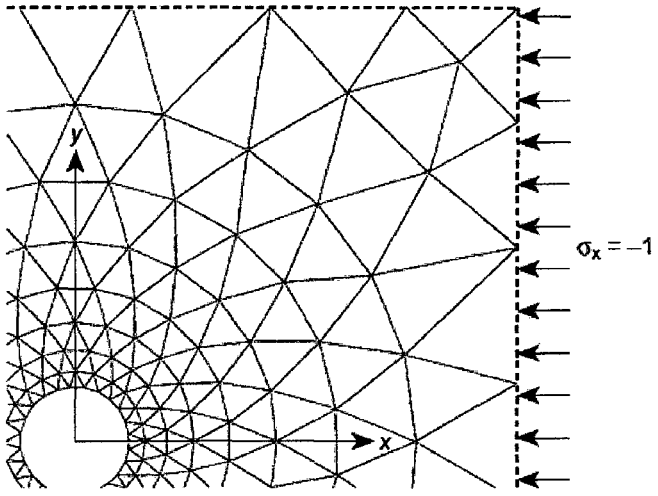


Figura 4.2: Tensión Plana

4.2.2. Campo de Deformaciones.

Analizando el campo de desplazamientos y haciendo uso de la teoría general de elasticidad .

$$\xi_x = \frac{\partial u}{\partial x} \tag{4.2}$$

$$\xi_y = \frac{\partial v}{\partial y} \tag{4.3}$$

$$\gamma_{xy} = \frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} \tag{4.4}$$

$$\gamma_{xz} = \gamma_{yz} = 0 \tag{4.5}$$

Con respecto a la deformación longitudinal ξ_z hay que señalar que en el caso de tensión plana dicha deformación no es nula ,pero se supone que lo es la tensión σ_z.Por consiguiente,en ninguno de los dos casos hay que considerar la deformación ξ_z ya que no interviene en las ecuaciones de trabajo de deformación al ser producto $\sigma_z \xi_z$ nulo.Así,pues el vector de deformación significativas de un punto se define para tensión y deformación plana como [4, Pag.134]

$$\xi = [\xi_x, \xi_y, \gamma_{xy}]^T \tag{4.6}$$

1

4.2.3. Campo de Tensiones.

Se deduce en las ecuaciones Ec.(4.2)... Ec(4.5) que las tensiones tangenciales τ_{xz} y τ_{yz} son nulas.Por otra parte por los mismos motivos explicados en el apartado anterior para la deformación ξ_z,la tensión σ_z no trabaja y el vector de tensiones significativas es [3, Pag.158]

$$\sigma = [\sigma_x, \sigma_y, \tau_{xy}]^T \tag{4.7}$$

4.2.4. Relación Tensión - Deformación.

La relación entre tensiones y deformaciones se deduce de la ecuación constitutiva de la elasticidad tridimensional,con la hipótesis simplificativas descritas anteriormente($\sigma_z = 0$ para tensión plana,$\xi_z = 0$ para deformación plana y $\gamma_{xz} = \gamma_{yz} = 0$ en ambos casos).Tras realizar las correspondientes operaciones puede encontrarse la siguiente relación matricial entre tensiones y deformaciones

$$\sigma = D\xi \tag{4.8}$$

En la ecuación Ec(4.8) D es la matriz de constantes elásticas (o matriz constitutiva)

$$D = \begin{bmatrix} d_{11} & d_{12} & 0 \\ d_{21} & d_{22} & 0 \\ 0 & 0 & d_{33} \end{bmatrix} \tag{4.9}$$

[1] [4]⟶ Sergio Gallegos Cazares,Análisis de Sólidos y Estructural mediante el método de elementos finitos

Del teorema de Maxwell-Betti se deduce que D es siempre simétrica,y $d_{12} = d_{21}$.Para elasticidad isotropa se tiene:

	Tensión Plana	Deformación Plana
	$d_{11} = d_{22} = \frac{E}{1-v^2}$	$d_{11} = d_{22} = \frac{E(1-v)}{(1+v)(1-2v)}$
	$d_{12} = d_{21} = \frac{vE}{1-v^2}$	$d_{12} = d_{21} = (\frac{E}{1-v^2})(\frac{v}{1-v})$
	$d_{33} = \frac{E}{2(1+v)}$	$d_{33} = \frac{E}{2(1+v)} = G$

(4.10)

Siendo E el módulo de elasticidad y v es el coeficiente de Poisson. Para un material ortótropo con direcciones principales de ortotropía según x e y,la matriz D tiene la expresión siguiente:

$$\text{Tensión Plana} \quad D = \frac{1}{1-v_{xy}v_{yx}} \begin{bmatrix} v_x & v_{xy}E_x & 0 \\ v_{yx}E_y & E_y & 0 \\ 0 & 0 & (1-v_{xy}v_{yx})G_{xy} \end{bmatrix} \tag{4.11}$$

$$\text{Tensión Plana} \quad D = \frac{1}{ad-bc} \begin{bmatrix} aE_x & bE_x & 0 \\ cE_y & dE_y & 0 \\ 0 & 0 & (ad-bc)G_{xy} \end{bmatrix}$$

Donde los diferentes factores estarían dados por:

$$\frac{1}{G_{xy}} = \frac{1+v_{yx}}{E_x} + \frac{1+v_{xy}}{E_y} \tag{4.12}$$

$$a = 1 - v_{yz}v_{zy} \quad b = v_{xy} + v_{xz}v_{zy} \tag{4.13}$$

$$c = v_{yx} + v_{yz}v_{zx} \quad d = 1 - v_{xz}v_{zx}$$

puesto que D debe ser simétrica se cumple

$$\frac{E_y}{E_x} = \frac{v_{xy}}{v_{yx}}(\text{Tensión Plana}) \text{ y } \frac{E_y}{E_x} = \frac{b}{c}(\text{Deformación Plana}) \tag{4.14}$$

Si las direcciones principales de ortotropía $x^{'}$,$y^{'}$ están inclinadas un ángulo θ con respecto a los ejes globales de la estructura x e y la matriz constitutiva se obtiene como sigue.Las deformaciones en eje x' y y',se expresan en función de sus valores en ejes x,y,por: [3, Pag 160]

$$(P-U)u = x^{'} \tag{4.15}$$

$$(P-U)u^{ort} = y^{'} \tag{4.16}$$

$$xcos(\theta) + ysen(\theta) = x^{'} \tag{4.17}$$

$$-xsen(\theta) + ycos(\theta) = y^{'} \tag{4.18}$$

- Con esto podemos analizar también los desplazamientos en u y en v con lo cual tendríamos lo siguiente: [6]

$$u^{'} = ucos(\theta) + vsen(\theta) \tag{4.19}$$

$$v^{'} = -ucos(\theta) + vcos(\theta) \tag{4.20}$$

- Con los datos Anteriores y usando las formulas general $\varepsilon = \frac{du}{dx}$ con la cual analizaremos los diferentes paso de coordenadas .

$$\varepsilon_{x'} = \frac{\partial u^{'}}{\partial x^{'}} = \frac{\partial u^{'}}{\partial x}\frac{\partial x}{\partial x^{'}} + \frac{\partial u^{'}}{\partial y}\frac{\partial y}{\partial x^{'}} \tag{4.21}$$

$$\varepsilon_{y'} = \frac{\partial v^{'}}{\partial y^{'}} = \frac{\partial v^{'}}{\partial x}\frac{\partial x}{\partial y^{'}} + \frac{\partial v^{'}}{\partial x}\frac{\partial x}{\partial y^{'}} \tag{4.22}$$

$$\gamma_{xy'} = \frac{\partial u^{'}}{\partial y^{'}} + \frac{\partial v^{'}}{\partial x^{'}} = \frac{\partial v^{'}}{\partial x}\frac{\partial x}{\partial x^{'}} + \frac{\partial v^{'}}{\partial y}\frac{\partial y}{\partial x^{'}} + \frac{\partial u^{'}}{\partial x}\frac{\partial x}{\partial y^{'}} + \frac{\partial u^{'}}{\partial x}\frac{\partial x}{\partial y^{'}} \tag{4.23}$$

- Reemplazando las ecuaciones Ec(4.17-4.18-4.19-4.20) en las ecuaciones Ec(4.21-4.22-4.23) con la cual tendríamos lo siguiente:

$$\varepsilon_x^{'} = \varepsilon_x cos^2(\theta) + \gamma_{xy} sen(\theta)cos(\theta) + \varepsilon_y sen^2(\theta) \quad (4.24)$$

$$\varepsilon_y^{'} = \varepsilon_x sen^2(\theta) - \gamma_{xy} sen(\theta)cos(\theta) + \varepsilon_y cos^2(\theta) \quad (4.25)$$

$$\gamma_{xy'} = \varepsilon_x - 2cos(\theta)sen(\theta) + \varepsilon_y 2cos(\theta)sen(\theta) + \gamma_{xy}(cos^2(\theta) - sen^2(\theta)) \quad (4.26)$$

- Acomodando el sistema en un sistema matricial el sistema quedaría de la siguiente manera:

$$\begin{Bmatrix} \varepsilon_{x'} \\ \varepsilon_{y'} \\ \varepsilon_{xy'} \end{Bmatrix} = \begin{pmatrix} cos^2(\theta) & sen^2(\theta) & sen(\theta)cos(\theta) \\ sen^2(\theta) & cos^2(\theta) & -sen(\theta)cos(\theta) \\ -2sen(\theta)cos(\theta) & 2sen(\theta)cos(\theta) & cos^2(\theta) - sen^2(\theta) \end{pmatrix} \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} \quad (4.27)$$

- La formula quedaría de la siguiente manera con la cual se podría establecer la siguiente formula:

$$\varepsilon^{'} = T\varepsilon \quad (4.28)$$

- Por otra parte también puede demostrase que que las tensiones en los ejes globales se relacionan con sus valores en ejes $x^{'}$ y $y^{'}$

$$\sigma = T^T \sigma^{'} \quad (4.29)$$

$$\sigma = D^{'} \varepsilon^{'} \quad (4.30)$$

Donde $D^{'}$ viene expresada en Ec(4.11) para material ortotrópo.

Finalmente,haciendo uso de las Ecs(4.29)-(4.30) se obtiene:

$$\sigma = T^T D^{'} T\varepsilon = D\varepsilon \quad (4.31)$$

$$D = T^T D^{'} T \quad (4.32)$$

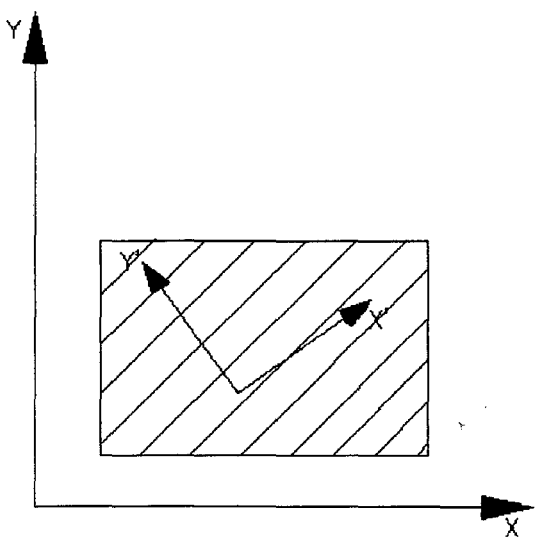


Figura 4.3: Material Ortótropo con direcciones principales de ortotropía x' y y'

Es fácil comprobar que la matriz de D en los ejes globales es también simétrica .La expresión de los coeficientes d_{ij} para el caso de elasticidad anisótropa. Si el solido esta sometido a un estado de deformación inicial,tal como puede suceder en el caso de deformación térmica,las relaciones pueden modificarse. La deformación totalε es ahora igual a la elástica ε^e más la inicial.Por otra parte,las tensiones siguen siendo proporcionales a las deformaciones elásticas,con lo que la ecuación constitutiva se describe como: [3, Pag 161]

$$\sigma = D\varepsilon^e = D(\varepsilon - \varepsilon^0) \tag{4.33}$$

4.2.5. Formulación de Elementos Finitos.Elemento de tres Nodos.

Consideraremos en primer lugar el uso del sencillo elemento triangular de tres nodos.Este elemento está considerado primario en el estudio de problemas estructurales bidimensionales por el método de elementos finitos.ya hemos comentado que mucho antes de la aparición de este método Courant sugirió el

uso de una interpolación polinomica lineal sobre sus subdominios triangulares para aproximar la solución numérica de ecuaciones diferenciales. [3, Pag 168]

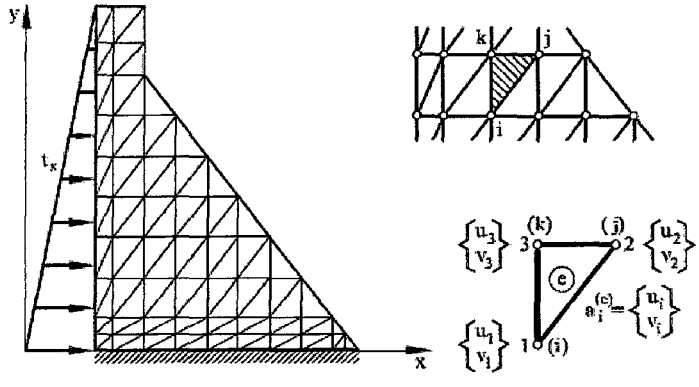
Años después,Turner et al.En un clásico articulo propusieron la división de los dominios bidimensionales en triángulos de tres nodos para facilitar su análisis matricial.Por ello,dicho elemento es conocido como elemento de Turner.El triangulo de tres nodos pronto adquirió gran popularidad entre ingenieros estructurales.De las muchas aplicaciones practicas de dicho elemento en su primera etapa hay que destacar las relaciones con el calculo de presas de gravedad que constituyeron una autentica innovación en la metodología tradicional del análisis de dichas estructuras [2, pag.55].La clave del éxito del elemento triangular de tres nodos fue su gran versatilidad y sencillez que,como veremos,permite asimilar fácilmente el proceso de análisis de un dominio bidimensional complejo a las etapas del clásico calculo matricial de estructuras de barras,familiar a la mayor parte de ingenieros estructurales.Por contrapartida,es un elemento de precision limitada,como corresponde a su aproximación lineal lo que obliga usualmente a la utilización de mallas muy tupidas pese a ello,en la actualidad,sigue siendo un elemento popular y competitivo,ademas de servir de ejemplo excelente para introducir la formulación de elementos finitos en problemas bidimensionales.

4.2.6. Discretzación del Campo de Desplazamiento.

En la figura 4.4 se muestra la sección transversal cualquiera que se analiza bajo la hipótesis de elasticidad bidimensional.La primera etapa del análisis es como siempre la discretizacion en elementos finitos.En la misma figura puede verse la discretizacion de la sección en elementos Triangulares de tres nodos es importante recordar de nuevo que la malla de elementos finitos representa una idealización de la geometría lineal.por consiguiente,el análisis por elementos finitos reproduce el comportamiento de la malla escogida y no el de la estructura

real.Solamente comprobando la convergencia de la solución podemos estimar el grado de aproximación de la solución de elementos finitos a la exacta.

Un elemento triangular de tres nodos típico se caracteriza por el numero de sus nodos $N : 1, 2, 3$ y sus coordenadas.Los tres nodos del elemento tienen en la malla la numeración global (i, j, k) y coordenadas x_1, y_1,x_2, y_2 y x_3, y_3.Los números globales de los nodos (i, j, k) corresponden con los locales 1,2 y 3,respectivamente en la practica es usual utilizar la numeración local para el calculo de las matrices del elemento y hacer uso de la correspondencia entre números locales y globales para el ensamblaje similar mente a como ocurre en calculo matricial de estructuras. Considerando un elemento aislado como en



Element	Node		Nodal variables		Nodal coordinates	
	Local	Global	Local	Global	Local	Global
ⓔ	1	i	u_1 v_1	u_i v_i	x_1 y_1	x_i y_i
	2	j	u_2 v_2	u_j v_j	x_2 y_2	x_j y_j
Area: $A^{(e)}$	3	k	u_3 v_3	u_k v_k	x_3 y_3	x_k y_k

Figura 4.4: Discretización de una Estructura con Elementos Triangulares.

la figura,podemos expresar los dos desplazamientos cartesianos de un punto cualquiera del interior del elemento en función de los desplazamientos de sus

nodos como

$$u = N_1 u_1 + N_2 u_2 + N_3 u_3 \tag{4.34}$$

$$v = N_1 v_1 + N_2 v_2 + N_3 v_3 \tag{4.35}$$

donde (u_i, v_i) y N_i son los desplazamientos horizontal y vertical y la función de forma del nodo i del elemento,respectivamente.No hay ninguna razón fundamental para escoger las mismas funciones para definir los desplazamientos en dirección horizontal y vertical.No obstante,por simplicidad, y a menos que haya claros indicios de que dicha aproximación debe diferenciarse, es usual utilizar la misma interpolación para ambos desplazamientos u,v. [3, Pag 170]

$$u = \left\{ \begin{array}{c} u \\ v \end{array} \right\} = \left[\begin{array}{cccccc} N_1 & 0 & N_2 & 0 & N_3 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 \end{array} \right] \left\{ \begin{array}{c} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{array} \right\} \tag{4.36}$$

$$u = N_i a^e \tag{4.37}$$

$$u = \left\{ \begin{array}{c} u \\ v \end{array} \right\} \tag{4.38}$$

Es el vector de desplazamientos de un punto del elemento.

$$N = [N_1, N_2, N_3] : N_i = \left[\begin{array}{cc} N_i & 0 \\ 0 & N_i \end{array} \right] \tag{4.39}$$

Son las funciones de forma del elemento y el nodo i del elemento,respectivamente,y

$$a^e = \left\{ \begin{array}{c} a_1^e \\ a_2^e \\ a_3^e \end{array} \right\} \tag{4.40}$$

$$a_i^e = \left\{ \begin{array}{c} u_i \\ v_i \end{array} \right\} \tag{4.41}$$

Son el vector de desplazamientos nodales del elemento y un nodo i. Adviertan se que N y a^e están compuestos de tantas submatrices N_i y subvectores a_i^e,respectivamente ,como nodos tiene el elemento.Esto es una propiedad general que se cumple en todos los casos.

La expresión de las funciones de forma del elemento triangular de tres nodos se puede obtener como sigue.

Los tres nodos del elemento definen una variación lineal del campo de desplazamientos que puede escribirse como:

$$u = \alpha_1 + \alpha_2 x + \alpha_3 y \tag{4.42}$$

$$v = \alpha_4 + \alpha_5 x + \alpha_6 y \tag{4.43}$$

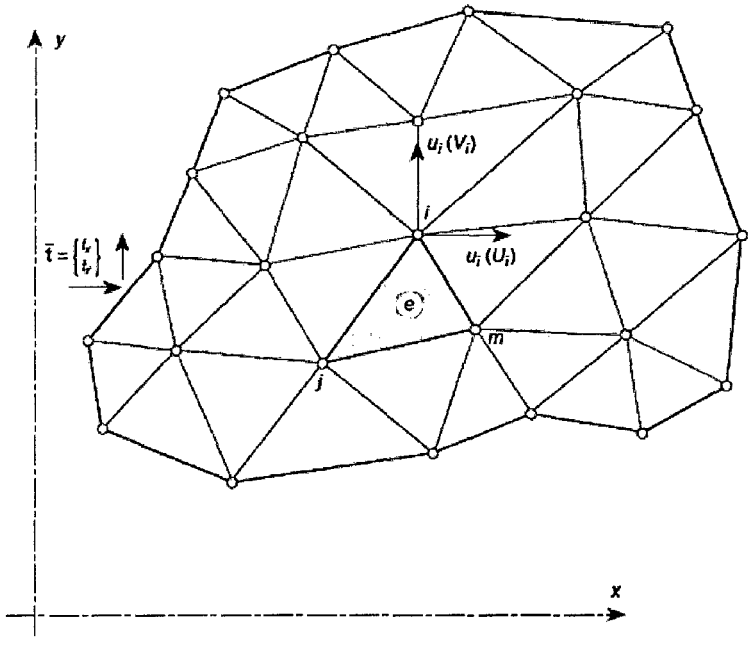


Figura 4.5: Elemento discretizado en forma Triangular.

$$u_1 = \alpha_1 + \alpha_2 x_1 + \alpha_3 y_1 \tag{4.44}$$

$$u_2 = \alpha_1 + \alpha_2 x_2 + \alpha_3 y_2 \tag{4.45}$$

$$u_3 = \alpha_1 + \alpha_2 x_3 + \alpha_3 y_3 \tag{4.46}$$

Ahora resolviendo dicho problema se tendría de la siguiente manera:

$$u = \frac{1}{2A}[(a_1 + b_1 x + c_1 y)u_1 + (a_2 + b_2 x + c_2 y)u_2 + (a_3 + b_3 x + c_3 y)u_3] \tag{4.47}$$

Se deduce que las funciones de forma del elemento triangular tienen la siguiente forma: [3, Pag.75]

$$N_i = \frac{1}{2A}(a_i + b_i x + c_i y) \tag{4.48}$$

4.2.7. Discretización del campo de deformaciones.

Usando la forma general de la deformación unitaria en un punto del elemento como:

$$\varepsilon_x = \frac{\partial N_1}{\partial x}u_1 + \frac{\partial N_2}{\partial x}u_2 + \frac{\partial N_3}{\partial x}u_3 \tag{4.49}$$

$$\varepsilon_y = \frac{\partial N_1}{\partial y}v_1 + \frac{\partial N_2}{\partial y}v_2 + \frac{\partial N_3}{\partial y}v_3 \tag{4.50}$$

$$\gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} = \frac{\partial N_1}{\partial y}u_1 + \frac{\partial N_1}{\partial x}v_1 + \frac{\partial N_2}{\partial y}u_2 + \frac{\partial N_2}{\partial x}v_2 + \frac{\partial N_3}{\partial y}u_3 + \frac{\partial N_3}{\partial x}v_3 \tag{4.51}$$

Haciendo un arreglo matricial de las ecuaciones planteadas en la parte anterior tendríamos:

$$\varepsilon = \begin{Bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} \end{Bmatrix} = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial y} & 0 & \frac{\partial N_3}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{Bmatrix} \tag{4.52}$$

$$\varepsilon = Ba^e \tag{4.53}$$

$$B = [B_1, B_2, B_3] \tag{4.54}$$

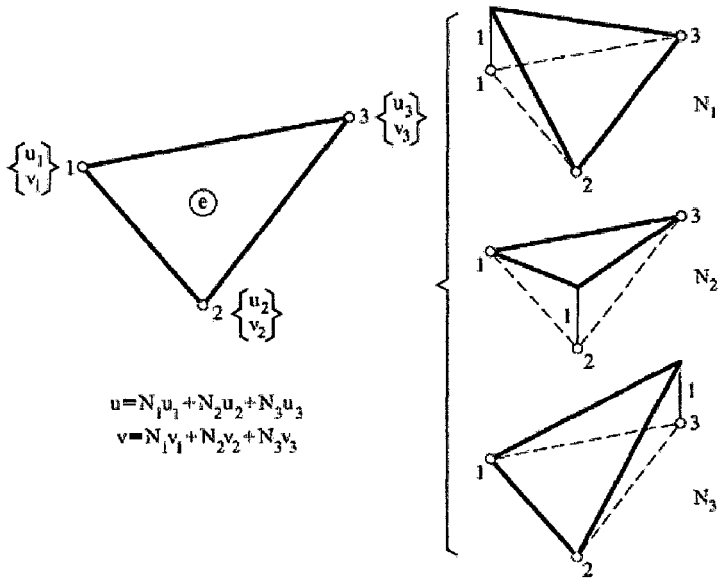


Figura 4.6: Funciones de forma del elemento triangular de tres nodos.

Es la matriz de deformación del elemento,y en forma general quedaría de la siguiente manera:

$$B_i = \begin{bmatrix} \frac{\partial N_i}{\partial x} & 0 \\ 0 & \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} \end{bmatrix} \tag{4.55}$$

es la matriz de deformación del nodo i.

Adviertan se que B está compuesta de tantas submatrices B_i como nodos tiene el elemento,lo que también es una propiedad de carácter general particularizando para los elementos triangular de tres nodos se obtiene,utilizando [4,

pag 195]

$$B = \frac{1}{2A}\begin{bmatrix} b_1 & 0 & b_2 & 0 & b_3 & 0 \\ 0 & c_1 & 0 & c_2 & 0 & c_3 \\ c_1 & b_1 & c_2 & b_2 & c_3 & b_3 \end{bmatrix} \tag{4.56}$$

y por consiguiente

$$B_i = \frac{1}{2A}\begin{bmatrix} b_i & 0 \\ 0 & c_i \\ c_i & b_i \end{bmatrix} \tag{4.57}$$

4.2.8. Discretización del campo de tensiones.

La expresión discretizada del vector de tensiones en el interior del elemento se obtiene mediante sustitución directa de las ecuaciones vistas en la primera parte de la tesis:

$$\sigma = D\varepsilon = DBa^e \tag{4.58}$$

Si existiera tensiones o deformaciones iniciales la expresión a utilizar se deduce de la siguiente manera:

$$\sigma = D(\varepsilon - \varepsilon_0) + \sigma^0 = DBa^e - D\varepsilon^0 + \sigma^0 \tag{4.59}$$

Puede apreciarse en la ecuación EC(5.47) que la matriz de deformación del elemento triangular de tres nodos es constante,lo que implica que las deformaciones y tensiones son constantes en todo el elemento.Esto es consecuencia directa del campo de desplazamiento lineal escogido, cuyos gradientes son obviamente,constantes.

Por consiguiente,en zonas de alta concentración de tensiones será necesario utilizar una malla tupida para aproximar la solución de tensiones con suficiente precisión. [3, Pag 173]

4.2.9. Ecuaciones de Equilibrio de la Discretización.

Para la obtención de las ecuaciones de equilibrio de la discretización partiremos de la expresión del PTV aplicada al equilibrio de un elemento aislado hay que resaltar que,aunque nos referiremos al elemento triangular de tres nodos la mayoría de las expresiones que se obtendrán en este apartado son completamente generales y aplicables a cualquier elemento bidimensional. Supondremos

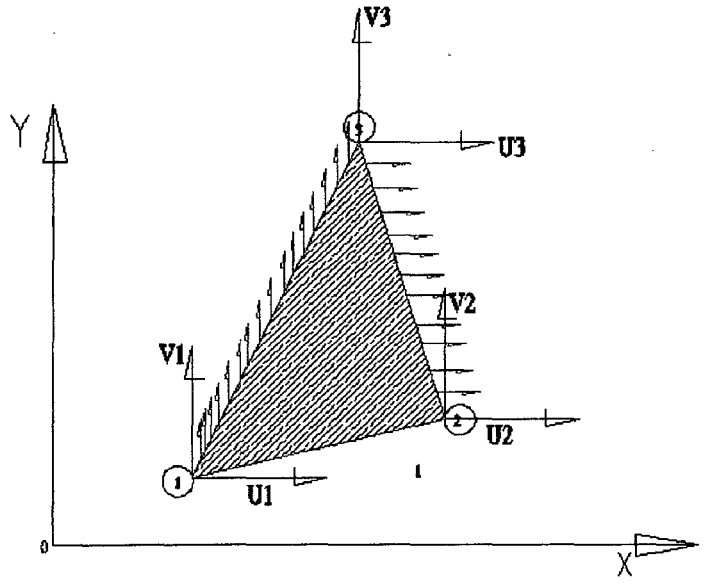


Figura 4.7: Fuerza Sobre un elemento triangulo de tres nodos

ahora que el equilibrio del elemento se establece únicamente en los nodos.Podemos entonces definir unas fuerzas puntuales que actúen sobre los nodos (denominaremos fuerzas nodales de equilibrio) y que se equilibren las fuerzas debidas a la deformación del elemento y al resto de las fuerza actuantes del mismo.Para

el cálculo de las fuerzas nodales de equilibrio haremos uso de la expresión del P.T.V aplicada el elemento,que se describe como: [4, Pag 200]

$$\int\int_A \delta\varepsilon^T \sigma t dA = \int\int_{A^e} \delta u^T b t dA + \oint_{l_e} \delta u^T t ds + \sum_{i=1}^{3} \delta u_i U_i + \sum_{i=1}^{3} \delta v_i V_i \quad (4.60)$$

Donde δu_i y δv_i son los desplazamientos virtuales de los nodos del elemento U_i y V_i las fuerzas nodales de equilibrio que corresponden a dichos desplazamientos .El trabajo virtual de dichas fuerzas pueden despejarse de la ecuación anterior como:

$$\int\int_{A^e} \delta\varepsilon^T \sigma t dA - \int\int_{A^e} \delta u^T b t dA - \oint \delta u^T t t ds = [\delta a_e]^T q_e \quad (4.61)$$

donde para el elemento triangular de tres nodos.

$$\delta a^e = [\delta u_1, \delta v_1, \delta u_2, \delta v_2, \delta u_3, \delta v_3]^T \quad (4.62)$$

$$q^e = [U_1, V_1, U_2, V_2, U_3, V_3] \quad (4.63)$$

Podemos escribirlo de la siguiente manera:

$$\delta u^T = [\delta a^T] N^T \quad (4.64)$$

$$\delta\varepsilon^T = [\delta a^T] B^T \quad (4.65)$$

Sustituyendo en la ecuación EC(4.60) en la EC(4.61) se obtiene ,tras sacar factor común δa^T,en el primer miembro.

$$[\delta a^e]^T [\int\int_{A^e} B^T \sigma t dA - \int\int_{A^e} N^T b t dA - \oint_{l_e} N^T \mathbf{t} t dS] = [\delta a^e]^T q^e \quad (4.66)$$

Teniendo en cuenta que los desplazamientos virtuales son arbitrarios ,se deduce que:

$$\int\int_{A^e} B^T \sigma t dA - \int\int_{A^e} N^T b t dA - \oint_{l^e} N^T \mathbf{t} t ds = q^e \quad (4.67)$$

Las ecuaciones anterior el equilibrio,entre las fuerzas nodales de equilibrio y las fuerzas debidas a la deformación del elemento(primera integral),las fuerzas másicas (segunda integral) y las de superficie (tercera integral).Sustituyendo

ahora el vector de tensiones σ por sus valor en función de los desplazamientos nodales utilizando la forma más general de la ecuaciones anteriores ya vistas. [3, Pag 175]

$$[\int\int_{A^e} B^T DBtdA]a^e - \int\int_{A^e} B^T D\varepsilon^0 tdA + \int\int_{A^e} B^T \sigma^0 tdA - \int\int_{A^e} N^T btdA - \oint_{l_e} N^T tds = q_e \tag{4.68}$$

Simplificando la ecuación en forma general la cual se estaría estableciendo de la siguiente manera:

$$K^e a^e - f^e = q^e \tag{4.69}$$

$$K^e = \int\int_{A^e} B^T DBtdA \tag{4.70}$$

Es la Matriz de Rigidez del elemento y:

$$f^e = f^e_\varepsilon + f^e_\sigma + f^e_b + f^e_t \tag{4.71}$$

el vector de fuerzas nodales equivalentes del elemento,siendo:

$$f^{(e)}_\varepsilon = \int\int_{A^e} B^T D\varepsilon^0 tdA \tag{4.72}$$

$$f^{(e)}_\sigma = -\int\int_{A^{(e)}} B^T \sigma^0 tdA \tag{4.73}$$

$$f^{(e)}_b = \int\int_{A^e} N^T btdA \tag{4.74}$$

$$f^{(e)}_t = \oint_{l^e} N^T \mathbf{t} tds \tag{4.75}$$

Los vectores de fuerzas nodales equivalentes debidos a deformaciones iniciales,tensiones iniciales,y fuerzas repartidas por unidad de área y fuerzas repartidas en el contorno,respectivamente.

La ecuación de equilibrio global de la malla se obtiene,como en el caso de problemas unidimensionales,estableciendo simplemente que la suma de sus fuerzas nodales de equilibrio en cada nodo debe ser igual a la fuerza nodal exterior.Es decir: [4, pag 205]

$$\sum_e q^{(e)}_i = P_j \tag{4.76}$$

4.3. Otros Elementos Bidimensionales y el Método de Interpolación de Lagrange.

La definición de las funciones de interpolación o funciones bases que pueden realizarse en el sistema de coordenadas global que se emplean para definir una estructura;sin embargo,se ha encontrado que ello no es muy conveniente por que los elementos así definidos ,por lo general,dependen de los parámetros característicos de la geometría y no son fáciles de adaptar a nuevas configuraciones .Tal es el caso de uno de los primeros elementos bidimensional que tienen forma rectangular: [4, Pag 81]

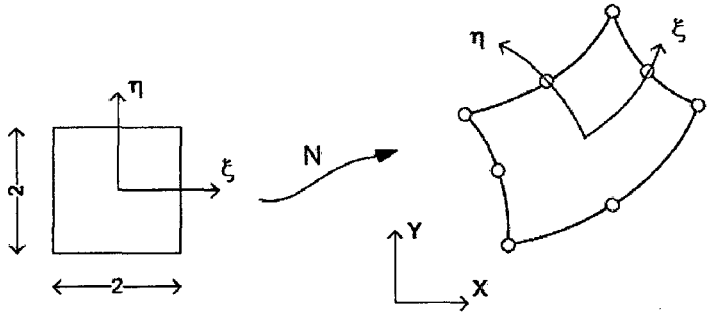


Figura 4.8: Interpolación de Coordenadas

Una manera de mantener una geometría simple durante la definición de las funciones de interpolación,y de ganar flexibilidad en la representación geométrica de los problemas ,es la definición de un espacio Natural que es independiente del sistema de coordenadas del sistema. [4, Pag 83]

Para definir elementos bidimensionales se emplean ,comúnmente ,dos espacios diferentes ,uno es rectangular y otro triangular en las cuales se definen ,respectivamente ,elementos cuadriliterales y triangulares . El uso de los espacios naturales para la definición de los elementos implica,asimismo,la utilización de funciones de mapeo hacia el espacio real de aplicación del elemento implica ,asimismo ,la utilización de funciones de mapeo hacia el espacio carte-

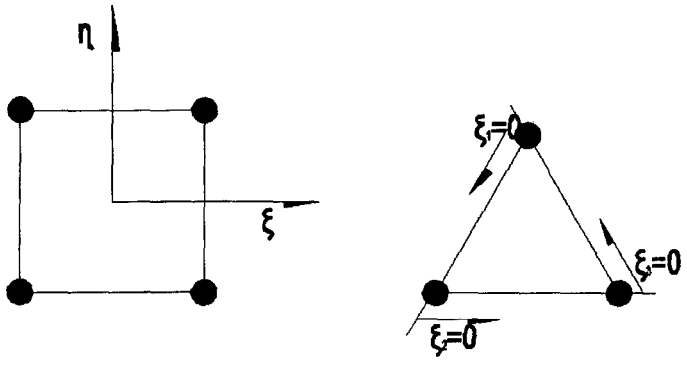


Figura 4.9: Espacios Triangular y Rectangular

siano. [4, Pag 85] Se inicia relacionando las geometrías de los espacios,para

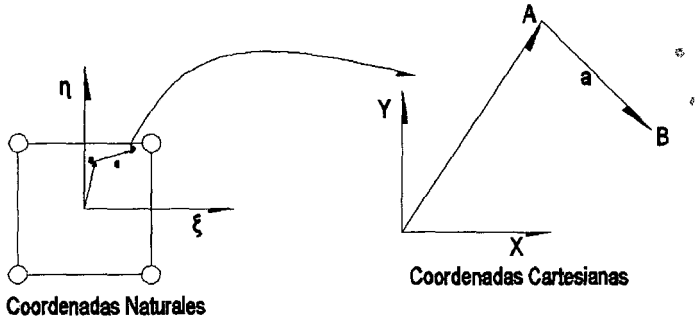


Figura 4.10: Mapeo entre los puntos del espacio Natural y el Espacio Cartesiano

lo cual en la figura 4.10 anterior se muestra como un ejemplo de geometría de un cuadrilátero en dos dimensiones .El punto a en el espacio con coordenadas ε_a se relacionan con el punto A de l espacio cartesiano con coordenadas X_A mediante la expresión:

$$X_A = \varphi_{(\xi a, \eta_a)} = \varphi_{\xi_a} \tag{4.77}$$

Un segundo par de puntos,b y B adyacentes a los puntos α y A se relacionan de la misma manera:

$$X_B = \varphi_{(\xi_b, \eta_{(b)})} = \varphi_{\xi_b} \tag{4.78}$$

Si el punto B se posiciona con respecto al punto A mediante el vector unitario a y la diferencia lineal ΔL,y el punto b se posiciona respecto al punto α mediante el vector unitario entonces:

$$X_B = X_A + \Delta La \tag{4.79}$$

$$\xi_b = \xi_a + \Delta l\alpha \tag{4.80}$$

Considerando esta última relación ,la ecuación se puede desarrollar empleando una expansión de la serie de **Taylor** al rededor del punto mediante.

$$x_B = \varphi_{\xi_b} = \varphi_{\xi_a + \Delta l\alpha} = \varphi_{\xi_a} + \frac{\delta\varphi}{\delta\xi}\,|_{\xi_a}\,\{\Delta l\alpha\} + O\{\Delta l^2\} \tag{4.81}$$

Igualando las ecuaciones EC(4.79) con la EC(4.75),se relacionan las distancias entre los puntos AB y ab en los dos espacios la cual quedaría de la siguiente manera:

$$\Delta La = \frac{\delta\varphi}{\delta\xi}|_{\xi_a}\{\Delta l\alpha\} + O(\Delta l^2) \tag{4.82}$$

En el limite,cuando el punto B se aproxima el punto A,y el punto b se aproxima al punto a,los vectores diferenciales entre los segmentos de linea correspondientes en el espacio natural y cartesiano. [4, Pag 85]

$$dx = \frac{\partial\varphi}{\partial\xi}d\xi = Jd\xi \tag{4.83}$$

La matriz que liga los elementos diferenciales de linea se denomina el jacobiano de la transformación y se denota por J.

Una dimensión:

$$J = \frac{\partial x}{\partial\xi} \tag{4.84}$$

En dos dimensiones.

$$J = \begin{bmatrix} \frac{\partial x}{\partial\xi} & \frac{\partial y}{\partial\xi} \\ \\ \frac{\partial x}{\partial\eta} & \frac{\partial y}{\partial\eta} \end{bmatrix} \tag{4.85}$$

En tres dimensiones:

$$J = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} \tag{4.86}$$

4.3.1. Elementos Lineales.

Considere un elemento lineal elástico definido por dos nodos .Este elemento se muestra un espacio natural y el espacio real. Usando el método de álgebra

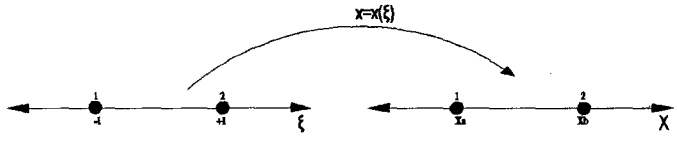


Figura 4.11: Mapeo del espacio natural al espacio real para un elemento lineal unidimensional

lineal se tendría que la forma de relacionar los dos espacios serian:

$$x = a_0 + a_1\xi = [1, \xi] \begin{Bmatrix} a_0 \\ \\ a_1 \end{Bmatrix} \tag{4.87}$$

la aproximación por elementos finitos emplea valores nodales de la función misma,de manera que es importante utilizar la aproximación nodal sorteando los valores de ξ,para distintos casos:

esta tabla se emplea para generar mediante la ecuación $Pa = \widehat{u}$ y también

Nodos a	ξ_a
1	-1
2	+1

Cuadro 4.1: Relación entre nodos de los elementos lineales y sus coordenadas naturales.

$a = A^{-1}\overline{u}.$

$$\begin{Bmatrix} x_1^e \\ \\ x_2^e \end{Bmatrix} = \begin{bmatrix} 1 & -1 \\ \\ 1 & 1 \end{bmatrix} \begin{Bmatrix} a_0 \\ \\ a_1 \end{Bmatrix} \tag{4.88}$$

$$[1, \xi] \begin{bmatrix} 1 & -1 \\ \\ 1 & 1 \end{bmatrix}^{-1} \begin{Bmatrix} x_1^e \\ \\ x_2^e \end{Bmatrix} \tag{4.89}$$

$$[N_1, N_2] \begin{Bmatrix} x_1^e \\ \\ x_2^e \end{Bmatrix} \tag{4.90}$$

$$N_1 = \frac{1-\xi}{2} \tag{4.91}$$

$$N_2 = \frac{1+\xi}{2} \tag{4.92}$$

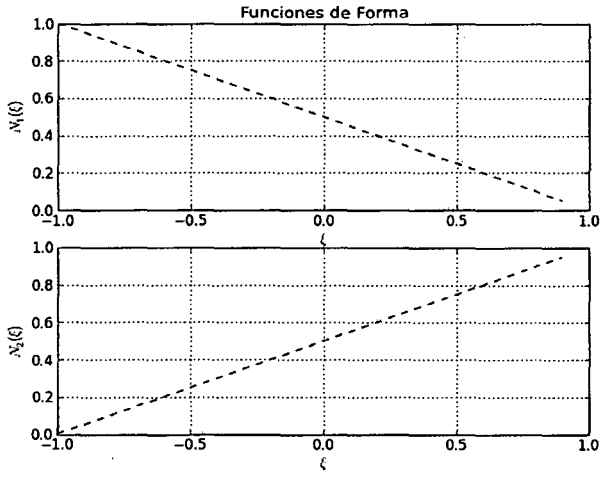


Figura 4.12: Funciones de forma para un elemento lineal Unidimensional.

```
from numpy import *
import matplotlib.pyplot as plt
from pylab import *

def f(x):
    return 0.5*(1-x)
def g(x):
    return 0.5*(1+x)
x=arange(-1,1,0.1)
figure(1)
subplot(211)
title('Funciones de Forma')
xlabel(r'$\xi$')
ylabel(r'$N_{1}(\xi)$')
grid(True)
plot(x,f(x),'r--')
subplot(212)
xlabel(r'$\xi$')
ylabel(r'$N_{2}(\xi)$')
grid(True)
```

```
plot(x,g(x),'r--')
show()
```

Puesto que el elemento es isoparametrico,se empleará la misma interpolación para interpolar funciones en este espacio:

$$\widehat{u} = \sum_{a=1}^{2} N_a(\xi)\overline{u_a}^e \tag{4.93}$$

Esta aproximación será lineal a partir de los valores nodales ,para que esta interpolación sea adecuada,es necesario verificar que se cumplen los requisitos de continuidad ,suavidad y completez.

La continuidad en el dominio de este elemento será del tipo C^0 puesto que las funciones de forma son lineales;la primera derivada será constante,pero la segunda derivada no está ya definida.A través de la frontera, la función es continua si dos elementos se unen con el mismo nodo,pero la primera derivada es discontinua.Estas funciones serán adecuadas en problemas que tengan hasta primeras derivadas $(m = 1)$ en el integrando de las ecuaciones se nota continuidad en la primera función y discontinuidad en la primera derivada.

La suavidad de la interpolación se verifica gráficamente,en donde se observa que las funciones de forma no tiene ningún punto singular en el dominio del elemento y,por tanto,la función interpolada tampoco tendrá y el mapeo se realizará uno a uno y sobre todo el dominio.El jacobiano de transformación para este elemento se define mediante L^e del elemento como: [4, Pag 88]

$$\frac{\partial x}{\partial \xi} = \frac{dx}{d\xi} = \left[\frac{dN_1}{d\xi}, \frac{dN_2}{d\xi}\right] \begin{Bmatrix} x_1^e \\ x_2^e \end{Bmatrix} = \left[\frac{-1}{2}, \frac{1}{2}\right] = \frac{x_2^e - x_1^e}{2} = \frac{L_e}{2} \tag{4.94}$$

Esto es, el jacobiano es un escalar que representa la distancia o longitud entre los dos nodos del elemento,por lo que la única posibilidad de que sea cero es que el mapeo se realice por dos nodos coincidentes.Fuera de ese caso,esta interpolación siempre será suave.

Finalmente,la completes se verifica al comprobar que la interpolación es capaz

de describir exactamente en un polinomio de primer grado.Esto es todo lo que necesita una función de la clase C^0.Considérese ,entonces,que los grados de libertad nodales se especifican de acuerdo con un polinomio de primer grado:

$$\overline{u}_0^e = c_0 + c_1 x_a^e \tag{4.95}$$

Utilizando la interpolación propuesta en la ecuación $\widehat{u} = \sum_{a=1}^{2} N_a(\xi)\overline{u}_a^e$,junto con estos valores nodales,se obtienen:

$$\widehat{u} = \sum_{a=1}^{2} N_a(\xi)\overline{u}_a^e \tag{4.96}$$

$$\overline{u} = \sum_{a=1}^{2} N_a(\xi)(c_0 + c_1 x_a^e) \tag{4.97}$$

$$\overline{u} = \left(\sum_{a=1}^{2} N_a(\xi)\right) c_0 + \left(\sum_{a=1}^{2} N_a(\xi) x_a^e\right) c_1 \tag{4.98}$$

El coeficiente del primer término implica la partición de la unidad:

$$\sum_{a=1}^{2} N_a(\xi) = N_1(\xi) + N_2(\xi) = \frac{1}{2}(1-\xi) + \frac{1}{2}(1+\xi) = 1 \tag{4.99}$$

Debe notarse que el uso del concepto isoparametrico es el que permite la simplificación de la ecuación a un polinomio lineal.Esta ventaja de los elementos isoparametricos,ya que,básicamente ,para verificar los requisitos de convergencia de interpolación lo único que hace falta es demostrar que se cumpla los requisito de partición de la unidad. [4, Pag 89]

$$\sum_{a=1}^{2}(\xi) = 1 \tag{4.100}$$

El elemento que se presenta en esta sección se emplea para modelar un dominio bidimensional plano que se define mediante la posición de 4 nodos .La numeración de los nodos y la configuración geométrica del elemento se muestran en la figura 4.14: [4, Pag 89] De nueva cuenta ,como en el caso unidimensional,el espacio de definición será el espacio natural y será necesario realizar un mapeo al espacio real de la aplicación ,el cual se muestra en la siguiente figura: la aproximación geométrica se puede expresar mediante polinomios.Dado

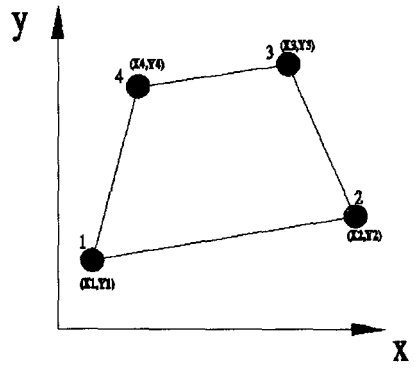


Figura 4.13: Elemento bilineal de cuatro nodos.

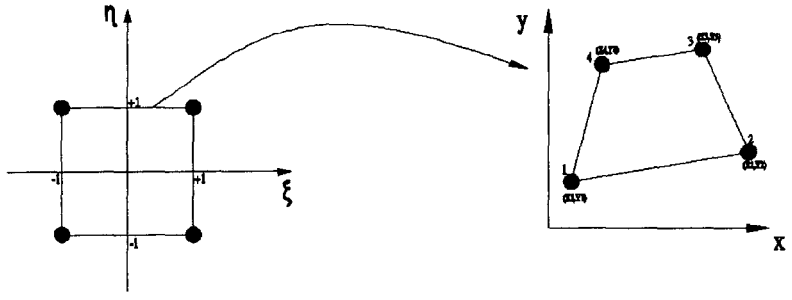


Figura 4.14: Mapeo del espacio natural bidimensional al espacio cartesiano real.

que se tiene cuatro nodos para representación geométrica,es natural pensar en proponer en aproximaciones no-nodal en términos de cuatro coeficientes

indeterminados para x,y. [4, Pag 95]

$$x(\xi,\eta) = a_0 + a_1\xi + a_2\eta + a_3\xi\eta = \begin{bmatrix} 1 & \xi & \eta & \xi\eta \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{Bmatrix} \tag{4.101}$$

$$y(\xi,\eta) = \beta_0 + \beta_1\xi + \beta_2\eta + \beta_3\xi\eta = \begin{bmatrix} 1 & \xi & \eta & \xi\eta \end{bmatrix} \begin{Bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{Bmatrix} \tag{4.102}$$

La transformación a una aproximación nodal se logra al evaluar las coordenadas mediante.

$$x(\xi_a, \eta_a, a_0, \cdots) \tag{4.103}$$

Nodo a	ξ_a	η_a
1	-1	-1
2	+1	-1
3	+1	+1
4	-1	+1

Cuadro 4.2: Relación de los nodos del elemento bilineal y sus coordenadas naturales.

Sustituyendo estas coordenadas de las ecuaciones anteriores se obtienen:

$$\{x_a^e\} = \begin{bmatrix} x_1^e \\ x_2^e \\ x_3^e \\ x_4^e \end{bmatrix} = \begin{bmatrix} 1 & -1 & -1 & +1 \\ 1 & +1 & -1 & -1 \\ 1 & +1 & +1 & +1 \\ 1 & -1 & +1 & -1 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{Bmatrix} = Aa_i \tag{4.104}$$

$$\{x_a^e\} = \begin{bmatrix} y_1^e \\ y_2^e \\ y_3^e \\ y_4^e \end{bmatrix} = \begin{bmatrix} 1 & -1 & -1 & +1 \\ 1 & +1 & -1 & -1 \\ 1 & +1 & +1 & +1 \\ 1 & -1 & +1 & -1 \end{bmatrix} \begin{Bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{Bmatrix} = A\beta_i \tag{4.105}$$

Los coeficientes indeterminados se pueden obtener de esta ultima expresión:

$$a_i = A^{-1}\{x_a^e\} \tag{4.106}$$

$$\beta_i = A^{-1}\{y_a^e\} \tag{4.107}$$

Finalmente,la interpolación geométrica nodal se obtiene como:

$$x = \begin{bmatrix} 1 & \xi & \eta & \eta\xi \end{bmatrix} a_i \tag{4.108}$$

$$x = \begin{bmatrix} 1 & \xi & \eta & \eta\xi \end{bmatrix} A^{-1}\{x_i\} \tag{4.109}$$

$$x = \begin{bmatrix} N_1 & N_2 & N_3 & N_4 \end{bmatrix} \{x_i\} \tag{4.110}$$

Las funciones de forma o interpolación están dadas por:

$$N_1 = \frac{1}{4}(1-\xi)(1-\eta) \tag{4.111}$$

$$N_2 = \frac{1}{4}(1+\xi)(1-\eta) \tag{4.112}$$

$$N_3 = \frac{1}{4}(1+\xi)(1+\eta) \tag{4.113}$$

$$N_4 = \frac{1}{4}(1-\xi)(1+\eta) \tag{4.114}$$

4.3.2. Interpolación de alta jerarquía de clase C_0,Interpolación Lagrangiana.

Para el tipo de funciones C_0 solo se requiere que las funciones de interpolación se evalúen a partir de valores nodales.En teoría de interpolación se encuentran que una familia de polinomios que cumplen precisamente con ese requisito es la de polinomios de Lagrange.

La interpolación se plantea de la siguiente forma para problemas unidimensionales.Si tienen (n) puntos $\xi_0 \cdots \xi_n$ y la función f tiene valores conocidos en esos puntos,entonces es posible encontrar un polinomio (ξ) único,de grado máximo $(n-1)$,que al evaluarse en esos puntos ,toma el valor de la función .El polinomio se expresa como:[4, Pag 106]

$$P(\xi) = f(\xi_1)L_{n-1,1}(\xi) + \cdots + f(\xi_n)L_{n-1,n}(\xi) \tag{4.115}$$

en donde los polinomios de lagrange se calculan mediante:

$$L_{n-1,k}(\xi) = \prod_{i=1}^{n} \frac{(\xi - \xi_i)}{(\xi_k - \xi_i)} \tag{4.116}$$

Comparando la ecuación EC(4.115) con la forma estandard de la interpolación:

$$\widehat{u} = \sum_{a=1}^{n} N_a(\xi)\overline{u_a}^e \tag{4.117}$$

Es posible concluir que los polinomios de Lagrange pueden ser las funciones de forma directamente:

$$N_a(\xi) = L_{n-1,a}(\xi) \tag{4.118}$$

Figura 4.15: Elemento Unidimensional de dos nodos.

- **Ejemplo de Aplicación 01.**

  Considerando el elemento unidimensional de dos nodos que se presentan en el espacio natural en la figura ,obtenga las funciones de forma mediante el polinomios de Lagrange.

- **Solución**:

  Se considera la siguiente aproximación:

$$\widehat{u} = \sum_{a=1}^{n} N_a(\xi)\overline{u_a}^e \tag{4.119}$$

  Las funciones de forma se determinan mediante los polinomios de Lagrange:

$$N_1 = L_{1,1} = \frac{(\xi - \xi_2)}{(\xi_1 - \xi_2)} = \frac{(1-\xi)}{2} \tag{4.120}$$

$$N_2 = L_{1,2} = \frac{(\xi - \xi_1)}{(\xi_2 - \xi_1)} = \frac{(1+\xi)}{2} \tag{4.121}$$

  Puede observarse que estas funciones de forma son idénticas a las obtenidas por el método de álgebra lineal estudiadas ya anteriormente.

- **Ejemplo de Aplicación 02.**

  Considere el elemento unidimensional de tres nodos que se presentan en el espacio natural en la figura obtenga las funciones de forma mediante los polinomios de lagrange.

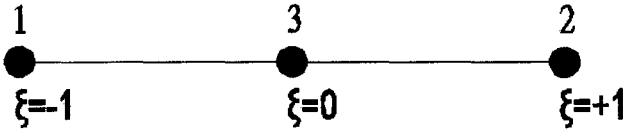


Figura 4.16: Elemento Unidimensional de tres nodos.

- **Solución:** Se considera la siguiente aproximación:

$$\widehat{u} = \sum_{a=1}^{n} N_a(\xi)\overline{u}_a^e \tag{4.122}$$

Las funciones de forma se presentan mediante los polinomios de Lagrange.

$$N_1 = L_{2,1} = \frac{(\xi-\xi_2)(\xi-\xi_3)}{(\xi_1-\xi_2)(\xi_1-\xi_3)}$$
$$= \frac{(\xi-1)(\xi-0)}{(-1-1)(-1-0)} = \frac{1}{2}\xi(\xi-1) \tag{4.123}$$
$$N_2 = L_{2,2} = \frac{(\xi-\xi_1)(\xi-\xi_3)}{(\xi_2-\xi_1)(\xi_2-\xi_3)}$$
$$= \frac{(\xi+1)(\xi-0)}{(+1+1)(+1-0)} = \frac{1}{2}\xi(\xi+1) \tag{4.124}$$
$$N_3 = L_{2,3} = \frac{(\xi-\xi_1)(\xi-\xi_2)}{(\xi_3-\xi_1)(\xi_3-\xi_2)}$$
$$= \frac{(\xi+1)(\xi-1)}{(+0+1)(+0-1)} = 1(1-\xi^2) \tag{4.125}$$

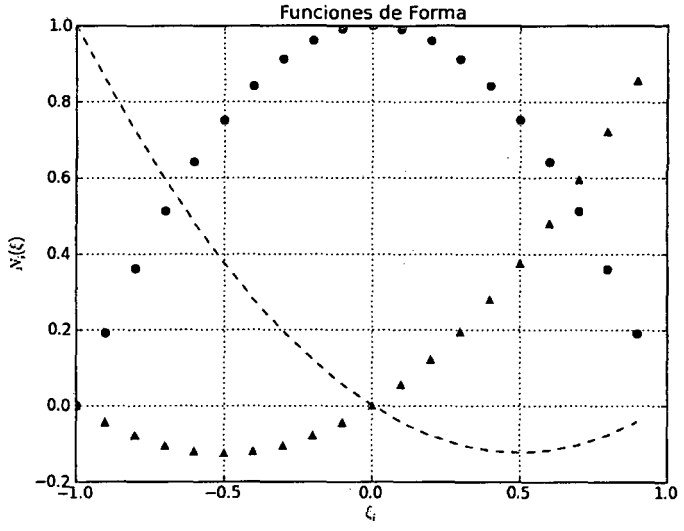


Figura 4.17: Funciones de Forma Cuadrática Lagrangiana.

```
#GRAFICO DE FUNCIONES DE FORMA DE SEGUNDO GRADO
from numpy import*
from matplotlib import*
from pylab import*
import matplotlib.pylab as plt
def N1(x):
    return 0.5*x*(x-1)
def N2(x):
    return 0.5*x*(x+1)
def N3(x):
    return (1-x**2)
x=arange(-1,1,0.1)
figure(1)
title('Funciones de Forma')
xlabel(r'$\xi_{i}$')
ylabel(r'$N_{i}(\xi)$')
grid(True)
plot(x,N1(x),'r--',x,N2(x),'g^',x,N3(x),'bo')
show()
```

Para interpolar en dos y tres dimensiones,simplemente se mezclan los productos de las interpolaciones unidimensionales de las direcciones correspondientes .Para dos dimensiones si se tienen $(n+1)$ puntos en ξ y$(m+1)$ puntos en η,entonces las funciones de forma se calcularan mediante: [4, Pag 109]

$$N_a(\xi,\eta) = L_{n,b}(\xi) x L_{m,c}(\eta) \tag{4.126}$$

- **Ejemplo de Aplicación 03.**

  Considere el elemento bidimensional de cuatro nodos que se presenta en el espacio natural en la figura obtenga las funciones de forma mediante combinaciones de polinomio de Lagrange.

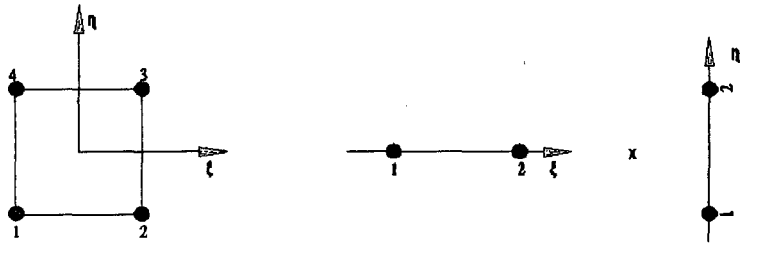


Figura 4.18: Elemento bidimensional de cuatro nodos.

- Solución.

  Se considera la siguiente aproximación :

$$\widehat{u} = \sum_{a=1}^{n} N_a(\xi,\tau)\overline{u}_a^e \tag{4.127}$$

Las funciones de forma se determinan mediante interpolación de Lagrange unidimensionalmente en las direcciones ξ y η respectivamente.Debe notarse que,en cada dirección,se tiene dos estaciones de interpolación por lo que es posible descomponer la interpolación bidimensional en dos interpolaciones unidimensionales como se muestran en la figura.

La siguiente tabla se emplea para relacionar la numeración nodal del elemento bidimensional con la numeración nodal de las interpolaciones unidimensionales: La tabla anterior se emplea para relacionar la numeración

n	$L(\xi)$	$L(\eta)$
1	1	1
2	2	1
3	2	2
4	1	2

Cuadro 4.3: Ordenamiento de elementos en dos dimensiones.

nodal del elemento bidimensional con la numeración nodal de las interpolaciones unidimensionales.

Las funciones de forma del elemento bidimensional se encuentran aplicando la ecuación:

$$N_1 = L_{1,1}(\xi)xL_{1,1}(\eta) = \frac{1}{2}(1-\xi)\frac{1}{2}(\eta-1) = \frac{1}{4}(1-\xi)(1-\eta) \quad (4.128)$$

$$N_2 = L_{1,2}(\xi)xL_{1,1}(\eta) = \frac{1}{2}(\xi+1)\frac{1}{2}(\eta-1) = \frac{1}{4}(1+\xi)(1-\eta) \quad (4.129)$$

$$N_3 = L_{1,2}(\xi)xL_{1,2}(\eta) = \frac{1}{2}(\xi+1)\frac{1}{2}(\eta+1) = \frac{1}{4}(1+\xi)(1+\eta) \quad (4.130)$$

$$N_4 = L_{1,1}(\xi)xL_{1,2}(\eta) = \frac{1}{2}(1-\xi)\frac{1}{2}(\eta+1) = \frac{1}{4}(1-\xi)(1+\eta) \quad (4.131)$$

Estas son las funciones de forma halladas anteriormente por el análisis de Álgebra lineal.

- El siguiente código esta hecho en python v.2.7 la cual esta usando las librerías de ipython(librería de cálculo algebraico) y numpy(librería de calculo de Matrices),la cual nos resuelve la interpolación de lagrange en n puntos.

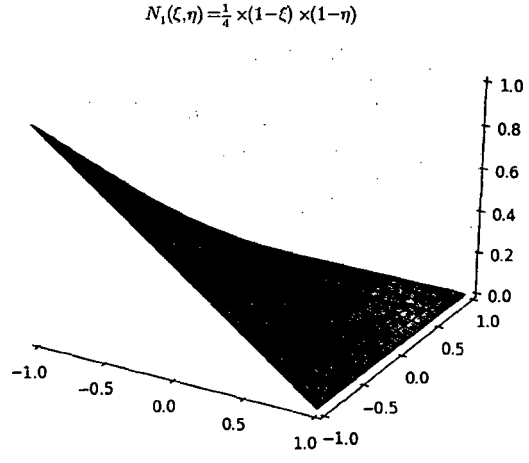


Figura 4.19: Discretización del elemento bidimensional de cuatro nodos N_1

```
#PROGRAMA DE FUNCIONES DE FORMA LINEALES.
from numpy import *
from sympy import *
def Forma_X(n,o):
    x=Symbol('x')
    lista_datos=arange(-1,1,2/float(n))
    lista_final01=lista_datos.tolist()
    lista_final02=lista_final01+[1.00]
    producto01=1
    producto02=1
    numero_elegido=lista_final02[o]
    lista_final02.pop(o)
    lista_definitiva=lista_final02
    for i in range(len(lista_definitiva)):
        producto01=producto01*(x-lista_final02[i])
    for i in range(len(lista_definitiva)):
        producto02=producto02*
        (numero_elegido-lista_definitiva[i])
    return together(producto01/(producto02))
```

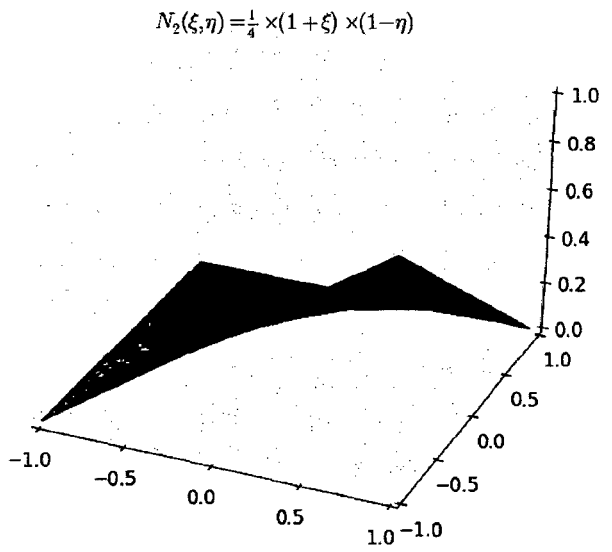


Figura 4.20: Discretización del elemento bidimensional de cuatro nodos N_2

- **Ejemplo 04.**

  Considere el elemento Bidimensional de nueve nodos que se presenta en el espacio natural en la siguiente figura,obtener las funciones de forma mediante combinaciones de Lagrange. Se considera la siguiente aproximación :

$$\widehat{u} = \sum_{a=1}^{n} N_a(\xi,\eta)\overline{u}_a^e \tag{4.132}$$

  Las funciones de forma se determinan mediante interpolación de Lagrange unidimensionales en las direcciones ξ y η respectivamente.Debe notarse que,en cada dirección,se tiene ahora tres estaciones de interpolación.La interpolación bidimensional se compone de las dos interpolaciones unidimensionales que se mostró en la figura.

  La siguiente tabla se emplea para relacionar la numeración nodal del elemento bidimensional con la numeración nodal de la interpolación unidimensional. Luego analizaremos las funciones de forma para cada nodo,la

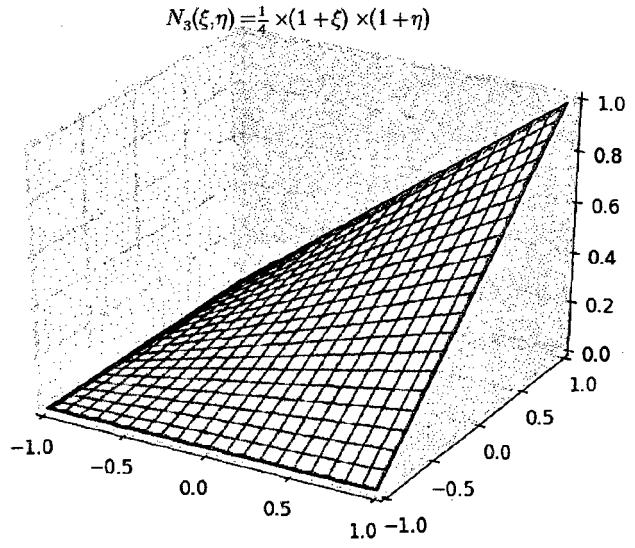


Figura 4.21: Discretización del elemento bidimensional de cuatro nodos N_3

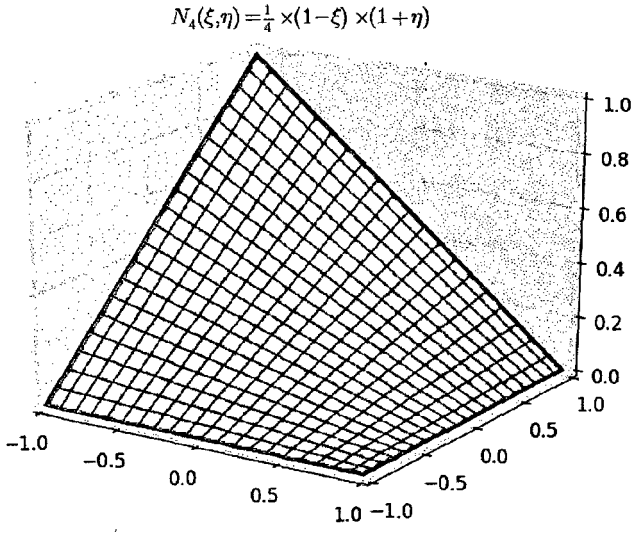


Figura 4.22: Discretización del elemento bidimensional de cuatro nodos N_4

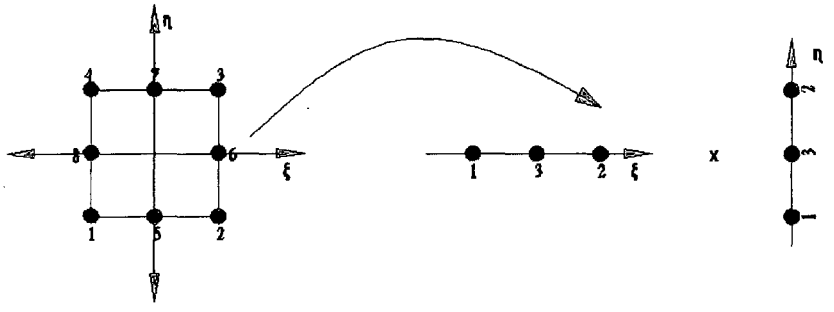


Figura 4.23: Elemento Bidimensional de nueve nodos.

n	$L(\xi)$	$L(\eta)$
1	1	1
2	2	1
3	2	2
4	1	2
5	3	1
6	2	3
7	3	2
8	1	3
9	3	3

Cuadro 4.4: Tabla nodal del problema de nueve nodos.

cual se analizaría el espacio para calculo de interpolación nodal general

del problema de nueve nodos antes mencionados.

$$N_1(\xi,\eta) = L_{2,1}(\xi)xL_{2,1}(\eta) = \frac{1}{2}(\xi)(\xi-1)\frac{1}{2}(\eta)(\eta-1) = \frac{1}{4}(\xi\eta)(\xi-1)(\eta-1) \quad (4.133)$$

$$N_2(\xi,\eta) = L_{2,2}(\xi)xL_{2,1}(\eta) = \frac{1}{2}(\xi)(\xi+1)\frac{1}{2}(\eta)(\eta-1) = \frac{1}{4}(\xi\eta)(\xi+1)(\eta-1) \quad (4.134)$$

$$N_3(\xi,\eta) = L_{2,2}(\xi)xL_{2,2}(\eta) = \frac{1}{2}(\xi)(\xi+1)\frac{1}{2}(\eta)(\eta+1) = \frac{1}{4}(\xi\eta)(\xi+1)(\eta+1) \quad (4.135)$$

$$N_4(\xi,\eta) = L_{2,1}(\xi)xL_{2,2}(\eta) = \frac{1}{2}(\xi)(\xi-1)\frac{1}{2}(\eta)(\eta+1) = \frac{1}{4}(\xi\eta)(\xi-1)(\eta+1) \quad (4.136)$$

$$N_5(\xi,\eta) = L_{2,3}(\xi)xL_{2,1}(\eta) = (1-\xi^2)\frac{1}{2}(\eta)(1-\eta) = \frac{1}{2}(1-\xi^2)(\eta)(1-\eta) \quad (4.137)$$

$$N_6(\xi,\eta) = L_{2,2}(\xi)xL_{2,3}(\eta) = (1-\eta^2)\frac{1}{2}(\xi)(1+\xi) = \frac{1}{2}(1-\eta^2)(\xi)(1+\xi) \quad (4.138)$$

$$N_7(\xi,\eta) = L_{2,3}(\xi)xL_{2,2}(\eta) = (1-\xi^2)\frac{1}{2}(\eta)(1+\eta) = \frac{1}{2}(1-\xi^2)(\eta)(1+\eta) \quad (4.139)$$

$$N_8(\xi,\eta) = L_{2,1}(\xi)xL_{2,3}(\eta) = (1-\eta^2)\frac{1}{2}(\xi)(1-\xi) = \frac{1}{2}(1-\eta^2)(\xi)(1-\xi) \quad (4.140)$$

$$N_9(\xi,\eta) = L_{2,3}(\xi)xL_{2,3}(\eta) = (1-\xi^2)(1-\eta^2) = (1-\xi^2)(1-\eta^2) \quad (4.141)$$

Los elementos formados así generan familias de elementos Lagrangianos,una de las familias de elementos estándar,que se presentan en la siguiente figura: [4]

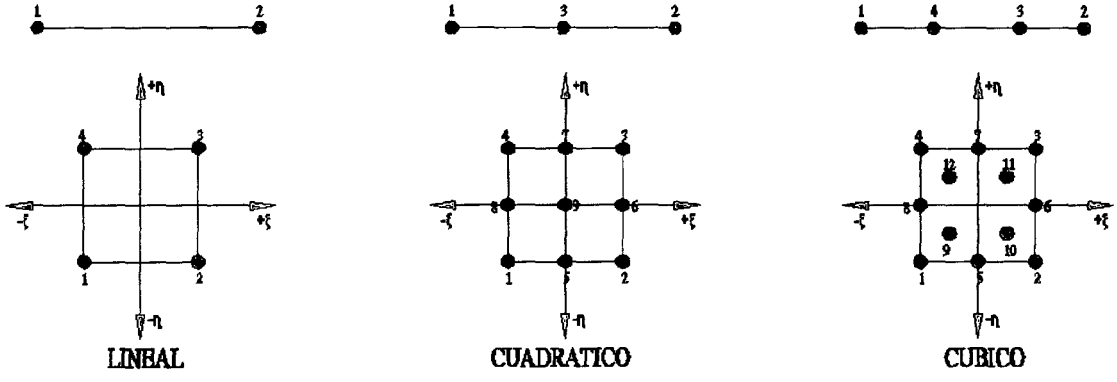


Figura 4.24: Elementos de Discretización lineal y bidimensional.

4.3.3. Elementos de Transición.

Considere el caso en que una parte de la malla generada para modelar un problema bidimensional se ha formado con elementos biliniales de cuatro nodos,y otra parte con elementos cuadráticos de 8 nodos.Esto puede haberse hecho para obtener un mayor grado de precisión de la región donde se presentan un cambio rápido en el valor de la función que se busca,sin tener que refinar en exceso las zonas en las que el valor de la función es prácticamente constante.En la figura se muestra la zona en donde las dos mallas se unen. [4, Pag 114]

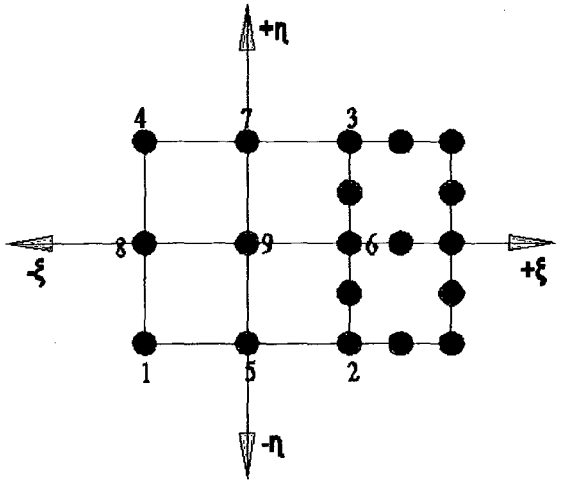


Figura 4.25: Transición de Malla Cuadrática.

El problema que se presentan en esta unión que la interpolación no será continua a través de la frontera entre elementos disimilares, y por tanto,se pierde la convergencia hacia la solución correcta.Este problema no se eliminaría empleando elementos lineales más pequeños cuyos nodos coincidieran con lo nodos intermedios de la malla cuadrática,porque la interpolación sobre el lado de los elementos de cuatro nodos sigue siendo lineal,mientras que,sobre el lado de los elementos de ocho nodos,sigue siendo parabólica.Una forma de resolver este tipo de conexiones es mediante el uso de elementos de transición,como el que se muestra en la figura,en donde tres de los lados son lineales mientras que el otro es cuadrático. [4, PAg 114]

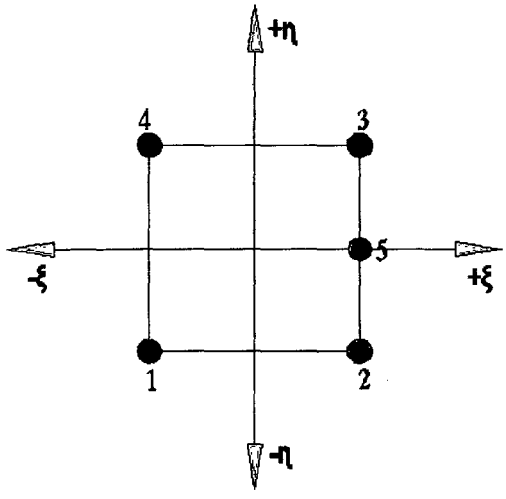


Figura 4.26: Elemento de transición lineal cuadrático.

La función de forma correspondiente al nodo 5 se puede obtener al mezclar interpolación lagrangianas lineales en ξ y cuadráticas en η,esto es:

$$N_5(\xi,\eta) = L_{1,2}(\xi) x L_{2,3}(\eta) \tag{4.142}$$

$$N_5(\xi,\eta) = \frac{1}{2}(1+\xi)(1-\eta^2) \tag{4.143}$$

Esta función de forma cumple con los requisitos básicos de la interpolación

nodal.

$$N_5(\xi,\eta) = \begin{cases} 0 & j = 1,2,3,4 \\ \\ 1 & j = 6 \end{cases} \tag{4.144}$$

Las funciones de forma de 1 al 4 son funciones bilineales ,las cuales ,al evaluarse en la posición del nuevo nodo 6, valen:

$$N_1 = N_2 = 0 \tag{4.145}$$

$$N_2 = N_3 = \frac{1}{2} \tag{4.146}$$

Las funciones de forma de los nodos 2 y 3 no cumplen con el requisitos de desvanecerse en la posición de nodo 5;por lo tanto, es necesario modificarlas para que satisfagan el requisito.Una forma de realizar esta modificación consiste en sustraer la función N_5 escalada adecuadamente.De esta manera,las funciones de forma para los cuatro nodos de esquina quedaría:[4, Pag 116]

$$N_1 = \frac{1}{4}(1-\xi)(1-\eta) \tag{4.147}$$

$$N_2 = N_2 - \frac{1}{2}N_5 \tag{4.148}$$

$$N_2 = \frac{1}{4}(1+\xi)(1-\eta) - \frac{1}{4}(1+\xi)(1-\eta^2) N_2 = \frac{1}{4}(1+\xi)\eta(\eta-1) \tag{4.149}$$

$$N_3 = N_3 + \frac{1}{2}N_5 \tag{4.150}$$

$$N_3 = \frac{1}{4}(1+\xi)(1+\eta) - \frac{1}{4}(1+\xi)(1-\eta^2) \tag{4.151}$$

$$N_3 = \frac{1}{4}(1+\xi)\eta(\eta+1) \tag{4.152}$$

$$N_4 = \frac{1}{4}(1+\eta)(1-\xi) \tag{4.153}$$

4.3.4. Elementos Triangulares y tetahedricos de clase C^0.

En secciones anteriores de este capitulo,se ha encontrado que mediante la técnica de degeneración es posible crear elementos triangulares y tetrahedricos a

partir de cuadriláteros y hexaedros,respectivamente.Se hizo la observación de que la generación produce mapeos que no son uno a uno en los nodos o lados en donde se colapsan los elementos originales, y por lo tanto, se pierden la convergencia de la solución sin embargo,puesto que las ecuaciones de elementos finitos se evalúan numéricamente en puntos interiores de los elementos,en donde siempre estarán bien definidos los mapeos,se ha encontrado que estos elementos producen también convergencia a la solución correcta en el límite.No obstante, es adecuado desarrollar elementos de este tipo en un sistema diferente de coordenadas,llamadas coordenadas triangulares o barométricas debido a que producen interpolaciones más simples que las que producen en coordenadas rectangulares como se observan a continuación.[4, Pag 118]

Elemento Triangular de 3 nodos.

La configuración del elemento se muestra en la figura ,se muestra la definición de un nuevo sistema de coordenadas ,denominadas coordenadas triangulares ,que puedan definirse con base en las sub-áreas que se generan al trazar lineas de los vértices al punto en cuestión.Se definen tres coordenadas para el punto: [4, Pag 118]

$$r = \frac{A_1}{A} \quad s = \frac{A_2}{A} \quad t = \frac{A_3}{A} \quad 0 \leq r, s, t \leq 1 \tag{4.154}$$

Cada terna de coordenadas define una forma única a cualquier punto del trián-

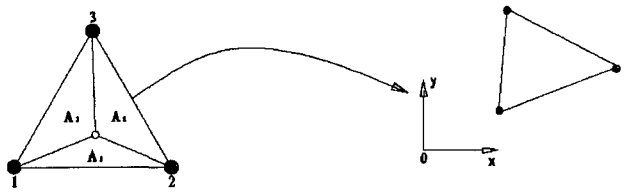


Figura 4.27: Elemento Triangular de tres nodos definido en coordenadas triangulares.

gulo.Siendo una figura plana,parece equívoco que se repitan tres coordenadas

para definir un punto;esto es correcto,ya que en realidad las coordenadas no son independientes,sino que están ligadas por la ecuación de restricción:

$$r + s + t = 1 \tag{4.155}$$

Se observa que el rango de valores para cada coordenada va desde cero hasta uno.Dada la posición de los nudos en los vértices del triangulo, es posible emplear directamente estas coordenadas como funciones como muestran en las siguientes ecuaciones:

$$x = N_1\overline{x_1^e} + N_2\overline{x_2^e} + N_3\overline{x_3^e} \tag{4.156}$$

$$y = N_1\overline{y_1^e} + N_2\overline{y_2^e} + N_3\overline{y_3^e} \tag{4.157}$$

$$\tag{4.158}$$

con:

$$N_1 = r \quad N_2 = s \quad N_3 = t \tag{4.159}$$

La aproximación a la solución se propone siguiendo el mismo formato:

$$\widehat{u} = N_1\overline{u_1}^e + N_2\overline{u_2}^e + N_3\overline{u_3}^e \tag{4.160}$$

Debe recordarse que solo se emplean realmente dos coordenadas independientes,por lo que se utiliza la ecuación como una ecuación de restricción para una de las variables,digamos t.

$$t = 1 - r - s \tag{4.161}$$

La convergencia en este caso,se asegura,como en los elementos cuadriliterales,verificando que hay continuidad y suavidad en las funciones de interpolación.En particular,se tendrá que verificar que el determinante del jacobiano de la transformación es positivo.Debido a la dependencia entre las variables,dada por la ecuación: [4, Pag 120]

$$J = \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} \\ \\ \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} \end{bmatrix} \tag{4.162}$$

Interpolación de Elementos Triangulares.

Se empleará la interpolación de lagrange para desarrollar interpolaciones de orden más alto que el lineal.Se define la interpolación para triángulos mediante.

$$L_{I-1,I}(r) = \prod_{i=1}^{I} \left(\frac{r - r_i}{r_I - r_i}\right) \text{ Para todo } : I \neq 1 \tag{4.163}$$

$$1 \text{ Para Valores de } : I = 1 \tag{4.164}$$

Para elementos en dos dimensiones,se hace el producto en tres interpolaciones,en dirección de las tres coordenadas triangulares ,formando una retícula de nodos como la que se muestra en la siguiente figura,de manera que la función de forma para un nodo a se expresa mediante:

$$N_a(r, s, t) = T_i(r)T_j(s)T_k(t) \tag{4.165}$$

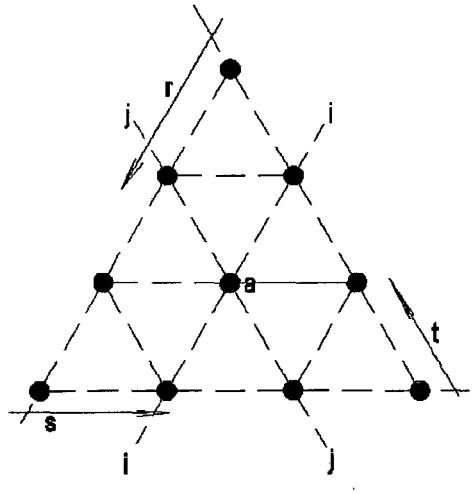


Figura 4.28: Patrón para generar elementos de alta jerarquía en coordenadas triangulares.

- **Ejemplo 01.**

Como ejemplo,las funciones de forma del elemento triangular de tres nodos

que se muestran en la figura se obtienen con dos estaciones a lo largo de cada coordenada.Para las funciones de interpolación se aplica EC(163),EC(164) resultan las mismas funciones que antes:

$$N_1 = T_2(r) \times T_1(s) \times T_1(t) = \frac{r - r_i}{r_2 - r_1} = r \tag{4.166}$$

$$N_2 = T_1(r) \times T_2(s) \times T_1(t) = \frac{s - s_1}{s_2 - s_1} = s \tag{4.167}$$

$$N_3 = T_1(r) \times T_1(s) \times T_2(t) = \frac{s - s_1}{s_2 - s_1} = t \tag{4.168}$$

- **Ejemplo 02.**

Hallar las funciones de forma por el metodo antes mencionado de la siguiente forma Triangular: Este elemento se presentan en la figura ,en ella se observa

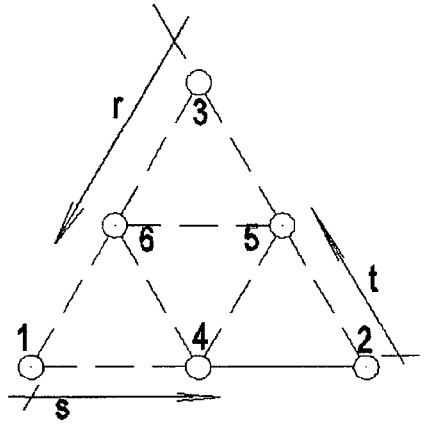


Figura 4.29: Elemento Triangular de 6 nodos.

que se tiene 3 estaciones en cada dirección de cada coordenada.

Las funciones de forma son: [4, Pag 124]

$$N_1(r,s,t) = T_3(r) \times T_1(s) \times T_1(t) = \frac{(r-r_1)(r-r_2)}{(r_3-r_1)(r_3-r_2)} = r(2r-1) \quad (4.169)$$

$$N_2(r,s,t) = T_3(s) \times T_1(r) \times T_1(t) = \frac{(s-s_1)(s-s_2)}{(s_3-s_1)(s_3-s_2)} = s(2s-1) \quad (4.170)$$

$$N_3(r,s,t) = T_3(t) \times T_1(r) \times T_1(s) = \frac{(t-t_1)(t-t_2)}{(t_3-t_1)(t_3-t_2)} = t(2t-1) \quad (4.171)$$

$$N_4(r,s,t) = T_2(s) \times T_2(r) \times T_1(t) = \frac{s-0}{0{,}5-0} \times \frac{r-0}{0{,}5-0} \times 1 = 4rs \quad (4.172)$$

$$N_5(r,s,t) = T_2(s) \times T_2(s) \times T_1(r) = \frac{s-0}{0{,}5-0} \times \frac{t-0}{0{,}5-0} \times 1 = 4st \quad (4.173)$$

$$N_6(r,s,t) = T_2(r) \times T_2(t) \times T_1(r) = \frac{r-0}{0{,}5-0} \times \frac{t-0}{0{,}5-0} \times 1 = 4rt \quad (4.174)$$

(4.175)

4.4. Cálculo Analítico sobre Elementos Triangulares y Rectangulares de lados rectos.

La integración de polinomios sobre el área de elementos bidimensionales puede ser laboriosa y en general se hace uso de integración numérica.No obstante,en el caso de elementos rectangulares o triangulares de lados rectos existen expresiones analíticas de gran utilidad práctica así,expresando los términos del integrando de las diferentes matrices en función de las coordenadas cartesianas locales $\overline{x}$ y $\overline{y}$ que se muestran en la figura ,puede obtenerse que la integral de un término típico ,tal como.

$$\overline{K_{ij}} = D \int \int_{A^e} \overline{x}^m \overline{y}^n dA \quad (4.176)$$

venga dada por las expresiones siguientes:

$$\overline{K_{ij}} = Dc^{n+1} \left[a^{m+1} - (-b)^{m+1}\right] \frac{m!n!}{(m+n+2)!} \longrightarrow \text{Elementos Triangulares} \quad (4.177)$$

$$\overline{K_{ij}} = D\frac{(2a)^{m+1}(2b)^{n+1}}{(m+1)(n+1)} \longrightarrow \text{Elementos Rectangulares} \quad (4.178)$$

En las ecuaciones anteriores m y n son enteros, a,b y c dimensiones típicas del elemento de la figura ; y D una constante que depende del tipo de elemento,de las propiedades del material y de las derivadas de las funciones de forma.Una vez obtenida la correspondiente matriz $\overline{K}$ o el vector $\overline{f}$ en coordenadas locales $\overline{x}$,$\overline{y}$,pueden transformarse a ejes globales mediante la clásica expresiones:

$$K_{ij} = T^T \overline{K_{ij}} T, f_i = T^T \overline{f}_i \tag{4.179}$$

donde T es la matriz de cambio de ejes:

$$T = \begin{bmatrix} cos(\overline{x}x) & cos(\overline{x}y) \\ \\ cos(\overline{y}x & cos(\overline{y}y) \end{bmatrix} \tag{4.180}$$

siendo$(\overline{x}x)$ el ángulo que forma el eje $\overline{x}$ con el eje global x,etc. El cálculo de

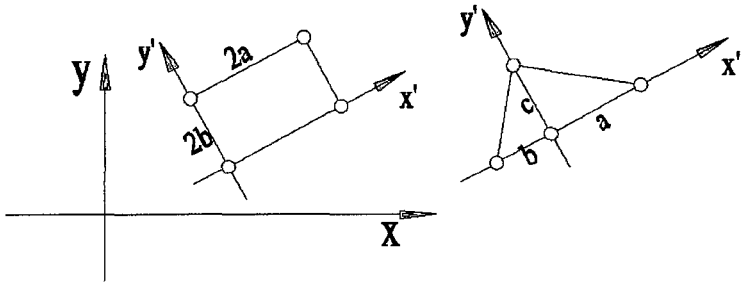


Figura 4.30: Coordenadas x',y' para el cálculo analítico de las integrales de elementos triangulares y rectangulares.

la matriz de rigidez en elementos triangulares exige obtener derivadas de las funciones de forma expresadas en coordenadas de área es decir (r, s, t),con respecto a las coordenadas cartesianas.Así por ejemplo [3]

$$\frac{\partial(r,s,t)}{\partial x} = \frac{\partial N_1}{\partial r}\frac{\partial r}{\partial x} + \frac{\partial N_1}{\partial s}\frac{\partial s}{\partial x} + \frac{\partial N_1}{\partial t}\frac{\partial t}{\partial x} \tag{4.181}$$

Si el elemento triangular es de lados rectos se deduce que:

$$\frac{\partial(r,s,t)}{\partial x} = \frac{b_i}{2A_e} \text{ y } \frac{\partial(r,s,t)}{\partial y} = \frac{c_i}{2A_e} \tag{4.182}$$

pudiendo obtenerse fácilmente una expresión de los integrados de la matriz de rigidez en función de las coordenadas de área.Dichas integrales pueden evaluarse directamente por las expresiones siguientes:

$$\int\int_{A^e} r^k s^l t^m dA = 2A^e \frac{k! \times l! \times m!}{(2+k+l+m)!} \tag{4.183}$$

$$\oint_{l(e)} r^k s^l ds = l^e \frac{k! \times l!}{(1+k+l)!} \tag{4.184}$$

Si en las integrales anteriores no aparece alguna de las coordenadas de área,simplemente se omite el correspondiente exponente en los denominadores de los segundos miembros de la ecuación anterior y se hace igual a la unidad de los numeradores. [3, Pag 214,215,216]

Capítulo 5

SÓLIDOS DE REVOLUCIÓN.

5.1. Introducción.

En este capitulo estudiaremos el análisis por método de los elementos finitos de sólidos con simetría axial.Es decir,consideremos sólidos en los que su geometría y propiedades mecánicas son independientes de la coordenada circunferencial θ_i.Aunque el comportamiento de dichos sólidos es tridimensional,pero su estudio es generalmente bidimensional ya que en la mayoría de los casos puede efectuarse utilizando variables que dependen únicamente de dos coordenadas cartesianas. [3, Pag 245]

Si las cargas exteriores son también de revolución,el desplazamiento de un punto de una estructura considerada como sólido de revolución tiene solo componentes en direcciones radial (u) y el axial (w).El estudio de dichas estructuras por elementos finitos no es complicado y sigue prácticamente los mismos pasos que se explicaron en el capitulo anterior para problemas de elasticidad bidimensional.Si las cargas no son de revolución hay que realizar un análisis tridimensional.

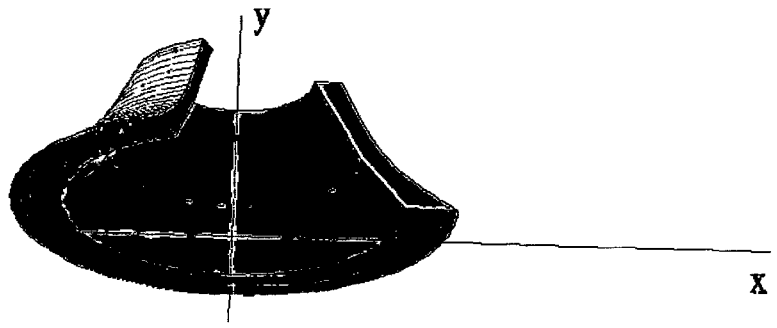


Figura 5.1: Sólido de Revolución.

5.2. Formulación Básica.

5.2.1. Campo de Desplazamiento.

Sea el sólido de revolución de la figura.Si las cargas son también de revolución el movimiento de un punto queda perfectamente definido por las componentes de los desplazamientos radiales u y el axial w,siendo nula la componente circunferencial ν.Por consiguiente,se puede definir el factor desplazamiento de un punto por:

$$u = \left\{ \begin{array}{c} u(r,s) \\ w(r,s) \end{array} \right\} \tag{5.1}$$

5.2.2. Campo de Deformaciones.

Debido a la simetría axial del problema los dos desplazamientos no nulos u y w son independientes de las coordenadas circunferencial θ.Por consiguiente se deduce de inmediato que las deformaciones tangenciales $\gamma_{r\theta}$ y $\gamma_{z\theta}$ son nulas.Asimismo,de la teoría de elasticidad se obtiene.

$$\varepsilon_r = \frac{\partial u}{\partial r} : \varepsilon_z = \frac{\partial w}{\partial z} : \gamma_{rz} = \frac{\partial u}{\partial z} + \frac{\partial w}{\partial r} \tag{5.2}$$

siendo ε_r,ε_z y σ_{rz} las deformaciones radial,axial,tangencial respectivamente. Por otra parte la deformación axial del cuerpo provoca que los puntos situados sobre una circunferencia de radio r pasen después de la deformación a estar situados sobre otra de radio $r+u$.Por ello,se define la deformación circunferencial ε_θ como la variación relativa de longitud entre dichas circunferencias Es decir:

$$\varepsilon_\theta = \frac{2\pi(r+u)}{2\pi r} = \frac{u}{r} \tag{5.3}$$

El vector de deformaciones de un punto tiene,por tanto,las cuatro componentes siguientes:

$$\varepsilon = \left\{ \begin{array}{c} \varepsilon_r \\ \\ \varepsilon_z \\ \\ \varepsilon_\theta \\ \\ \sigma_{rz} \end{array} \right\} = \left\{ \begin{array}{c} \frac{\partial u}{\partial r} \\ \\ \frac{\partial w}{\partial z} \\ \\ \frac{u}{r} \\ \\ \frac{\partial u}{\partial z} + \frac{\partial w}{\partial r} \end{array} \right\} \tag{5.4}$$

5.2.3. Campos de Tensiones.

Las tensiones no nulas se corresponden no nulas se corresponden con las deformaciones no nulas Así,pues el vector de tensiones se escribe como:

$$\sigma = \{\sigma_r, \sigma_z, \sigma_\theta, \tau_{rz}\} \tag{5.5}$$

donde $\sigma_r, \sigma_z, \sigma_\theta$ son,respectivamente ,las tensiones radial,axial y circunferencial, y γ_{rz}, es la tensión tangencial.

5.2.4. Ecuación Constitutiva.

la Relación entre tensiones y deformaciones se deduce de la elasticidad tridimensional de forma análoga al caso de elasticidad bidimensional.En presencia

de tensiones y deformaciones iniciales se obtienes: [4, Pag 246]

$$\sigma = D(\varepsilon - \varepsilon^0) + \sigma^0 \tag{5.6}$$

donde si el material es isotropo:

$$D = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 \\ \nu & 1-\nu & \nu & 0 \\ \nu & \nu & 1-\nu & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix} \tag{5.7}$$

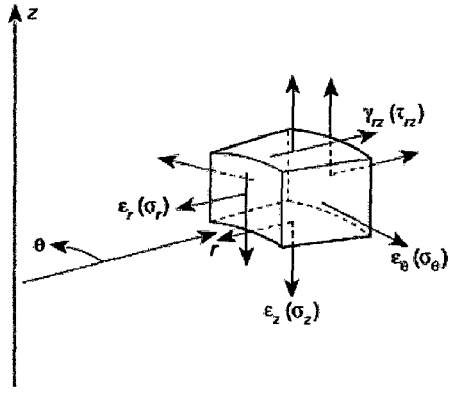


Figura 5.2: Tensiones actuando sobre un elemento diferencial de un solido de revolución.

5.2.5. Expresión del Principio de los Trabajos Virtuales.

La expresión del PTV(principio de trabajos virtuales) es análoga a la de elasticidad biimensional ,estando ahora todas las integrales referidas al volumen del

sólido de revolución.Se deduce que el diferencial de volumen se puede expresar como:

$$dV = (rd\theta)dr.dz = rd\theta dA \tag{5.8}$$

donde dA es la diferencial de área de la sección meridional del sólido .Así ,pues ,la expresión del PTV en un sólido de revolución se escribe como:

$$\int\int_A\int_0^{2\pi} \delta\varepsilon^T\sigma r d\theta dA = \int\int_A\int_0^{2\pi} \delta u^T b r d\theta dA + \oint\int_0^{2\pi} \delta u^T b r d\theta ds + \sum_i \int_0^{2\pi} \delta u_i^T q_i r_i d\theta$$

(5.9)

donde l es el contorno de la sección meridional y:

$$b = \begin{Bmatrix} b_r \\ \\ b_z \end{Bmatrix} : t = \begin{Bmatrix} t_r \\ \\ t_z \end{Bmatrix} \text{ y } q_i = \begin{Bmatrix} q_{ri} \\ \\ q_{zi} \end{Bmatrix} \tag{5.10}$$

Son los vectores de fuerzas exteriores másicas ,de superficie y puntuales ,respectivamente .Recordemos de nuevo que ahora todas las cargas tienen simetría de revolución.

Haciendo uso de la simetría axial puede efectuarse la integración sobre la variable circunferencial θ para dar:

$$2\pi\int\int_A \delta\varepsilon^T\sigma dA = 2\pi\int\int_A \delta u^T b r dA + 2\pi\oint_l \delta u^T r ds + 2\pi\sum_i \delta a_i^T q_i r_i \tag{5.11}$$

Se observa que el coeficiente 2π aparece multiplicando todos los términos ,pudiendo por lo tanto eliminarse.No obstante,conviene mantenerlo por razones didácticas,y,fundamentalmente ,para recordar que los valores de las cargas puntuales q_i se refieren a las intensidades de carga por unidad de longitud circunferencial. [3, Pag 249]

5.3. Formulación de Elementos Finitos.

Presentamos seguidamente la formulación de elementos finitos utilizando,en primer lugar,el elemento de sólido de revolución más sencillo que,análogamente al caso del plano, es el elemento triangular de revolución de tres nodos.El elemento,como se puede apreciar en la figura es un anillo de sección triangular.Hay que resaltar que un sólido de revolución los elementos son anulares ,aunque en virtud de la ecuaciones anteriores todas las integrales del elemento se evalúen únicamente sobre una sección meridional operándose,por tanto,sobre un elemento bidimensional. [3, Pag 250]

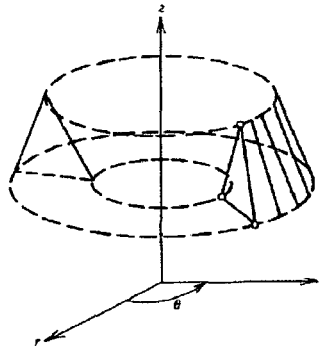


Figura 5.3: Elemento sólido de Revolución triangular de tres nodos.

5.3.1. Discretización del campo de desplazamientos.

Dentro de la Sección meridional de un elemento,el campo de desplazamientos se interpola en forma análoga al caso plano.Así,para el elemento triangular de

tres nodos.

$$u = \left\{ \begin{array}{c} u \\ \\ w \end{array} \right\} = \left[\begin{array}{ccc} N_1 \times u_1 & N_2 \times u_2 & N_3 \times u_3 \\ \\ N_1 \times w_1 & N_2 \times w_2 & N_3 \times w_3 \end{array} \right] \tag{5.12}$$

$$N = [N_1, N_2, N_3] = \left[\begin{array}{cccccc} N_1 & 0 & N_2 & 0 & N_3 & 0 \\ \\ 0 & N_1 & 0 & N_2 & 0 & N_3 \end{array} \right] \tag{5.13}$$

$$a^e = \left\{ \begin{array}{c} a_1 \\ \\ a_2 \\ \\ a_3 \end{array} \right\} = [u1, w1, u2, w2, u3, w3]^T \tag{5.14}$$

Las funciones de forma N_i se obtienen directamente de las del elemento triangular de tres nodos de elasticidad plana,sustituyendo simplemente las coordenadas x,y por r,z.La interpolación de las expresiones anteriores al caso de un elemento de n nodos.

5.3.2. Discretización del campo de deformaciones y tensiones.

Del Vector de deformaciones y las ecuaciones de obtienen.

$$\varepsilon = \sum_{i=1}^{3} \left[\begin{array}{cc} \frac{\partial N_i}{\partial r} & 0 \\ \\ 0 & \frac{\partial N_i}{\partial z} \\ \\ \frac{N_i}{r} & 0 \\ \\ \frac{\partial N_i}{\partial z} & \frac{\partial N_i}{\partial r} \end{array} \right] \left\{ \begin{array}{c} u_i \\ \\ w_i \end{array} \right\} = [B_1, B_2, B_3] \left\{ \begin{array}{c} a_1 \\ \\ a_2 \\ \\ a_3 \end{array} \right\} = Ba^e \tag{5.15}$$

siendo B la matriz deformada del elemento del nodos i.

Para un elemento de n nodos se tendría n submatrices B_i,observase que la matriz de deformación no es constante como en el caso de elasticidad plana debido al termino $\frac{N_i}{r}$ que contiene una singularidad en el origen de coordenadas .Más adelante comentaremos como puede evitarse este problema.

Para el elemento triangular de tres nodos se obtiene una forma más explicita de B_i,sustituyendo la expresión de las funciones de forma :

$$B_i = \frac{1}{2A}\begin{bmatrix} b_i & 0 \\ 0 & c_i \\ \frac{(a_i+b_i r+c_i z)}{r} & 0 \\ c_i & b_i \end{bmatrix} \tag{5.16}$$

El vector de tensiones se obtiene sustituyendo por :

$$\sigma = \sum_{i=1}^{3} B_i D a^{(e)} - D\varepsilon^0 + \sigma^0 \tag{5.17}$$

Durante el cálculo de las tensiones puede surgir problemas para encontrar el valor de $\varepsilon_\theta = \frac{u}{r}$ en puntos del eje z al aparecer el cociente indeterminado $\frac{0}{0}$.Este problema puede sortearse calculando el ε_θ en puntos ligeramente alejados def eje, o lo que es más usual extrapolando las tensiones de puntos del interior del elemento(que generalmente son los puntos de Gauss) al eje .Otro procedimiento muy sencillo es utilizar la propiedad de que $\varepsilon_\theta = \varepsilon_r$ en el eje , lo que simplemente implica reemplazar la segunda fila de B correspondiente a ε_θ por la primera que corresponde a ε_r.

5.3.3. Matriz de Rigidez del Elemento.

Partiendo del primer miembro de la expresión del PTV(principio de trabajos virtuales) particularizando para un elemento utilizado en la figura ,se hace el

análisis de manera idéntica a como se hizo en el caso de elasticidad plana ,se obtienen la expresión del equilibrio del elemento como.

$$K^e a^e - f^e = q^e \tag{5.18}$$

en la que la matriz de rigidez del elemento de la rigidez del elemento tiene la expresión siguiente:

$$K_{ij}^{(e)} = 2\pi \int \int_{A^{(e)}} B_i^T D B_j r dr dz \tag{5.19}$$

La expresión anterior es válida para cualquier elemento bidimensional de n nodos.En la figura se muestra la forma desarrollada K_{ij}^e para el elemento triangular de tres nodos .

5.3.4. Vectores de fuerzas nodales equivalentes.

De la expresión de los trabajos virtuales se obtienen el vector de fuerzas nodales equivalentes:

$$f^e = 2\pi \int \int_A N^T b r dA + 2\pi \oint_{l^e} N^T t r ds + 2\pi \int \int_A B^T D \varepsilon^0 r dA$$
$$- 2\pi \int \int_{A^e} B^T \sigma^0 r dA \tag{5.20}$$

Donde la primera integral corresponde al vector de fuerzas másicas de volumen f_b^e;la segunda a las fuerzas de superficie f_t^e;la tercera a la fuerza debida a las tensiones iniciales f_ε^e y la cuarta debida a tensiones iniciales f_σ^e. [3, Pag 257]

METODOLOGÍA DE ESTUDIO.

Capítulo 6

Metodología de Estudio.

La metodología de estudio para la tesis,fue teórico ya que se usó el método inductivo-deductivo,basándose en información hecha por investigaciones anteriores,para dar fruto a unos scripts en lenguaje python.Así como los métodos de acoplamiento de matrices de rigidez usándose para tal fin el sistema ordenamiento de la matriz π que es un sencillo algoritmo de acoplamiento general para todo tipo de matrices de rigidez.El análisis de fuerzas viene sometido al estudio de las funciones de forma las cuales son resultado de la demostración teórica de la formula de la deformada general de vigas la cual se puede esclarecer con las formulas obtenidas en el capitulo 3,específicamente en el apartado de análisis de flexión de vigas por la teoría Euler-Bernulli.

$$u = N_2 w_1 + N_3 \theta_1 + N_4 w_2 + N_5 \theta_2 + R_i(x, l) \tag{6.1}$$

A continuación se esquematiza el método de investigación ha empleado:

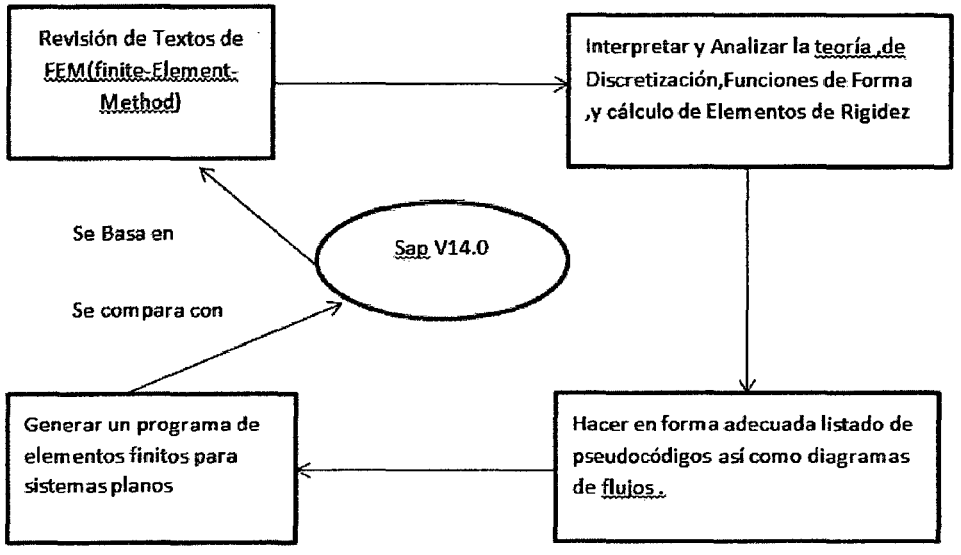


Figura 6.1: Esquema de Investigación.

EJEMPLOS,PRESENTACIÓN Y DISCUSIÓN DE RESULTADOS .

Capítulo 7

Ejemplos y Problemas de Elementos Finitos.

7.1. Vigas.

7.1.1. Problema N01:

Se tiene en la figura donde se trata de una barra continua de 0.30x0.30 m2 de Área y una E=1400000.00,nos piden resolver el sistema.

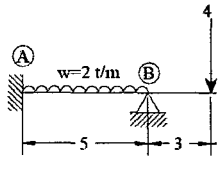


Figura 7.1: Primer Problema para uso del Programa Femax.

- Puesta de datos en el programa Materiales.

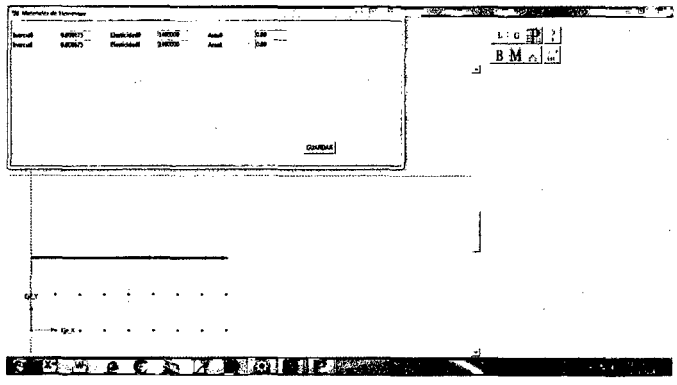


Figura 7.2: Puesta de Datos de Materiales.

- puesta total de datos:

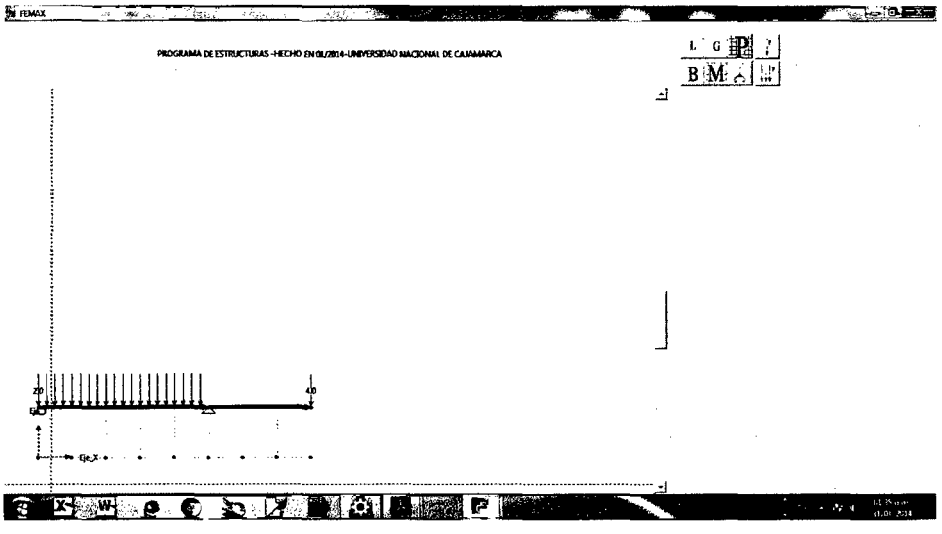


Figura 7.3: puesta de datos total del esquema a prueba.

- Puesta de Datos en el sap 2000 V14.
- El programa Femax te halla la matriz de rigidez,una ventaja sobre el SAP ya que el sap no te da la matriz,claro que en el fondo es un dato que en el diseño comercial es impráctico mas no en el entendimiento del método de elementos finitos.

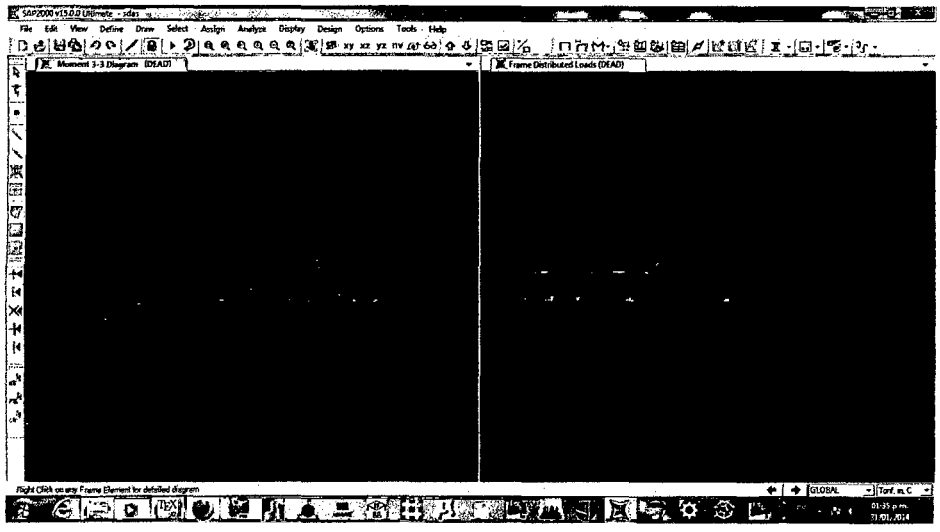


Figura 7.4: Puesta de datos en Sap 2000 V14.

25200	0.00	0.00	-25200	0.00	0.00	0.00	0.00	0.00
0.00	90.72	226.8	0.00	-90.72	226.80	0.00	0.00	0.00
0.00	226.8	756	0.00	-226.80	378.00	0.00	0.00	0.00
-25200	0.00	0.00	67200.00	0.00	0.00	-42000.00	0.00	0.00
0.00	-90.72	-226.8	0.00	510.72	403.2	0.00	-420.00	630
0.00	226.80	378.00	0.00	403.20	2016.00	0.00	-630.00	630.00
0.00	0.00	0.00	-42000.00	0.00	0.00	420000.00	0.00	0.00
0.00	0.00	0.00	0.00	-420.00	-630.00	0.00	420.00	630.00
0.00	0.00	0.00	0.00	630.00	630.00	0.00	-630.00	1260.00

Cuadro 7.1: Matriz de Rigidez General hallada por el programa Femax.

- También el programa calcula la matriz simplificada o matriz generada por los grados de libertad.

2016.00	0.00	-630.00	630.00
0.00	420000	0.00	0.00
-630.00	0.00	420.00	-630.00
630.00	0.00	-630.00	1260.00

Cuadro 7.2: Matriz simplificada copiada del Femax.

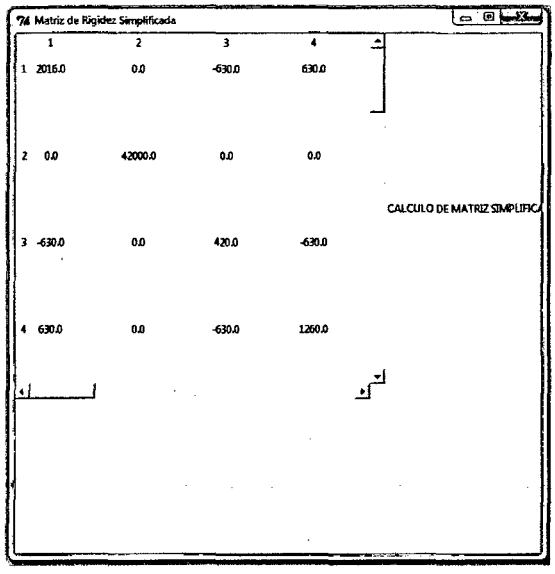


Figura 7.5: Matriz de Rigidez obtenida por Femax.

- Calculo de las deformaciones se dan por medio de la matriz de rigidez disminuida o simplificada $u_i = K_D^{-1} F_i$.
- Haciendo la respectiva comparación de los resultados del análisis de los desplazamientos totales en los nudos.
- Cálculo de las fuerzas internas en el sap V14 2000 y en el Femax.

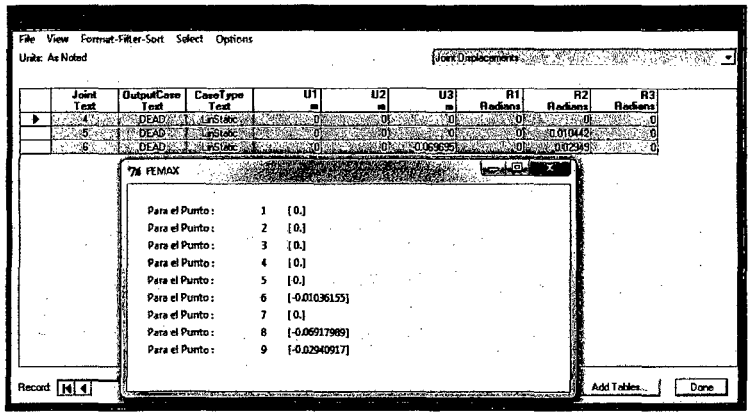


Figura 7.6: Comparación de Resultados de Desplazamientos.

R	SAP 2000 V14	Femax 1.01	%Comparación
Nudo A	0.01044200	0.01036155	99.22 %
Nudo B-1	0.069695	0.06917898	99.25 %
Nudo B-2	0.0294900	0.02940917	99.72 %

Cuadro 7.3: Tabla de Comparación del Calculo de los Desplazamientos.

- Cuadro de comparación y porcentaje de las mismas.

F	Femax	Sap 2000 V14	% Comparación
Fy_1	2.65	2.6561	99.77 %
Mz_1	0.25	0.28	89.28 %
Fy_2	11.34	11.39	99.56 %

Cuadro 7.4: Comparación de Resultados de Fuerzas Internas de Sap 2000 V14 y Femax.

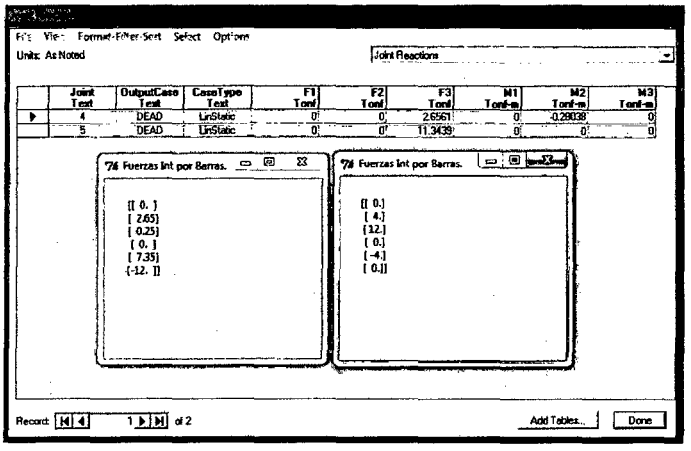


Figura 7.7: Comparación de Resultado de Fuerzas Int.

7.2. Pórticos.

7.2.1. Problema N02.

- Calcular el siguiente Marco con el Métodos de elementos finitos.
- E=1400000.00
- I=0.000675m^4
- Poner los datos y las condiciones iniciales del problema las cuales se dará en Femax y SAP al mismo tiempo para su respectiva comparación.

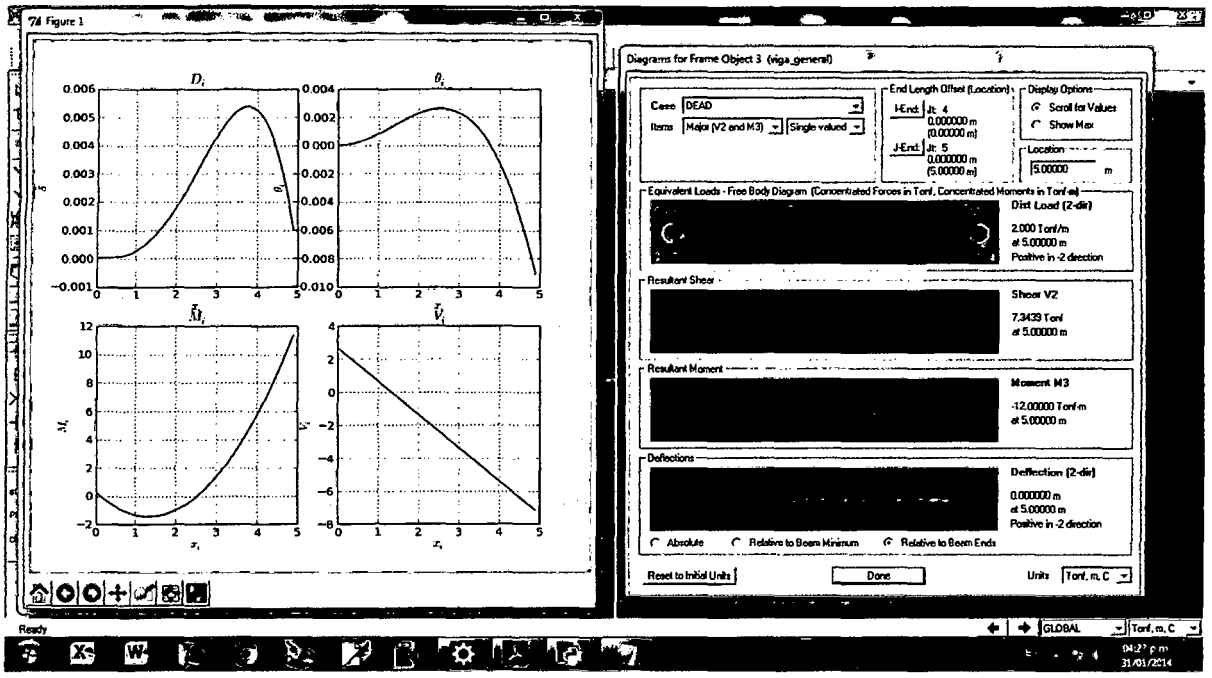


Figura 7.8: Comparación de Gráficos del Femax con Sap 2000 V14.

- Comparación de los desplazamientos en los nudos:

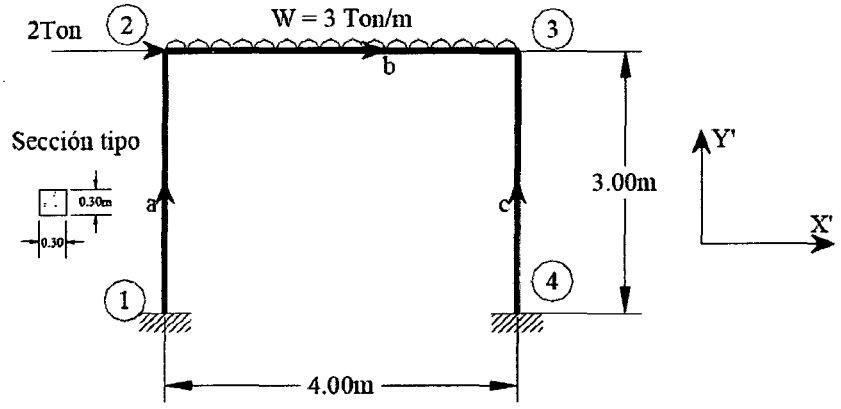


Figura 7.9: Esquema del Problema N02.

•	Sap 2000 V14	Femax	% Comparación
u2	0.0038200	0.00372745	97.57 %
v2	0.00012800	0.00012827	100.2 %
θ_2	0.003245	0.0031947	98.44 %
u3	0.003743	0.00364976	97.50 %
v3	0.00015745	0.00015745	100 %
θ_3	0.0014770	0.00145116	98.25 %

Cuadro 7.5: Tabla de comparación de las Deformaciones Sap y Femax.

- Veremos el Cálculo de las matrices de Rigideces y la matriz de rigidez simplificada.
- el cálculo de los resultados de las Fuerzas Internas.

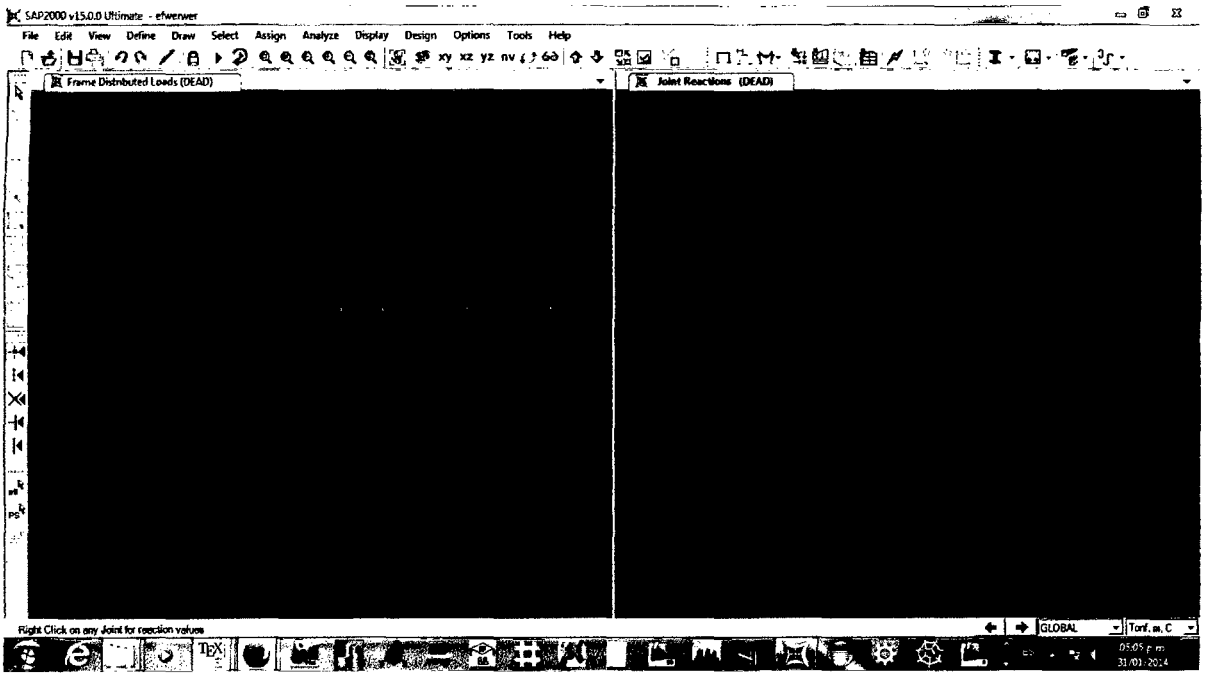


Figura 7.10: Puesta de Datos en Sap 2000 V14.

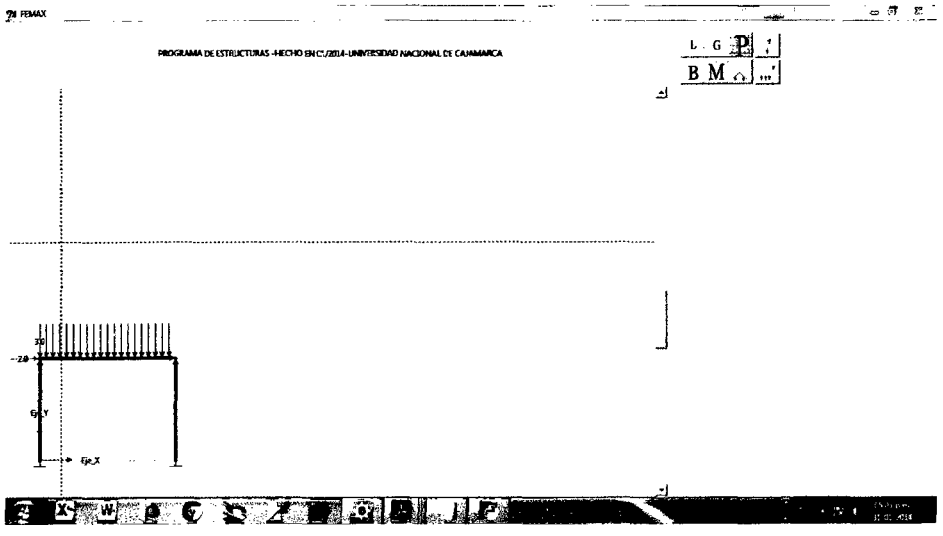


Figura 7.11: Puesta de Datos en Femax 1.01

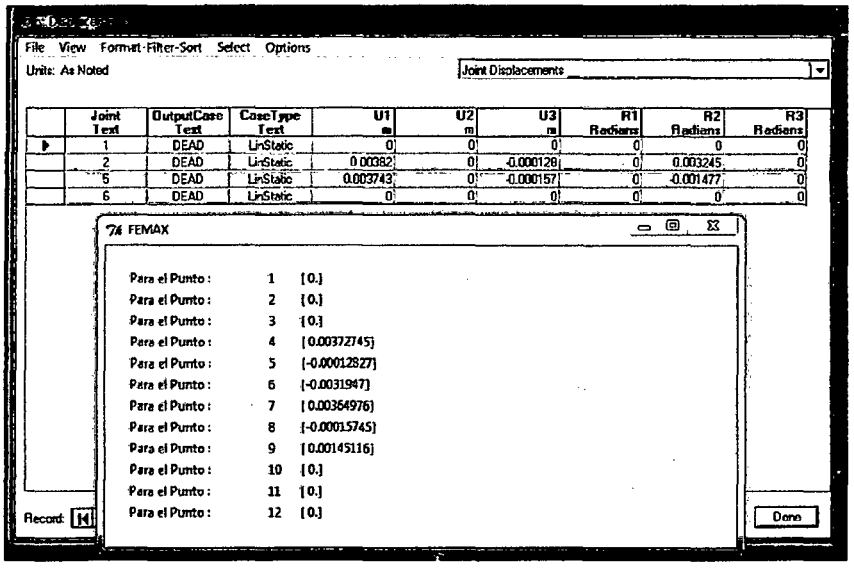


Figura 7.12: Comparación de los desplazamientos en Femax y Sap 2000 V14.

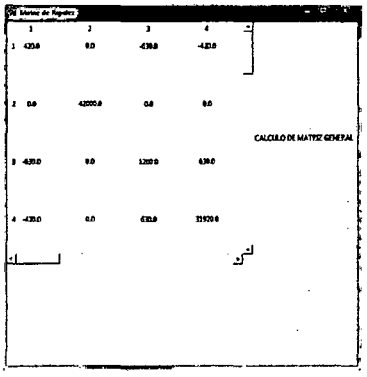


Figura 7.13: Matriz de Rigidez General del Esquema.

R	Sap 2000 V14.0	Femax	% Comparación.
Fx_1	0.427000	0.44712991	104.7 %
Fy_1	5.389200	5.387303570	99.96 %
Mz_1	0.38189	0.33563433	87.87 %
Fx_4	2.4270	2.4471	100.8 %
Fy_4	6.6108	6.6126	100.00 %
Mz_4	3.1751	3.2136	101.2 %

Cuadro 7.6: Tabla de Comparaciones de Calculo de Reacciones del esquema.

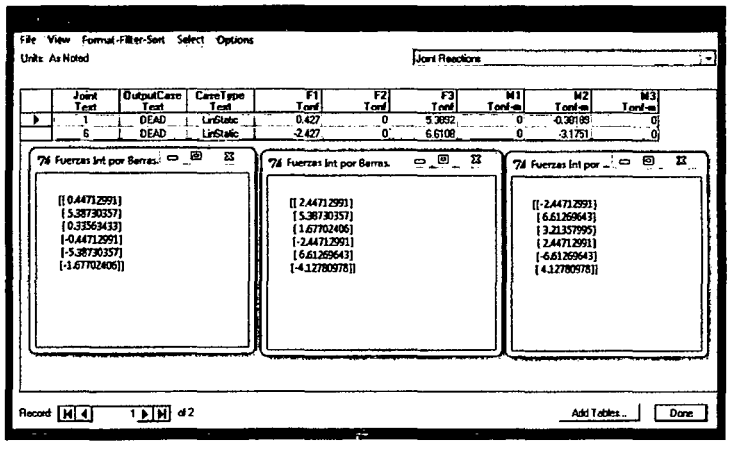


Figura 7.14: Comparación de Fuerzas Internas del Femax y el Sap 2000 V14.

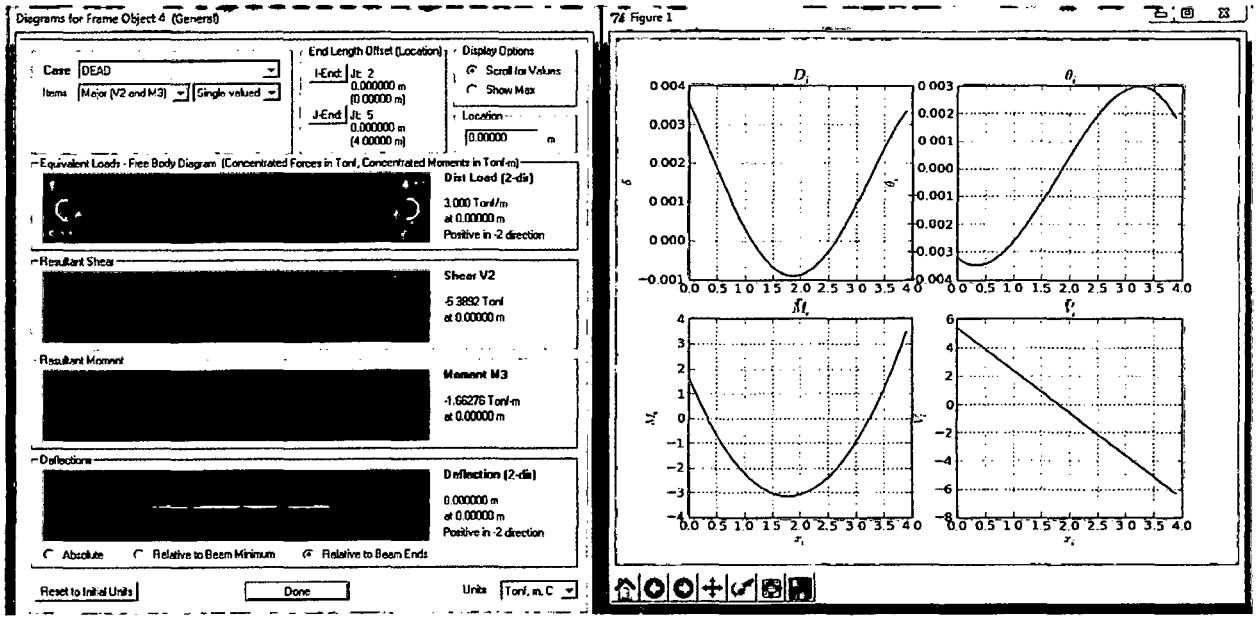


Figura 7.15: Comparación Grafica del Sap 2000 V14 y el Femax.

PROGRAMACIÓN Y MANUAL DEL PROGRAMA(FEMAX)

Capítulo 8

Programación general del Femax.

La aplicación con éxito en este caso del método de elementos finitos se reduce a un programa que he elaborado con uso del lenguaje de python y con un sistema de clases y funciones de listas y librerías,conceptos que fueron desarrollando con teorías de elementos finitos de los capítulos anteriores,el concepto que se elaboró para poder desarrollar el programa fue,en un sistema simple,gráfico y que tenga un fin solo explicativo,pues el programa aun es muy simple para ser usado para cálculos complejos,ya que su fin es simplemente el Femax 1.00 es solo para uso docente y para explicación en si del método ya que el lenguaje fue desarrollado de manera que pueda ser entendida para los que tengan interés en el tema y puedan seguir investigando en el método la cual se deja a criterio de posteriores tesistas que tengan interés de investigación en este método.

8.1. Caracteristicas del Programa Femax.

En la figura(7.1)se muestra el diagrama de flujo del programa ,en el cual se analizará para los ,sistemas o bucles mas importantes de ella la cual nos dará una idea total del sistema.Para centrar conceptos definiremos seguidamente

las etapas fundamentales asociadas al análisis de una estructura por un programa de elementos finitos,asi como la relación de cada etapa con la subrutinas puestas en el diagrama de flujo principal.

8.1.1. Puesta de Datos.

La puesta de datos se ha desarrollado con dos librerías principales de python a cual es numpy y Tkinter una es encargada de los datos numéricos en arreglos la cual es establecida con listas y matrices en la cual ya nos estaremos analizando seguidamente y la otra librería nos da la oportunidad de entorno gráfico,con la cual nos dará la oportunidad de darnos un entorno agradable y trabajable,la subrutina puesta de datos se basa en tres partes una de ellas es la puesta de grilla o ejes de trabajo la cual nos da la oportunidad de tener una cierta forma de ejes de referencia la misma está colocado en la parte superior derecha del programa Femax.

Grilla.

La grilla es un sistema de referencia que se trata de unas lineas de contorno débil y que son aproximadas en los puntos de interés con un sistema de osnap o de acercamiento cuando los eventos del mause se acercan a los (x, y) de los puntos ya derivados del software o puntos de interés del mismo,todos los puntos son referenciados con el osnap y se traduce en facilidad en el trazo de las barras,fuerzas.

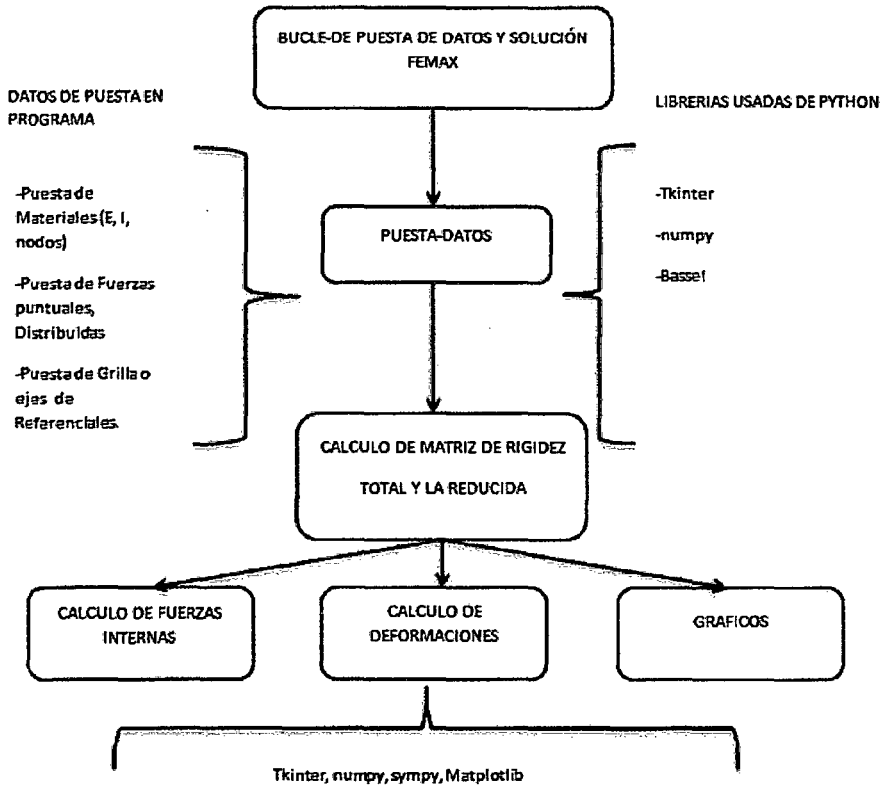


Figura 8.1: Diagrama de flujo del programa Femax.

```
def grilla():
    global Ubicaciones_Universales
    Ubicaciones_Universales=[]
    def dibujo_grilla():
        if nx.get()!=0 and ny.get()!=0:
            for i in range(nx.get()+1):
                dibujo.create_line(50+i*mx.get()*50,550,50
                +i*mx.get()*50,550-
                ny.get()*my.get()*50
                ,fill='gray',activefill='red',dash=(3,5),
                tag=('Recta01'))
            for i in range(ny.get()+1):
                dibujo.create_line(50,550-i*my.get()*50,
```

```
                50+(nx.get())*mx.get()*50,
                550-i*my.get()*50,
                fill='gray',activefill='red',dash=(3,5),
                tag=('Recta02'))
            for u in range(nx.get()+1):
                for m in range(ny.get()+1):
                    Ubicaciones_Universales.append([50+
                    u*mx.get()*50,550-m*my.get()*50])
                    dibujo.create_oval(50+
                    u*mx.get()*50-2,550-m*my.get()*50-2,
                    50+u*mx.get()*50+2,550-
                    m*my.get()*50+2,
                    fill='red',activefill='blue')


        else:
            tkMessageBox.showinfo(message='
            Por Favor poner Datos')

ventana_grilla=Toplevel()
ventana_grilla.title('DIBUJAR GRILLA P/ESQUEMA')
ventana_grilla.minsize(500,200)
ventana_grilla.geometry('500x200+0+0')
#-----------------------------------------------------
nx=IntVar()
mx=DoubleVar()
ny=IntVar()
my=DoubleVar()
#-----------------------------------------------------
texto_principal=Label(ventana_grilla,
text='COORDENADAS EN EL EJE X:Y')
.place(x=100,y=0)
Texto_01=Label(ventana_grilla,
text='N:Ejes-X').place(x=10,y=20)
texto_02=Label(ventana_grilla,
```

```
text='Dist-Ejes-X').place(x=250,y=20)
texto03=Label(ventana_grilla,
text='N:Ejes-Y').place(x=10,y=60)
texto04=Label(ventana_grilla,
text='Dist-Ejes-Y').place(x=250,y=60)
#-------------------------------------------------
Ejes_Horizontales=Entry(ventana_grilla,
textvariable=nx).place(x=80,y=20)
Distancias_Horizontales=Entry(ventana_grilla,
textvariable=mx).place(x=320,y=20)
#-------------------------------------------------
Ejes_Verticales=Entry(ventana_grilla,
textvariable=ny).place(x=80,y=60)
Distancias_Verticales=Entry(ventana_grilla,
textvariable=my).place(x=320,y=60)
Boton01=Button(ventana_grilla,text='APLICAR',
command=dibujo_grilla).place(x=300,y=150)
#-------------------------------------------------
Boton02=Button(ventana_grilla,
text='CANCELAR',
command=ventana_grilla.destroy)
.place(x=400,y=150)
#-------------------------------------------------
# VENTANA DE GRILLAS-LOKITOSAMAX-
#          ELEMENTOS DE VARIABLE
#-------------------------------------------------
ventana_grilla.mainloop()
```

Lineas de Barras.

Subrutina creada para el trazo de barras la cual contiene un algoritmo de eventos consecutivos en dos puntos [inicial, final],las cuales contienen eventos de posiciones y los dibujos creados en un sistema de la librería visual Tkinter,contiene el lienzo o Canvas la cual es un sector de dibujo creado para hacer

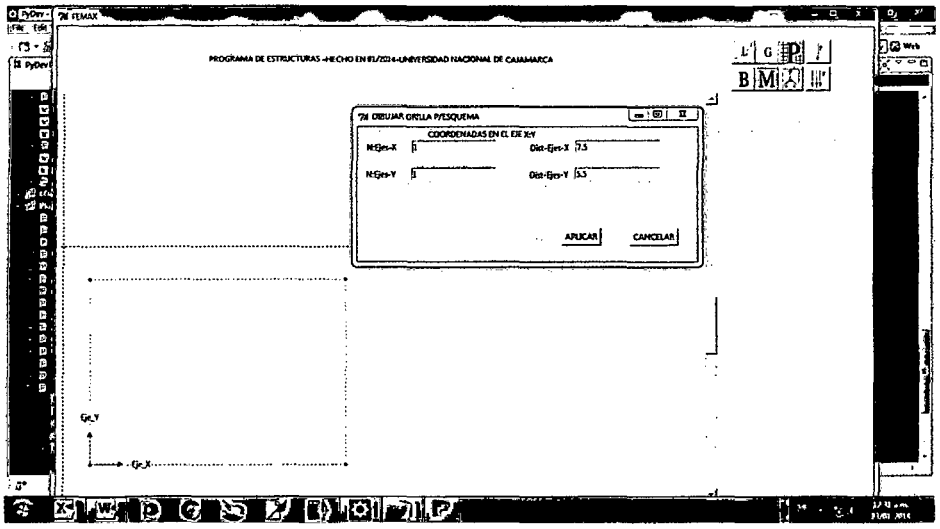


Figura 8.2: Entorno Gráfico del Femax y su función de Grilla y su resultado

representaciones gráficas;nos dan una lógica de armado del problema con las condiciones iniciales así como elementos de frontera o contorno que en el sistema biplanico lineal se basa en los apoyos la cual se verá mas adelante.

```
def dibuja_barra():
    tkMessageBox.showinfo(message=
    'Poner el Primero y Segundo punto.')
    global puntos
    global longitudes
    global u2
    global u
    start =None
    def punto(evento):
        global start
        global u

        start=[evento.x,evento.y]

    def punto2(evento):
        global puntos
        global longitudes
```

```
        global start
        global u
        global u2
        if start is not None:
            x=start[0]
            y=start[1]
            dibujo.create_line(x,y,evento.x,
            evento.y,width=4.5,arrow=LAST,
            activefill='gray',tag=('linea1'))
            dibujo.itemconfig('linea1', fill='red')
            u=u+[(x-50)/50,(550-y)/50,
            (evento.x-50)/50,(550-evento.y)/50]
            longitudes=longitudes+
            [norm(array([(evento.x-50)/50,(550-evento.y)/50])
            -array([(x-50)/50,(550-y)/50]))]
            start=None


dibujo.bind('<Button-1>',punto)
dibujo.bind('<Button-3>',punto2)
u1=array(u)
l1=len(u1)
u2=u1.reshape(l1/2,2)
print u2
print longitudes
```

Las figuras de barras son hechas de color rojo y flechas indicando su posición inicial y final de las mismas,la cual nos ayudaran a hallar las matrices o arreglos de sus características como longitud,Seno,Coseno de las barras respectivamente para porder calcular las matrices de transformación y consecuentemente hallar la matriz de rigidez.

Con el algoritmo descrito nos darán la forma para el calculo del esquema más aun definiremos también los materiales de las barras la cual estará descrito más adelante.

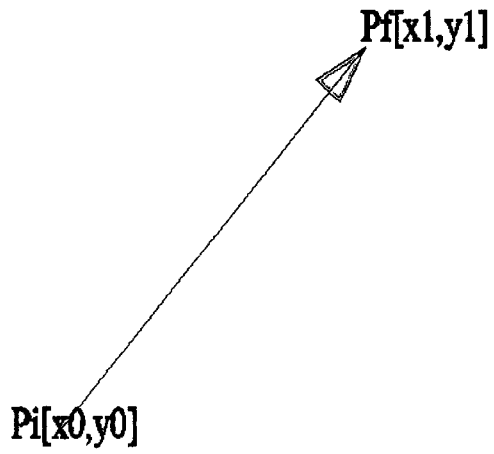


Figura 8.3: Descripción de trazo de Barras en Femax.

Materiales.

Es una subrutina encargada de obtener los materiales tanto de I_i, E_i, A_i,el sistema es consecuencia o depende del numero de las barras con la cual se puede introducir los datos en el arreglo para el respectivo cálculo de la matriz de rigidez,con la cual se hace el arreglo de la matriz general ya calculada y demostrada en el capi-

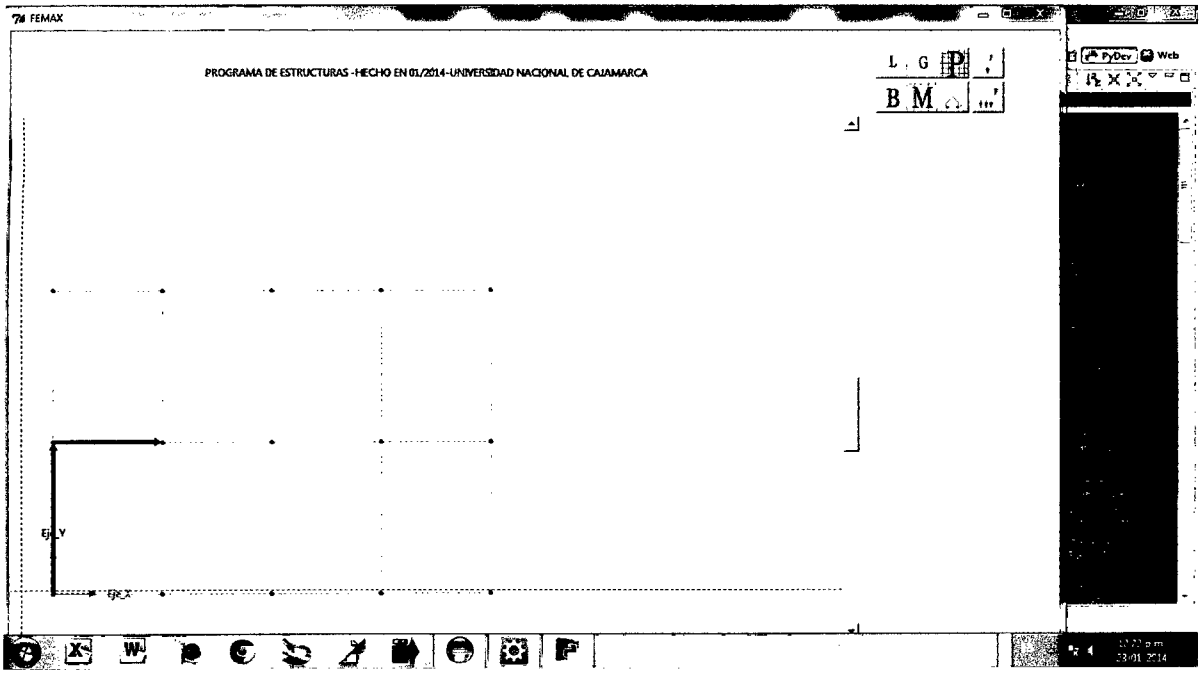


Figura 8.4: Trazo de las barras en el Programa Femax.

tulo 3.2 la cual queda según sus grados de libertad de la siguiente manera:

$$K_T = \begin{bmatrix} \frac{AE}{L} & 0 & 0 & -\frac{AE}{L} & 0 & 0 \\ 0 & 12\frac{EI}{L^3} & 6\frac{EI}{L^2} & 0 & -12\frac{EI}{L^3} & 6\frac{EI}{L^2} \\ 0 & 6\frac{EI}{L^2} & 4\frac{EI}{L} & 0 & -6\frac{EI}{L^2} & 2\frac{EI}{L} \\ -\frac{AE}{L} & 0 & 0 & \frac{AE}{L} & 0 & 0 \\ 0 & -12\frac{EI}{L^3} & -6\frac{EI}{L^2} & 0 & 12\frac{EI}{L^3} & -6\frac{EI}{L^2} \\ 0 & 6\frac{EI}{L^2} & 2\frac{EI}{L} & 0 & -6\frac{EI}{L^2} & 4\frac{EI}{L} \end{bmatrix} \quad (8.1)$$

Será establecido para casos particulares de de barras claro que en este sistema también se podrá evaluar las transformadas y de tal manera poder sumar las matrices

individuales y acoplarlos en la matriz de rigidez total.

$$K_u = T^T K_i T \tag{8.2}$$

esta subrutina genera un bucle de acuerdo a la cantidad de elementos de barra.

```
def Materiales_Calculo():
    global u2
    global numero
    global C3_Materiales
    global nodos_espacio
    nodos_espacio=u2
    ventana_Materiales=Toplevel()
    ventana_Materiales.title('Materiales de Elementos')
    lista_1=['Inercia'+str(i) for i in range(len(u2)/2)]
    lista_2=['Elasticidad'+str(i) for i in range(len(u2)/2)]
    lista_3=['Area'+str(i) for i in range(len(u2)/2)]
    ventana_Materiales.geometry('800x300+0+0')
    numero=len(u2)/2

    for i in range(len(u2)/2):
        lista_1[i]=DoubleVar()
        lista_2[i]=DoubleVar()
        lista_3[i]=DoubleVar()
    for i in range(len(u2)/2):
        u1=Label(ventana_Materiales,
        text='Inercia'+str(i),fg='red').place(x=0,y=20+i*20)
        u2=Label(ventana_Materiales,
        text='Elasticidad'+str(i),fg='red').place(x=200,y=20+i*20)
        u3=Label(ventana_Materiales
        ,text='Area'+str(i),fg='red').place(x=400,y=20+i*20)
    for j in range(numero):
        Entrada_Inercia=Entry(ventana_Materiales,
        textvariable=lista_1[j],width=10).place(x=100,y=20+j*20)
        Entrada_Elasticidad=Entry(ventana_Materiales,
        textvariable=lista_2[j],width=10).place(x=300,y=20+j*20)
        Entrada_Area=Entry(ventana_Materiales,
        textvariable=lista_3[j],width=10).place(x=500,y=20+j*20)
    #-----------------------------------------------------
    C_Materiales=None
```

```
def guardar_datos():
    global C_Materiales
    global C3_Materiales
    C_Materiales=[]
    for t in range(numero):
        C_Materiales=C_Materiales+[lista_1[t].get()]+
        [lista_2[t].get()]+[lista_3[t].get()]
    C2_Materiales=array(C_Materiales)
    C3_Materiales=C2_Materiales.reshape(numero,3)
    print C3_Materiales

#-----------------------------------------------------------------
#BOTON DE GUARDAR CARACTERISTICAS DE LOS ELEMENTOS
Boton01=Button(ventana_Materiales,text='GUARDAR',
command=guardar_datos).place(x=600,y=250)
#-----------------------------------------------------------------
```

Como vemos las caracteristicas del material lo controla un algoritmo que depende del numero de elementos de barra.

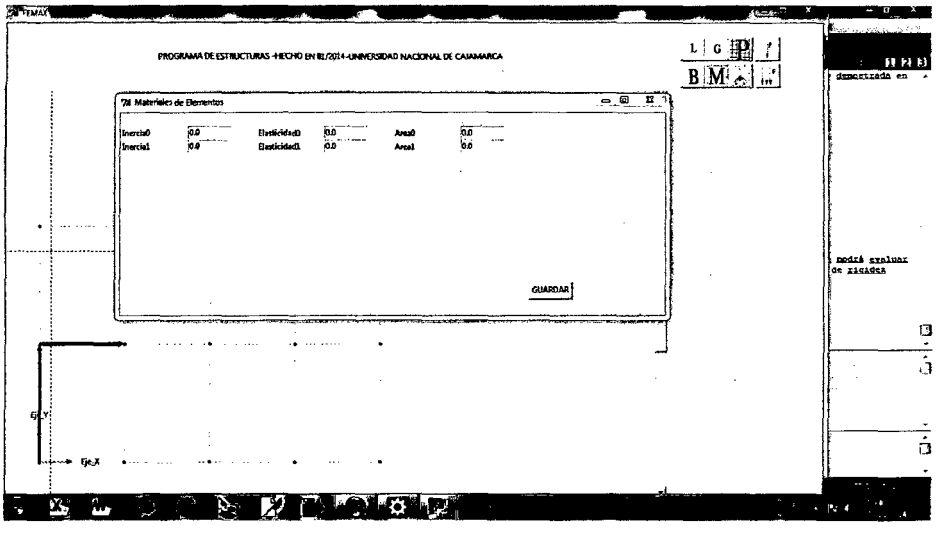


Figura 8.5: Subrutina de Materiales del programa Femax

Apoyos o elementos de Contorno.

Es una subrutina creada para el diagramatización e interpretación matricial de elementos de contorno y tiene como fin obtener los sistemas de condiciones iniciales la cual se da para el respectivo cálculo de la matriz de rigidez disminuida o simplificada ya que este sistema en concepto y en algoritmo nos da la lógica de los grados de libertad restringidos que según sus tipo de apoyos que pueden ser empotrados,móvil o fijos contienen,las restricciones necesarias para interpretarlos como elementos de contorno o condiciones iniciales:

- Apoyos Fijos:

  apoyos que se representa en el programa con un triangulo y contiene un solo grado de libertad en el sentido de giro θ_i sin embargo en el sentido de fuerzas internas contiene dos tanto en el eje x como en el eje y.

- **Apoyos Empotrados.**

  apoyos que se representa en el programa con una barra en T invertida y no con-

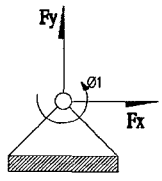


Figura 8.6: Apoyo Fijo y sus propiedades de contorno.

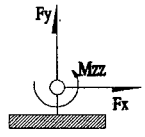


Figura 8.7: Apoyo Empotrado y sus propiedades de contorno.

tiene grados de libertad sin embargo en el sentido de fuerzas internas contiene tres tanto en el eje x , en el eje y el momento M_{zz}.

- **Apoyos Móviles.**

  apoyos que se representa en el programa con un triangulo y un circulo perpendicular y tangente al mismo contiene grados dos libertad en el eje X y la rotación θ_i sin embargo en el sentido de fuerzas internas contiene uno en el eje y.

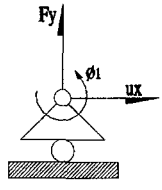


Figura 8.8: Apoyo Móvil y sus propiedades de contorno.

- **Clase Apoyos dibujo.** esta clase fué creada para poder establecer los gráficos en el lienzo de tkinter ya que se tenia que ahorrar lineas de código y para su mejor manejo de los conceptos a la hora de programar,la clase necesaria para tal fin en este sentido vemos la linea de código del paquete nodos.

```
class Apoyos():
    def __init__(self,evento,lienzo):
        self.evento=evento
        self.lienzo=lienzo

    def Aempotrados(self):
        self.lienzo.create_line(self.evento.x,self.evento.y,
        self.evento.x,self.evento.y+10,fill='blue')
        self.lienzo.create_line(self.evento.x,self.evento.y+10,
        self.evento.x-10,self.evento.y+10,fill='blue')
        self.lienzo.create_line(self.evento.x,self.evento.y+10,
        self.evento.x+10,self.evento.y+10,fill='blue')
    def Afijos(self):
        self.lienzo.create_line(self.evento.x,self.evento.y,
        self.evento.x-10,self.evento.y+10,fill='blue')
        self.lienzo.create_line(self.evento.x,self.evento.y,
        self.evento.x+10,self.evento.y+10,fill='blue')
        self.lienzo.create_line(self.evento.x-10,self.evento.y+10,
        self.evento.x+10,self.evento.y+10,fill='blue')
    def Amoviles(self):
        self.lienzo.create_line(self.evento.x,self.evento.y,
        self.evento.x-5,self.evento.y+5,fill='blue')
        self.lienzo.create_line(self.evento.x,self.evento.y,
        self.evento.x+5,self.evento.y+5,fill='blue')
        self.lienzo.create_line(self.evento.x-5,self.evento.y+5,
        self.evento.x+5,self.evento.y+5,fill='blue')
        self.lienzo.create_oval(self.evento.x-2,self.evento.y+5,
        self.evento.x+2,self.evento.y+10,fill='blue')
```

- **programación en el entorno gráfico.**

  en el entorno gráfico nos servimos del paquete apoyos para poder dibujar y representar en arreglos necesarios para poder establecer datos para su respectivo algoritmización en el programa en general.

```
def apoyos_general(evento):
    global tipo_apoyos
    global codigo_apoyos
    global tipo_apoyos1
    global codigo_apoyos1
    if selec.get()==1:
        Apoyos(evento,dibujo).Aempotrados()
        tipo_apoyos=tipo_apoyos+[(evento.x-50)/50,(550-evento.y)/50]
        codigo_apoyos=codigo_apoyos+[1,1,1]
    elif selec.get()==2:
        Apoyos(evento,dibujo).Afijos()
        tipo_apoyos=tipo_apoyos+[(evento.x-50)/50,(550-evento.y)/50]
        codigo_apoyos=codigo_apoyos+[1,1,0]
    elif selec.get()==3:
        Apoyos(evento,dibujo).Amoviles()
        tipo_apoyos=tipo_apoyos+[(evento.x-50)/50,(550-evento.y)/50]
        codigo_apoyos=codigo_apoyos+[0,1,0]

    AP=array(tipo_apoyos)
    AM=array(codigo_apoyos)
    codigo_apoyos1=AM.reshape(len(AM)/3,3)
    tipo_apoyos1=AP.reshape(len(AP)/2,2)
    print tipo_apoyos1
    print codigo_apoyos1
ventana_apoyos=Toplevel()
#-----------------------------------------------------------------
Im_Fijos=PhotoImage(file='fijo.gif')
Im_Empotrados=PhotoImage(file='empotrado.gif')
Im_Moviles=PhotoImage(file='movil.gif')
#-----------------------------------------------------------------
ventana_apoyos.title('Tipo-Apoyos')
selec=IntVar()
Texto_Empotrados=Label(ventana_apoyos,text='Apoyo Empotrado').place(x=
Texto_Fjos=Label(ventana_apoyos,text='Apoyo Fijo').place(x=100,y=60)
```

```
Texto_Moviles=Label(ventana_apoyos,text='Apoyo Moviles').place(x=100,y
Empotrados=Radiobutton(ventana_apoyos,image=Im_Empotrados,value=1,vari
Empotrados.place(x=20,y=20)
Fijos=Radiobutton(ventana_apoyos,image=Im_Fijos,value=2,variable=selec
Fijos.place(x=20,y=60)
Moviles=Radiobutton(ventana_apoyos,image=Im_Moviles,value=3,variable=s
Moviles.place(x=20,y=100)
#-----------------------------------------------------------------
#                    BOTON PARA INCLUIR APOYOS EN LOS PUNTOS
#-----------------------------------------------------------------
botton_01=Button(ventana_apoyos,text='Ejecutar',fg='blue').place(x=200
dibujo.bind('<Button-1>', apoyos_general)
ventana_apoyos.geometry('400x300+0+0')
ventana_apoyos.minsize(400,300)
ventana_apoyos.mainloop()
```

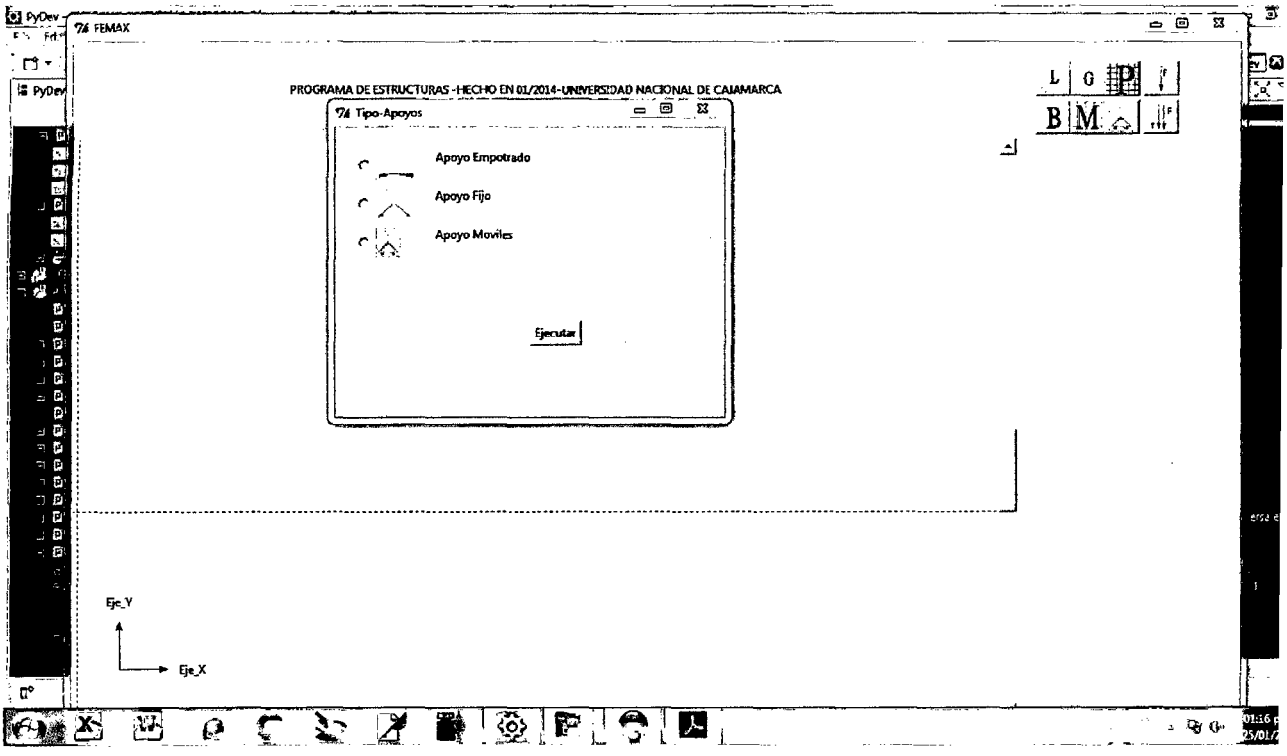


Figura 8.9: Apoyos en el cuadro de diálogo del Femax.

8.1.2. Matriz de Rigidez.

Esta subrutina se encarga del cálculo y acoplamiento de la matriz de rigidez tanto de la general como la matriz reducida,este sistema de acoplamiento fue resuelta con la matriz de ordenamiento o matriz π_i que es un sistema de arreglo basado en los grados de libertad y el numero de barras de análisis la cual vemos en la siguiente formula basado en la figura 7.10.

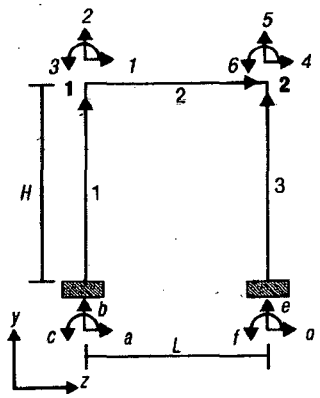


Figura 8.10: Marco de Referencia.

Este método ordena los grados de libertad según el sentido de la barra la cual nos dan la idea de cuales son los elementos de $K_{11}, K_{12}, K_{21}, K_{22}$, la cual está descrito en los puntos de coordenadas globales respecto del sistema de barras.

$$\pi = \begin{bmatrix} 1 & 2 & 3 \\ a & 1 & d \\ b & 2 & e \\ c & 3 & f \\ 1 & 4 & 4 \\ 2 & 5 & 5 \\ 3 & 6 & 6 \end{bmatrix} \rightarrow \text{Grados de Libertad} \tag{8.3}$$

[1] La formula general de ensamblaje tiene mucho que ver con el sentido de la

[1] Esta forma de acoplamiento se encuentra en el libro de Tena Colunga Análisis Estructural con Métodos Matriciales a esta forma de acoplamiento generalización de la matriz de acoplamiento elemento π_i

barra ya que en este sentido en la cual se ordena la matriz de rigidez total;la formula general de ensamblaje seria de la siguiente manera:

$$K_T\{\pi_i[i,n],\pi_i[j,n]\} \leftarrow K_i^n[i,j] \tag{8.4}$$

Donde el símbolo $\longleftarrow$ significa "ensambla dentro" o,explicado de otra manera al valor ya existente.En otras palabras,el coeficiente de rigidez K_{ij}^n se ensambla en el renglón $\pi(i,n)$,columna $\pi(j,n)$ de la matriz de rigidez global$[K]$.como se ilustra en la siguiente figura:

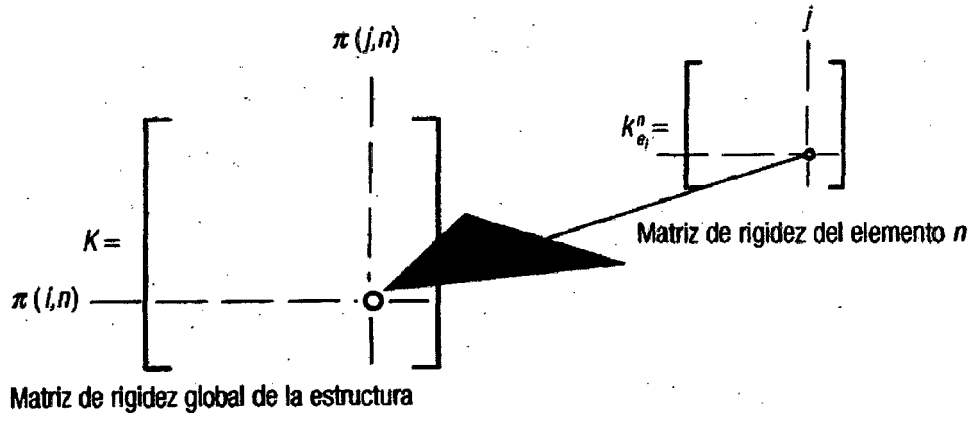


Figura 8.11: Procedimiento del ensamblaje global por medio del método π_i

- Programación del Método π_i. Se muestra en el código fuente del Femax en python para el sistema de acoplamiento usando el método.

```
#-------------------------------------------------------------------
#                          MATRIZ DE ORDENAMIENTO----->PI
#-------------------------------------------------------------------
for i in range(len(C)):
    ordenamiento[i]=C[i]
print ordenamiento
for i in range(numero_datos):
    for j in range(len(C)):
        if ordenamiento[j][0]==nodos_espacio[i][0] and
        ordenamiento[j][1]==nodos_espacio[i][1]:
            ordenamiento_primario.append(j)
print ordenamiento_primario
ordenamiento2=array(ordenamiento_primario)
numero_filas=len(ordenamiento2)/2
ultimo_ordenamiento=ordenamiento2.reshape(numero_filas,2)
P_01S=ultimo_ordenamiento+1
P_02S=P_01S*3
WQ=array([range(P_02S[i][0]-3,P_02S[i][0]) for i in range(len(P_02S))]
WR=array([range(P_02S[i][1]-3,P_02S[i][1]) for i in range(len(P_02S))]
O=[]
for i in range(len(WQ)):
    for j in range(len(WR)):
        if i==j:
            O=O+[array([WQ[i],WR[j]]).reshape(6)]

P_03S=array(O)
matriz_ordenamiento=transpose(P_03S)
print matriz_ordenamiento
```

Veremos Ahora el calculo de la Matriz de Rigidez y su respectivo globalización de la misma usando una clase y un sistema de libreria llamado Matrix2D, la cual será clave para el calculo más rápido claculo de la matriz de rigidez total y

la Matriz simplificada.

```
from numpy import *
from numpy.linalg import *
class Matrix2D():
    def __init__(self,A,E,I,L):
        self.A=A
        self.E=E
        self.I=I
        self.L=L
        self.Kt=zeros((6,6))
    def constructor(self):
        self.Kt[0][0]=self.Kt[3][3]=(self.A*self.E)/float(self.L)
        self.Kt[0][3]=self.Kt[3][0]=-(self.A*self.E)/float(self.L)
        self.Kt[1][1]=self.Kt[4][4]=(12*self.E*self.I)/float((self.L**3))
        self.Kt[4][1]=self.Kt[1][4]=-(12*self.E*self.I)/float((self.L**3))
        self.Kt[1][2]=self.Kt[2][1]=self.Kt[1][5]=self.Kt[5][1]=
        (6*self.E*self.I)/float((self.L**2))
        self.Kt[4][2]=self.Kt[2][4]=self.Kt[4][5]=self.Kt[5][4]=
        -(6*self.E*self.I)/float((self.L**2))
        self.Kt[2][2]=self.Kt[5][5]=4*self.E*self.I/float((self.L))
        self.Kt[2][5]=self.Kt[5][2]=2*self.E*self.I/float((self.L))
        return self.Kt
class Transformada():
    def __init__(self,x1,y1,x2,y2):
        self.x1=x1
        self.x2=x2
        self.y1=y1
        self.y2=y2
        self.L=sqrt((self.x2-self.x1)**2+(self.y2-self.y1)**2)
```

```
        self.h=self.y2-self.y1
        self.d=self.x2-self.x1

    def T(self):
        S=self.h/float(self.L)
        C=self.d/float(self.L)
        t=zeros((6,6))
        t[0][0]=t[1][1]=t[3][3]=t[4][4]=C
        t[0][1]=t[3][4]=S
        t[1][0]=t[4][3]=-S
        t[2][2]=t[5][5]=1

        return t
```

```
#-----------------------------------------------------------------------
#
#               ACOPLAMIENTO FINAL DE LA MATRIZ DE RIGIDEZ TOTAL DE LA EST
#                       k[PI[i][n],PI[j][n]]<------------------K{n}[i][n]
#-----------------------------------------------------------------------
    KT=zeros((3*len(C),3*len(C)))
    for i in range(6):
        for j in range(6):
            for n in range(len(K_Rtodos)):
                KT[matriz_ordenamiento[i][n]][matriz_ordenamiento[j][n]]
                =KT[matriz_ordenamiento[i][n]][matriz_ordenamiento[j][n]]+
    print KT
#-----------------------------------------------------------------------
#
#                                   CALCULO DE MATRIZ REDUCIDA --> Ki
#
#-----------------------------------------------------------------------
    GDL01=[]
    for i in range(len(C)):
        for j in range(len(tipo_apoyos1)):
            if ordenamiento[i][0]==tipo_apoyos1[j][0] and ordenamiento[i][
```

```
            ==tipo_apoyos1[j][1]:
                GDL01.append(i)
    GDL02=array(GDL01)
    GDL03=GDL02+1
    GDL04=GDL03*3
    MQ=array([range(GDL04[i]-3,GDL04[i]) for i in range(len(GDL04))])
    EL=MQ*codigo_apoyos1
    K_reducida_inicial=delete(KT,EL,axis=0)
    K_reducida_final=delete(K_reducida_inicial,EL,axis=1)#calculo de K_red
    print K_reducida_final
```

- Antes del acoplamiento es bueno saber que las matrices tendran que estar sujetas o calculadas en el sistema global la cual se calcula por medio de las siguientes formulas las cuales son establecidas por medio de las siguientes formulas.

$$K_i = \begin{bmatrix} \frac{AE}{L} & 0 & 0 & -\frac{AE}{L} & 0 & 0 \\ 0 & 12\frac{EI}{L^3} & 6\frac{EI}{L^2} & 0 & -12\frac{EI}{L^3} & 6\frac{EI}{L^2} \\ 0 & 6\frac{EI}{L^2} & 4\frac{EI}{L} & 0 & -6\frac{EI}{L^2} & 2\frac{EI}{L} \\ -\frac{AE}{L} & 0 & 0 & \frac{AE}{L} & 0 & 0 \\ 0 & -12\frac{EI}{L^3} & -6\frac{EI}{L^2} & 0 & 12\frac{EI}{L^3} & -6\frac{EI}{L^2} \\ 0 & 6\frac{EI}{L^2} & 2\frac{EI}{L} & 0 & -6\frac{EI}{L^2} & 4\frac{EI}{L} \end{bmatrix} \tag{8.5}$$

La matriz antes mencionada en la formula es establecida como matriz de elementos locales es decir para ejes que sean paralelos a la barra con la cual nosotros sabemos que en los problemas de análisis estructural están geométricamente basados en barras conectadas en forma geométricas complejas pues para ello nos basaremos en la matriz de transformación:

$$T = \begin{bmatrix} cos(\theta) & sen(\theta) & 0 & 0 & 0 & 0 \\ -sen(\theta) & cos(\theta) & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & cos(\theta) & sen(\theta) & 0 \\ 0 & 0 & 0 & -sen(\theta) & cos(\theta) & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{8.6}$$

La formula para llevarlo la matriz local a términos globales de la matriz general o matriz de rigidez total está basado en:

$$K_g = T^T K_i T \tag{8.7}$$

8.1.3. Cálculo de Deformaciones.

- Esta subrutina se encarga del cálculo de las deformaciones globales y deformaciones locales la cual nos dan la oportunidad de aplicar las funciones de deformadas por medio de las funciones de interpolación o funciones de forma que se usan para hallar las gráficas que son halladas por medio de la suma y multiplicación de las deformadas en los grados de libertad. $\theta_1, v_1, u_1, \theta_2, v_2, u_2$

$$u_g = N_1 u_1 + N_2 v_1 + N_3 \theta_1 + N_4 u_2 + N_5 v_2 + N_6 \theta_2 \tag{8.8}$$

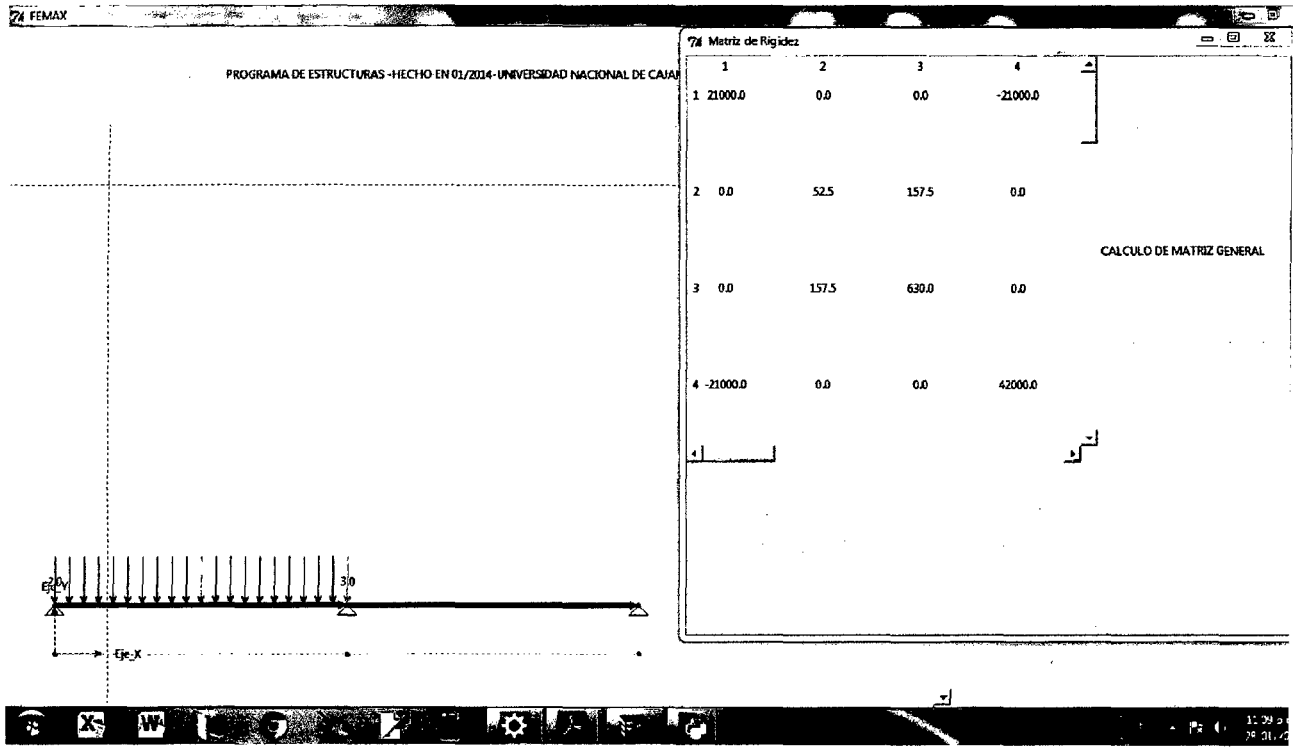


Figura 8.12: Tabla de Resultado de la Matriz de Rigidez Total Femax.

- Así que se habrá forma de calculo respecto de las ecuaciones generales y para poder hallar las ecuaciones se deben conocer las funciones ya estudiadas en anteriores capítulos sobre todo en el calculo de vigas:

$$N_1(x,l) = 1 - \frac{x}{l} \tag{8.9}$$

$$N_2(x,l) = 2(\frac{x}{l})^3 - 3(\frac{x}{l})^2 + 1 \tag{8.10}$$

$$N_3(x,l) = x(1 - \frac{x}{l})^2 \tag{8.11}$$

$$N_4(x,l) = \frac{x}{l} \tag{8.12}$$

$$N_5(x,l) = 3(\frac{x}{l})^2 - 2(\frac{x}{l})^3 \tag{8.13}$$

$$N_6(x,l) = -\frac{x^2}{l}(1 - \frac{x}{l}) \tag{8.14}$$

- El cálculo de las deformaciones son dadas por las siguientes formulas:

$$K_D^{-1} \times F_i = u_i \tag{8.15}$$

```
#.....................................................................
#
#                        Programacion de Calculo de Fuerzas Puntuales
#
#---------------------------------------------------------------------
    esquema_fuerzasp=[]
    for i in range(len(ordenamiento)):
        for j in range(len(Puntos_AFuerzasP1)):
            if ordenamiento[i][0]==
            Puntos_AFuerzasP1[j][0] and ordenamiento[i][1]==Puntos_AFuerza
                esquema_fuerzasp.append(i)
    print esquema_fuerzasp
    esquema_fuerzasp1=array(esquema_fuerzasp)
    esquema_fuerzasp2=(esquema_fuerzasp1+1)*3
    WR=array([range(esquema_fuerzasp2[i]-
    3,esquema_fuerzasp2[i]) for i in range(len(esquema_fuerzasp2))])
    WQ=WR.reshape(len(WR)*3,1)
    Fuerzas_puntuales=zeros((len(ordenamiento)*3,1))
    Puntos_FuerzasP2=Puntos_FuerzasP1.reshape(len(Puntos_FuerzasP1)*3,1)
    for i in range(len(Puntos_FuerzasP1)*3):
        Fuerzas_puntuales[WQ[i]]=Fuerzas_puntuales[WQ[i]]+Puntos_FuerzasP2
    print Fuerzas_puntuales
#---------------------------------------------------------------------
#
#                        Programacion de calculo de Fuerzas Distribuidas
#
#---------------------------------------------------------------------
    F5=F4.reshape(len(F4)*2,2)
    esquema_FuerzasD=[]
    for i in range(len(ordenamiento)):
        for j in range(len(F5)):
            if ordenamiento[i][0]==F5[j][0] and
            ordenamiento[i][1]==F5[j][1]:
                esquema_FuerzasD.append(i)
```

```
    esquema_FuerzasD1=array(esquema_FuerzasD)
    esquema_FuerzasD2=(esquema_FuerzasD1+1)*3
    WU=array([range(esquema_FuerzasD2[i]-3,
    esquema_FuerzasD2[i]) for i in range(len(esquema_FuerzasD2))])
    WP=WU.reshape(len(WU)*3,1)
    print 'este es la clave'+str(WP)

    Fuerzas_Distribuidas=zeros((len(ordenamiento)*3,1))
    Fuerzas_DD=[]
    for i in range(len(longitudesFD)):
        Fuerzas_DD=Fuerzas_DD+
        [[0,Valores_FD[i]*(longitudesFD[i])/float(2),
        Valores_FD[i]*(longitudesFD[i]**2)/float(12)],
        [0,Valores_FD[i]*(longitudesFD[i])/float(2),
        -Valores_FD[i]*(longitudesFD[i]**2)/float(12)]]
    FuerzasDD1=array(Fuerzas_DD)
    FuerzasDD2=FuerzasDD1.reshape(len(FuerzasDD1)*3,1)
    for i in range(len(FuerzasDD2)):
        Fuerzas_Distribuidas[WP[i]]=Fuerzas_Distribuidas[WP[i]]+FuerzasDD2
#-----------------------------------------------------------------------
#
#                        Programacion de Fuerzas Nodales Equivalentes
#
#-----------------------------------------------------------------------
    Fuerzas_nequivalentes=Fuerzas_puntuales-Fuerzas_Distribuidas
    Fuerzas_NodD=delete(Fuerzas_nequivalentes,EL,axis=0)
    print Fuerzas_NodD
#-----------------------------------------------------------------------
#
#                        Calculo de las deformaciones Nodales
#
#-----------------------------------------------------------------------
    Deformaciones01=dot(inv(K_reducida_final),Fuerzas_NodD)
    print Deformaciones01
    WUP=range(0,len(Fuerzas_nequivalentes))
```

```
WUP1=array(WUP)
WUP2=WUP1.reshape(len(WUP),1)
WUP3=delete(WUP2,EL,axis=0)
Deformaciones_totales=zeros((len(ordenamiento)*3,1))
for i in range(len(WUP3)):
    Deformaciones_totales[WUP3[i]]=
    Deformaciones_totales[WUP3[i]]+Deformaciones01[i]
print Deformaciones_totales
```

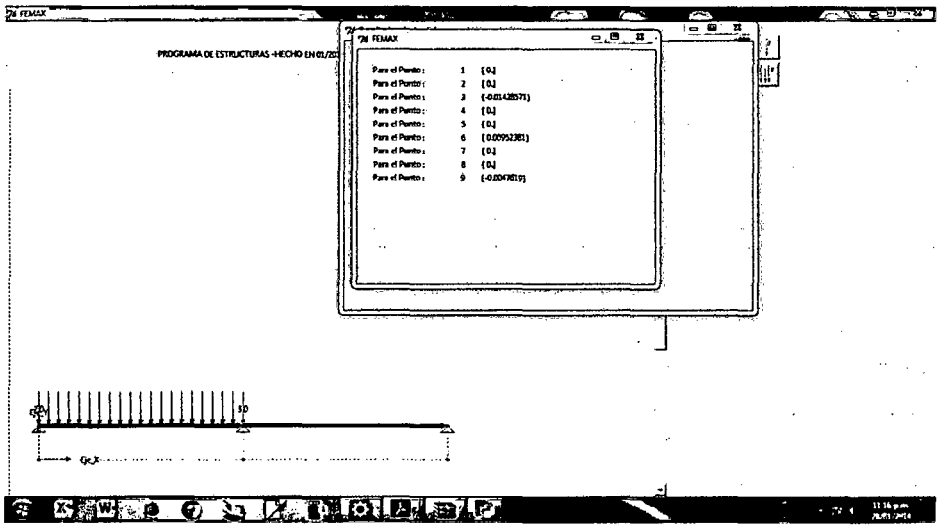


Figura 8.13: Generación de tabla de Resultados de la tabla de las deformadas Femax.

8.1.4. Cálculo de Fuerzas Internas.

Esta subrutina es para el cálculo de las fuerzas internas que en este caso es por barra la cual nos dan la oportunidad de hallar el cálculo de las fuerzas internas por barra:

$$u_i = Tu_i^{'} \tag{8.16}$$

$$K^i(Tu_i^{'}) = F_{\text{int x Barra}} \tag{8.17}$$

```
#-----------------------------------------------
#
#         Calculo de Fuerzas Internasx-Barras
#
#-----------------------------------------------
    us01=transpose(matriz_ordenamiento)
    ordenamiento_fuerzasU=[]
    for i in range(len(us01)):
        sx=us01[i].reshape(1,6)
        ordenamiento_fuerzasU=ordenamiento_fuerzasU+[sx]
    ordenamiento_ff=array(ordenamiento_fuerzasU).
    reshape(len(ordenamiento_fuerzasU),6)
    SMR=array([[0.000000,0.000000,0.000000,0.000000,
    0.000000,0.000000]
    for i in range(len(ordenamiento_ff))])
    for i in range(len(ordenamiento_ff)):
        for j in range(6):
            SMR[i][j]=SMR[i][j]+
            Deformaciones_totales[ordenamiento_ff[i][j]]

    print SMR
    SMRWW=[]
    for i in range(len(ordenamiento_ff)):
        SMRWW=SMRWW+[dot(T[i],SMR[i].reshape(6,1))]
    SMRWW1=array(SMRWW)
    SMRWW2=SMRWW1.reshape(len(SMRWW),6)
    print SMRWW2
    resultados_fuerzapb=[]
    for i in range(len(SMR)):
        resultados_fuerzapb=resultados_fuerzapb+
        [dot(K_Rtodos[i],SMR[i].reshape(6,1))]
    print resultados_fuerzapb
    restar_altotal=Fuerzas_nequivalentes-Fuerzas_puntuales
    SMR1=array([[0.000000,0.000000,0.000000,0.000000,
```

```
0.000000,0.000000]
for i in range(len(ordenamiento_ff))])
WNN=WP.reshape(len(WP)/6.00,6.00)
for i in range(len(ordenamiento_ff)):
    for j in range(len(WNN)):
        if ordenamiento_ff[i][0]==WNN[j][0] and
        ordenamiento_ff[i][1]==WNN[j][1] and
        ordenamiento_ff[i][2]==WNN[j][2] and
        ordenamiento_ff[i][3]==WNN[j][3] and
        ordenamiento_ff[i][4]==WNN[j][4] and
        ordenamiento_ff[i][5]==WNN[j][5]:
            SMR1[i][0]=ordenamiento_ff[i][0]
            SMR1[i][1]=ordenamiento_ff[i][1]
            SMR1[i][2]=ordenamiento_ff[i][2]
            SMR1[i][3]=ordenamiento_ff[i][3]
            SMR1[i][4]=ordenamiento_ff[i][4]
            SMR1[i][5]=ordenamiento_ff[i][5]
        else:
            SMR1[i][0]=0.000000000
            SMR1[i][1]=0.000000000
            SMR1[i][2]=0.000000000
            SMR1[i][3]=0.000000000
            SMR1[i][4]=0.000000000
            SMR1[i][5]=0.000000000
SMR2=array([[0.000000,0.000000,0.000000,0.000000,
0.000000,0.000000]
for i in range(len(ordenamiento_ff))])

for i in range(len(ordenamiento_ff)):
    for j in range(6):
        SMR2[i][j]=SMR2[i][j]+restar_altotal[SMR1[i][j]]



fuerzas_bfinal=[]
for i in range(len(ordenamiento_ff)):
```

```
        fuerzas_bfinal=fuerzas_bfinal+[resultados_fuerzapb[i]
        -SMR2[i].reshape(6,1)]
print fuerzas_bfinal
```

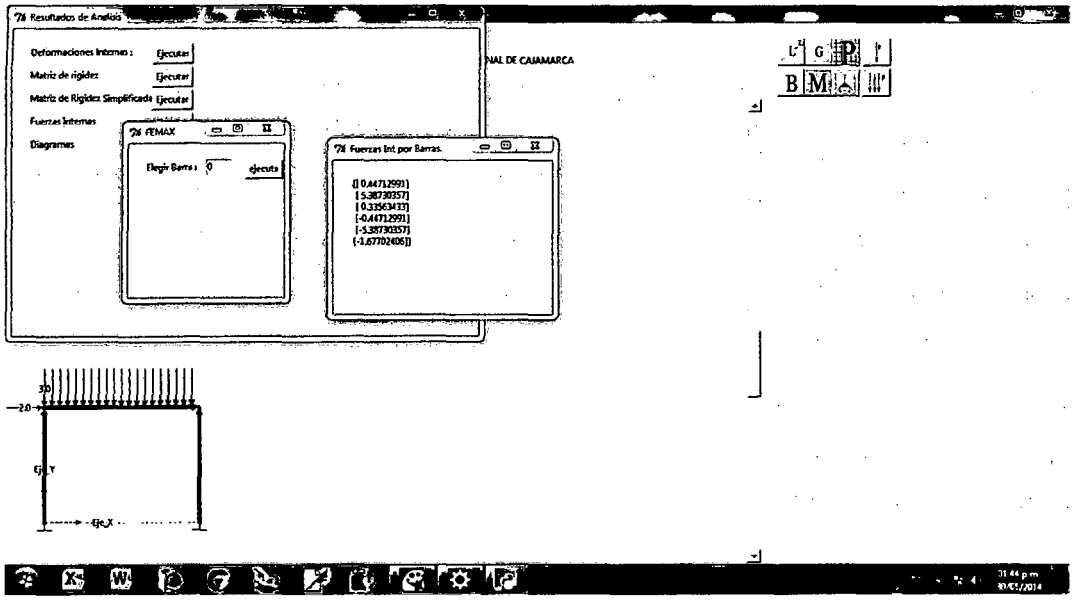


Figura 8.14: Tabla de Resultados para el cálculo de Fuerzas Internas del programa Femax.

8.1.5. Gráficos y Tablas de Resultados.

Esta subrutina es encargada del cálculo de los gráficos de los esquemas usados en los acapites anteriores de las subrutinas anteriores ,esta parte del programa esta programada con la librería de Matplotlib,una librería usada para gráficos de gran capacidad la cual nos ayudaran para graficar las funciones de Cortantes,Momentos,Deformadas y giros, la cual queda establecida por medio de las funciones de forma.

$$u_i = N_1(u_1) + N_2(v_1) + N_3(\theta_1) + N_4(u_2) + N_5(v_2) + N_6(\theta_3) + R_i(x,l) \quad (8.18)$$

$$\theta_i = \frac{\partial N_1}{\partial x}(u_1) + \frac{\partial N_2}{\partial x}(v_1) + \frac{\partial N_3}{\partial x}(\theta_1) + \frac{\partial N_4}{\partial x}(u_2) + \frac{\partial N_5}{\partial x}(v_2) + \frac{\partial N_6}{\partial x}(\theta_2) + R_i(x,l)' \quad (8.19)$$

$$M_i = EI\frac{\partial^2 u}{\partial x^2} = \frac{\partial^2 N_1}{\partial x^2}(u_1) + \frac{\partial^2 N_2}{\partial x^2}(v_1) + \frac{\partial^2 N_3}{\partial x^2}(\theta_1) + \frac{\partial^2 N_4}{\partial x^2}(u_2) + \frac{\partial^2 N_5}{\partial x^2}(v_2) + \frac{\partial^2 N_6}{\partial x^2}(\theta_2) + R_i(x,l)'' \quad (8.20)$$

$$V_i = EI\frac{\partial^3 u}{\partial x^3} = \frac{\partial^3 N_1}{\partial x^3}(u_1) + \frac{\partial^3 N_2}{\partial x^3}(v_1) + \frac{\partial^3 N_3}{\partial x^3}(\theta_1) + \frac{\partial^3 N_4}{\partial x^3}(u_2) + \frac{\partial^3 N_5}{\partial x^3}(v_2) + \frac{\partial^3 N_6}{\partial x^3}(\theta_2) + R_i(x,l)''' \quad (8.21)$$

- El algoritmo para la graficación de la misma se propone en el código en el siguiente item.

```
#-------------------------------------------------------
#
#          Algoritmo de Graficas de u,a_{i},M_{i},V_{i}
#
#-------------------------------------------------------
    X=[]
    for i in range(len(ordenamiento_ff)):
        xi=arange(0,longitudes[i],0.1)
        X=X+[xi]
    print X
    VVUW=nodos_espacio.reshape(len(nodos_espacio)/2,4)
    print 'esta mierda es :',VVUW
    PO=[]
    FMMUT=zeros((len(VVUW),1))
    for i in range(len(VVUW)):
        for j in range(len(F4)):
            if VVUW[i][0]==F4[j][0] and
            VVUW[i][1]==F4[j][1]
            and VVUW[i][2]==F4[j][2] and
```

```
            VVUW[i][3]==F4[j][3]:
                FMMUT[i][0]=-Valores_FD[j]
print FMMUT
F_DD=FMMUT

print F_DD
Form=[]
for i in range(len(ordenamiento_ff)):
    Form=Form+[Forma_GG(X[i],longitudes[i],
    C3_Materiales[i][1],C3_Materiales[i][0],F_DD[i])]
DEFF=[]
for i in range(len(ordenamiento_ff)):
    DEFF=DEFF+[Form[i].N_1()*SMRWW2[i][0]+Form[i].N_2()
    *SMRWW2[i][1]+
    Form[i].N_3()*SMRWW2[i][2]+Form[i].N_4()
    *SMRWW2[i][3]+
    Form[i].N_5()*SMRWW2[i][4]+Form[i].N_6()
    *SMRWW2[i][5]+
    Form[i].R_1()]
print DEFF
DEFG=[]
for i in range(len(ordenamiento_ff)):
    DEFG=DEFG+[(Form[i].W_1()*SMRWW2[i][0]
    +Form[i].W_2()*SMRWW2[i][1]+Form[i].W_3()
    *SMRWW2[i][2]
    +Form[i].W_4()*SMRWW2[i][3]+Form[i].W_5()
    *SMRWW2[i][4]
    +Form[i].W_6()*SMRWW2[i][5]+Form[i].R_2())]
print DEFG
Momenx=[]
for i in range(len(ordenamiento_ff)):
    Momenx=Momenx+[-1*(Form[i].M_1()*SMRWW2[i][0]
    +Form[i].M_2()*SMRWW2[i][1]+Form[i].M_3()
    *SMRWW2[i][2]
    +Form[i].M_4()*SMRWW2[i][3]+Form[i].M_5()
```

```
        *SMRWW2[i][4]
        +Form[i].M_6()*SMRWW2[i][5]+Form[i].R_3())]
    print Momenx
    Cort=[]
    for i in range(len(ordenamiento_ff)):
        Cort=Cort+[Form[i].V_1()*SMRWW2[i][0]+
        Form[i].V_2()*SMRWW2[i][1]+Form[i].V_3()
        *SMRWW2[i][2]+
        Form[i].V_4()*SMRWW2[i][3]+Form[i].V_5()
        *SMRWW2[i][4]+
        Form[i].V_6()*SMRWW2[i][5]+Form[i].R_4()]
    print Cort
    print longitudes

#-------------------------------------------------
#
#    Algoritmo de Presentacion de Resultados.
#
#-------------------------------------------------
    def Diagramasfinales():
        def Diagrama():
            fiig=figure(1)
            TDef=fiig.add_subplot(221)
            title(r'$D_{i}$')
            grid(True)
            xlabel(r'$x_{i}$')
            ylabel(r'$\delta$')
            TDef.plot(X[m.get()],DEFF[m.get()])
            TGir=fiig.add_subplot(222)
            title(r'$\theta_{i}$')
            xlabel(r'$x_{i}$')
            ylabel(r'$\theta_{i}$')
            grid(True)
            TGir.plot(X[m.get()],DEFG[m.get()])
            TMom=fiig.add_subplot(223)
```

```
            title(r'$M_{i}$')
            xlabel(r'$x_{i}$')
            ylabel(r'$M_{i}$')
            grid(True)
            TMom.plot(X[m.get()],Momenx[m.get()])
            TCort=fiig.add_subplot(224)
            title(r'$V_{i}$')
            xlabel(r'$x_{i}$')
            ylabel(r'$V_{i}$')
            grid(True)
            TCort.plot(X[m.get()],Cort[m.get()])
            show()
        ven01=Toplevel()
        ven01.title('Diagramas de las Barras')
        textoMM=Label(ven01,text='Elija la Barra :',
        fg='blue').place(x=20,y=20)
        m=IntVar()
        EntradaMM=Entry(ven01,
        textvariable=m,width=10).place(x=100,y=20)
        BotonMM=Button(ven01,text='ejecuta',
        command=Diagrama).place(x=200,y=20)
        ven01.mainloop()
    def Fuer_Int():
        def VFuerzas01():
            ventana_alterna01=Toplevel()
            texto=Label(ventana_alterna01,text=fuerzas_bfinal
            [n.get()]).place(x=20,y=20)
            ventana_alterna01.title('Fuerzas Int por Barras.')
            ventana_alterna01.mainloop()
        ventana_fuerzasint=Toplevel()
        texto_01=Label(ventana_fuerzasint,
        text='Elegir Barra :').place(x=20,y=20)
        n=IntVar()
        entrada01=Entry(ventana_fuerzasint,
        textvariable=n,width=5).place(x=100,y=20)
```

```
        Boton01=Button(ventana_fuerzasint,text='ejecuta',
        command=VFuerzas01).place(x=150,y=20)
        ventana_fuerzasint.mainloop()
    def Deformaciones_FFU():
        ventana_Deformaciones=Toplevel()
        for i in range(len(Deformaciones_totales)):
            Label(ventana_Deformaciones,
            text='Para el Punto :').place(x=20,y=i*20+20)
            Label(ventana_Deformaciones,
            text=i+1).place(x=150,y=i*20+20)
            Label(ventana_Deformaciones,
            text=Deformaciones_totales[i]).place(x=180,y=i*20+20)


        ventana_Deformaciones.mainloop()
    def Matriz_RRIG():
        ventana_matrix=Toplevel()
        ventana_matrix.geometry('600x600+0+0')
        ventana_matrix.title('Matriz de Rigidez')
        textos=Canvas(ventana_matrix,width=400,height=400,
        scrollregion=(0,0,2000,2000),
        highlightcolor='black',relief='solid',background='white')
        for i in range(len(KT)):
            for j in range(len(KT)):
                textos.create_text(100*i+40,100*j+40,text=KT[i][j])
        for i in range(len(KT)):
            textos.create_text(100*i+40,10,text=i+1,fill='red')
            textos.create_text(10,100*i+40,text=i+1,fill='red')
        textos01=Label(ventana_matrix,
        text='CALCULO DE MATRIZ GENERAL',fg='red')
        textos01.grid(row=1,column=3)
        scrollY=Scrollbar(ventana_matrix,
        orient=VERTICAL,command=textos.yview)
        scrollY.grid(row=1,column=2,sticky=N+S)
        scrollX=Scrollbar(ventana_matrix,
```

```
        orient=HORIZONTAL,command=textos.xview)
        scrollX.grid(row=2,column=1,sticky=E+W)
        textos['xscrollcommand']=scrollX.set
        textos['yscrollcommand']=scrollY.set
        textos.grid(row=1,column=1)

        ventana_matrix.mainloop()
    def Matriz_DMM():
        ventana_matrixD=Toplevel()
        ventana_matrixD.geometry('600x600+0+0')
        ventana_matrixD.title('Matriz de Rigidez Simplificada')
        textos1=Canvas(ventana_matrixD,width=400,height=400,
        scrollregion=(0,0,2000,2000),
        highlightcolor='black',relief='solid',background='white')
        for i in range(len(K_reducida_final)):
            for j in range(len(K_reducida_final)):
                textos1.create_text(100*i+40,100*j+40,
                text=K_reducida_final[i][j])
        for i in range(len(K_reducida_final)):
            textos1.create_text(100*i+40,10,
            text=i+1,fill='red')
            textos1.create_text(10,100*i+40,
            text=i+1,fill='red')
        textos02=Label(ventana_matrixD,
        text='CALCULO DE MATRIZ SIMPLIFICADA',fg='red')
        textos02.grid(row=1,column=3)
        scrollY=Scrollbar(ventana_matrixD,orient=VERTICAL,
        command=textos1.yview)
        scrollY.grid(row=1,column=2,sticky=N+S)
        scrollX=Scrollbar(ventana_matrixD,orient=HORIZONTAL,
        command=textos1.xview)
        scrollX.grid(row=2,column=1,sticky=E+W)
        textos1['xscrollcommand']=scrollX.set
        textos1['yscrollcommand']=scrollY.set
        textos1.grid(row=1,column=1)
```

```
    ventana_matrixD.mainloop()
ventana_Resultados=Toplevel()
ventana_Resultados.geometry('600x400+0+0')
ventana_Resultados.title('Resultados de Analisis')
texto_01=Label(ventana_Resultados,
text='Deformaciones Internas :',fg='blue').place(x=20,y=20)
Boton01=Button(ventana_Resultados,text='Ejecutar',fg='red',
command=Deformaciones_FFU).place(x=180,y=20)
texto_02=Label(ventana_Resultados,
text='Matriz de rigidez',fg='blue').place(x=20,y=50)
Boton02=Button(ventana_Resultados,
text='Ejecutar',fg='red',
command=Matriz_RRIG).place(x=180,y=50)
texto03=Label(ventana_Resultados,
text='Matriz de Rigidez Simplificada',fg='blue').place(x=20,y=80)
Boton03=Button(ventana_Resultados,text='Ejecutar',
command=Matriz_DMM,fg='red').place(x=180,y=80)
texto04=Label(ventana_Resultados,
text='Fuerzas Internas',fg='blue').place(x=20,y=110)
Boton04=Button(ventana_Resultados,text='Ejecutar',
command=Fuer_Int,fg='red').place(x=180,y=110)
texto05=Label(ventana_Resultados,
text='Diagramas',fg='blue').place(x=20,y=140)
Boton05=Button(ventana_Resultados,text='ejecuta',
command=Diagramasfinales,fg='red').place(x=180,y=140)
ventana_Resultados.mainloop()
```

- En las gráficas se dividen en 4,una es la gráfica de la deformada,la otra de gráfica de giro o desplazamiento de giro,la otra el Momento, y la ultima la cortante.

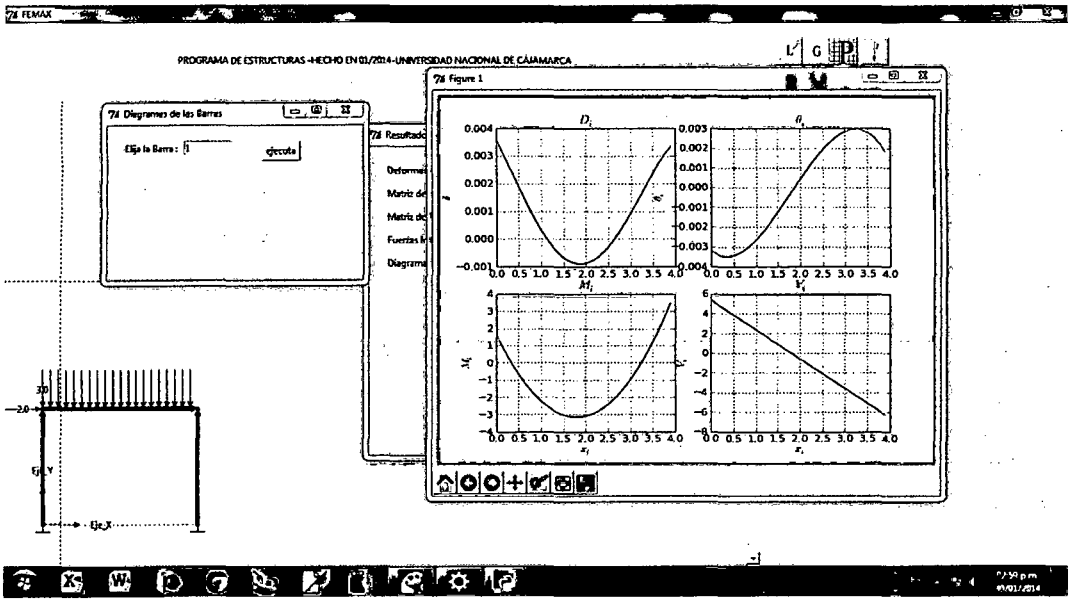


Figura 8.15: Gráfica de Resultados del Análisis del Femax de u_i,θ_i,M_i,V_i.

Con esta subrutina se termina el programa femax.

CONCLUSIONES Y RECOMENDACIONES.

Capítulo 9

Conclusiones y Recomendaciones.

9.1. Conclusiones.

- Se pudo ver que al ser los resultados obtenidos por el programa FEMAX,en contra partida de los resultados obtenidos por el Software Comercial SAP V14 hay un parecido de 99.8 % por lo tanto se puede concluir que los programas comerciales trabajan con el método de elementos finitos y nuestra tesis hipótesis ha sido resuelta como cierta.

- El sistema de acoplamiento generado para la matriz de rigidez,ha sido satisfactoria por lo cual se puede tomar como referencia el algoritmo para elementos de todo tipo así como placas,sólidos de revolución etc.

$$K_T\{(\pi(i,n),\pi(j,n))\} \longleftarrow K^n(i,j) \tag{9.1}$$

9.2. Recomendaciones.

- Se recomienda que se dicte el curso de elementos Finitos o que en sílabus de Análisis Estructural se la incluya ya que es un elemento importante para

entender como funcionan los programas comerciales de Estructuras y le dará más facilidad de entendimiento al alumno.

- Se da pie a que las investigaciones con elementos finitos sigan adelante ya que es una gran herramienta para poder calcular problemas complejos en la ingeniería.
- Para el cálculo de pórticos y vigas se uso los sistemas de acoplamiento de matriz π_i,este método puede ser usado para todo tipo de estructura ya que es un algoritmo universal ya que se establece la opción de dirección y grado de libertad.
- El uso de las funciones de forma son fundamentales en el análisis estructurales del método de elementos finitos y se recomienda el estudio minucioso de ello ya que es también un sistema universal para el cálculo de las gráficas y cálculo de u_i, θ_i, M_i, V_i en cualquier punto estructural.
- Se recomienda el uso masivo del lenguaje de programación como python para alumnos Universitarios por que nos dan gran alternativos de desarrollos para el cálculo de Ingeniería .

ANEXOS.

Capítulo 10

Codigo Fuente del Programa Femax.

```
#PROGRAMACION FEMAX
from matplotlib import *
from pylab import *
import matplotlib.pylab as plt
from Tkinter import*
from Matrix2D import *
import tkMessageBox
import time
from apoyos import *
from Fuerza2D import *
from Formax import *
#-----------------------------------------------
C=[]
C3_Materiales=[]
longitudes=[]
K_todos=[]
u=[]
u2=[]
nodos_espacio=None
T=[]
```

```
K_Rtodos=[]
ordenamiento=None
puntos=[]
tipo_apoyos=None
codigo_apoyos=None
tipo_apoyos1=None
codigo_apoyos1=None
Ubicaciones_Universales=None
Puntos_FuerzasP=None
Puntos_AFuerzasP=None
Puntos_FuerzasP1=None
Puntos_AFuerzasP1=None
inicio_FuerzasD=None
linea_FuerzasD=[]
F4=[]
Valores_FD=[]
longitudesFD=[]
#-------------------------------------------
def Fuerzas_puntuales():
    global Puntos_AFuerzasP
    global Puntos_FuerzasP
    global Puntos_FuerzasP1
    global Puntos_AFuerzasP1
    Puntos_FuerzasP=[]
    Puntos_AFuerzasP=[]
    def dibujar_Fuerzas(evento):
        global Puntos_FuerzasP1
        global Puntos_AFuerzasP1
        global Puntos_FuerzasP
        global Puntos_AFuerzasP
        if selec01.get()==1:
            val=selecP01.get()
            fuerzas(evento,dibujo,val).Fuerza_VP()
            Puntos_FuerzasP=Puntos_FuerzasP+[0,val,0]
            Puntos_AFuerzasP=Puntos_AFuerzasP+[(evento.x-50)/50,
```

```
            (550-evento.y)/50]
        if selec01.get()==2:
            val=selecP02.get()
            fuerzas(evento,dibujo,val).Fuerza_VN()
            Puntos_FuerzasP=Puntos_FuerzasP+[0,-val,0]
            Puntos_AFuerzasP=Puntos_AFuerzasP+[(evento.x-50)/50,
            (550-evento.y)/50]
        if selec01.get()==3:
            val=selecP03.get()
            fuerzas(evento,dibujo,val).Fuerza_HP()
            Puntos_FuerzasP=Puntos_FuerzasP+[val,0,0]
            Puntos_AFuerzasP=Puntos_AFuerzasP+[(evento.x-50)/50,
            (550-evento.y)/50]
        if selec01.get()==4:
            val=selecP04.get()
            fuerzas(evento,dibujo,val).Fuerza_HN()
            Puntos_FuerzasP=Puntos_FuerzasP+[-val,0,0]
            Puntos_AFuerzasP=Puntos_AFuerzasP+[(evento.x-50)/50,
            (550-evento.y)/50]
        u=array(Puntos_FuerzasP)
        s=len(u)
        u2=array(Puntos_AFuerzasP)
        s1=len(u2)
        Puntos_FuerzasP1=u.reshape(s/3,3)
        Puntos_AFuerzasP1=u2.reshape(s1/2,2)
        print Puntos_FuerzasP1
        print Puntos_AFuerzasP1

    ventana_FuerzasP=Toplevel()
    ventana_FuerzasP.title('Fuerzas Puntuales')
    ventana_FuerzasP.geometry('500x300+0+0')
    ventana_FuerzasP.minsize(500, 300)

    #-------------------------------------------------
    IM_FVP=PhotoImage(file='FPV.gif')
```

```
IM_FVN=PhotoImage(file='FNV.gif')
IM_FHP=PhotoImage(file='FHP.gif')
IM_FHN=PhotoImage(file='FHN.gif')
#-----------------------------------------------------
selec01=IntVar()
selecP01=DoubleVar()
selecP02=DoubleVar()
selecP03=DoubleVar()
selecP04=DoubleVar()
#-----------------------------------------------------
entrada01=Entry(ventana_FuerzasP,textvariable=selecP01,
width=10).place(x=300,y=20)
entrada02=Entry(ventana_FuerzasP,textvariable=selecP02,
width=10).place(x=300,y=60)
entrada03=Entry(ventana_FuerzasP,textvariable=selecP03,
width=10).place(x=300,y=100)
entrada04=Entry(ventana_FuerzasP,textvariable=selecP04,
width=10).place(x=300,y=140)
#-------------------------------------------------------------------
Texto_01=Label(ventana_FuerzasP,
text='Fuerzas Verticales Positivos.').place(x=100,y=20)
FVP=Radiobutton(ventana_FuerzasP,
image=IM_FVP,value=1,variable=selec01)
texto02=Label(ventana_FuerzasP,
text='Fuerzas Verticales Negativos').place(x=100,y=60)
FVP.place(x=20,y=20)

FVN=Radiobutton(ventana_FuerzasP,
image=IM_FVN,value=2,variable=selec01)
FVN.place(x=20,y=60)
texto03=Label(ventana_FuerzasP,
text='Fuerzas Horizontales Positivas').place(x=100,y=100)
FHP=Radiobutton(ventana_FuerzasP,
image=IM_FHP,value=3,variable=selec01)
FHP.place(x=20,y=100)
```

```
    texto04=Label(ventana_FuerzasP,
    text='Fuerzas Horizontales Negativas').place(x=100,y=140)
    FHN=Radiobutton(ventana_FuerzasP,
    image=IM_FHN,value=4,variable=selec01)
    FHN.place(x=20,y=140)
    dibujo.bind('<Button-1>',dibujar_Fuerzas)
    ventana_FuerzasP.mainloop()

def Fuerzas_distribuidas():
    global longitudesFD
    global Valores_FD
    global F4
    global inicio_FuerzasD
    global linea_FuerzasD
    inicio_FuerzasD=[]
    def inicio(evento):
        global longitudesFD
        global Valores_FD
        global F4
        global inicio_FuerzasD
        global linea_FuerzasD
        inicio_FuerzasD=[evento.x,evento.y]
    def final(evento):
        global longitudesFD
        global Valores_FD
        global F4
        global inicio_FuerzasD
        global linea_FuerzasD
        longitud=sqrt((evento.x-
        inicio_FuerzasD[0])**2+(evento.y-inicio_FuerzasD[1])**2)
        COS=(evento.x-inicio_FuerzasD[0])/float(longitud)
        SEN=-(evento.y-inicio_FuerzasD[1])/float(longitud)
        for i in range(20):
            dibujo.create_line(inicio_FuerzasD[0]+longitud/
```

```
            20*i*COS-50*SEN,inicio_FuerzasD[1]-
            longitud/20*i*SEN-50*COS,inicio_FuerzasD[0
            +longitud/20*COS*i,inicio_FuerzasD[1]-
            longitud/20*SEN*i,fill='green',arrow=LAST)
        dibujo.create_text(inicio_FuerzasD[0]-
        50*SEN,inicio_FuerzasD[1]-25*COS,
        text=Magnitud.get())
        linea_FuerzasD=linea_FuerzasD+[(inicio_FuerzasD[0]-50)/50,
        (550-inicio_FuerzasD[1])/50,(evento.x-50)/50,(550-evento.y)/50]
        longitudesFD=longitudesFD+
        [norm(array([(evento.x-50)/50,(550-evento.y)/50])-
        array([(inicio_FuerzasD[0]-50)/50,(550-inicio_FuerzasD[1])/50]))]
        F2=array(linea_FuerzasD)
        F3=F2.reshape(len(F2)/4,4)
        F4=F3
        Valores_FD=Valores_FD+[Magnitud.get()]
        print F4
        print Valores_FD
        print longitudesFD

        inicio_FuerzasD=None

ventana_distribuida=Toplevel()
ventana_distribuida.title('Fuerzas Distribuidas')
ventana_distribuida.geometry('400x200+0+0')
Im_Fdistribuida=PhotoImage(file='FFD.gif')
selec1=IntVar()
texto01=Label(ventana_distribuida,
text='Fuerzas Dist Magn :').place(x=100,y=20)
FPP=Radiobutton(ventana_distribuida,
image=Im_Fdistribuida,value=1,variable=selec1)
FPP.place(x=20,y=20)
Magnitud=DoubleVar()
entrada_general=Entry(ventana_distribuida,
textvariable=Magnitud,width=10).place(x=300,y=20)
```

```
    dibujo.bind('<Button-1>',inicio)
    dibujo.bind('<Button-3>',final)
    ventana_distribuida.mainloop()
def apoyos_universales():
    global codigo_apoyos
    global tipo_apoyos
    global tipo_apoyos1
    global codigo_apoyos1
    tipo_apoyos=[]
    codigo_apoyos=[]
    def apoyos_general(evento):
        global tipo_apoyos
        global codigo_apoyos
        global tipo_apoyos1
        global codigo_apoyos1
        if selec.get()==1:
            Apoyos(evento,dibujo).Aempotrados()
            tipo_apoyos=tipo_apoyos+[(evento.x-50)/50,(550-evento.y)/50]
            codigo_apoyos=codigo_apoyos+[1,1,1]
        elif selec.get()==2:
            Apoyos(evento,dibujo).Afijos()
            tipo_apoyos=tipo_apoyos+[(evento.x-50)/50,(550-evento.y)/50]
            codigo_apoyos=codigo_apoyos+[1,1,0]
        elif selec.get()==3:
            Apoyos(evento,dibujo).Amoviles()
            tipo_apoyos=tipo_apoyos+[(evento.x-50)/50,(550-evento.y)/50]
            codigo_apoyos=codigo_apoyos+[0,1,0]

        AP=array(tipo_apoyos)
        AM=array(codigo_apoyos)
        codigo_apoyos1=AM.reshape(len(AM)/3,3)
        tipo_apoyos1=AP.reshape(len(AP)/2,2)
        print tipo_apoyos1
        print codigo_apoyos1
    ventana_apoyos=Toplevel()
```

```
    #-------------------------------------------------------------------
    Im_Fijos=PhotoImage(file='fijo.gif')
    Im_Empotrados=PhotoImage(file='empotrado.gif')
    Im_Moviles=PhotoImage(file='movil.gif')
    #-------------------------------------------------------------------
    ventana_apoyos.title('Tipo-Apoyos')
    selec=IntVar()
    Texto_Empotrados=Label(ventana_apoyos,
    text='Apoyo Empotrado').place(x=100,y=20)
    Texto_Fjos=Label(ventana_apoyos,
    text='Apoyo Fijo').place(x=100,y=60)
    Texto_Moviles=Label(ventana_apoyos,
    text='Apoyo Moviles').place(x=100,y=100)
    Empotrados=Radiobutton(ventana_apoyos,
    image=Im_Empotrados,value=1,variable=selec)
    Empotrados.place(x=20,y=20)
    Fijos=Radiobutton(ventana_apoyos,
    image=Im_Fijos,value=2,variable=selec)
    Fijos.place(x=20,y=60)
    Moviles=Radiobutton(ventana_apoyos,
    image=Im_Moviles,value=3,variable=selec)
    Moviles.place(x=20,y=100)
    #-------------------------------------------------------------------
    #                    BOTON PARA INCLUIR APOYOS EN LOS PUNTOS
    #-------------------------------------------------------------------
    botton_01=Button(ventana_apoyos,text='Ejecutar',fg='blue').place(x=200,y
    dibujo.bind('<Button-1>', apoyos_general)
    ventana_apoyos.geometry('400x300+0+0')
    ventana_apoyos.minsize(400,300)
    ventana_apoyos.mainloop()

def Materiales_Calculo():
    global u2
    global numero
    global C3_Materiales
```

```
global nodos_espacio
nodos_espacio=u2
ventana_Materiales=Toplevel()
ventana_Materiales.title('Materiales de Elementos')
lista_1=['Inercia'+str(i) for i in range(len(u2)/2)]
lista_2=['Elasticidad'+str(i) for i in range(len(u2)/2)]
lista_3=['Area'+str(i) for i in range(len(u2)/2)]
ventana_Materiales.geometry('800x300+0+0')
numero=len(u2)/2

for i in range(len(u2)/2):
    lista_1[i]=DoubleVar()
    lista_2[i]=DoubleVar()
    lista_3[i]=DoubleVar()
for i in range(len(u2)/2):
    u1=Label(ventana_Materiales,text='Inercia'+str(i),
    fg='red').place(x=0,y=20+i*20)
    u2=Label(ventana_Materiales,text='Elasticidad'+str(i),
    fg='red').place(x=200,y=20+i*20)
    u3=Label(ventana_Materiales,text='Area'+str(i),
    fg='red').place(x=400,y=20+i*20)
for j in range(numero):
    Entrada_Inercia=Entry(ventana_Materiales,
    textvariable=lista_1[j],width=10).place(x=100,y=20+j*20)
    Entrada_Elasticidad=Entry(ventana_Materiales,
    textvariable=lista_2[j],width=10).place(x=300,y=20+j*20)
    Entrada_Area=Entry(ventana_Materiales,
    textvariable=lista_3[j],width=10).place(x=500,y=20+j*20)
#-----------------------------------------------------------------
C_Materiales=None
def guardar_datos():
    global C_Materiales
    global C3_Materiales
    C_Materiales=[]
    for t in range(numero):
```

```
            C_Materiales=C_Materiales+
            [lista_1[t].get()]+[lista_2[t].get()]+[lista_3[t].get()]
        C2_Materiales=array(C_Materiales)
        C3_Materiales=C2_Materiales.reshape(numero,3)
        print C3_Materiales

    #-----------------------------------------------------
    #BOTON DE GUARDAR CARACTERISTICAS DE LOS ELEMENTOS
    #-----------------------------------------------------
    Boton01=Button(ventana_Materiales,
    text='GUARDAR',command=guardar_datos).place(x=600,y=250)
    #-----------------------------------------------------------
#-------------------------------------------------------------
#
# FUNCIONES DE CALCULO DE LA MATRIZ DE RIGIDEZ -CALCULO
#          POR EL METODO DE ELEMENTOS FINITOS
#
#-------------------------------------------------------------
def nodos_calculo():
    global Valores_FD
    global F4
    global Puntos_FuerzasP1
    global Puntos_AFuerzasP1
    global codigo_apoyos1
    global tipo_apoyos1
    global ordenamiento
    global nodos_espacio
    global C
    global K_todos
    global C3_Materiales
    global T
    global K_Rtodos
    global longitudesFD
    numero_datos=len(nodos_espacio)
    numero_barras=numero_datos/2
```

```
C=[]
U=[]
lista_resumen=nodos_espacio.tolist()
for i in lista_resumen:
    if i not in C:
        C=C+[i]

print C
for s in range(numero_barras):
    K_local=Matrix2D(C3_Materiales[s][2],
    C3_Materiales[s][1],C3_Materiales[s][0],longitudes[s]).constructor()
    K_todos=K_todos+[K_local]
Coord_Barras=nodos_espacio.reshape(numero_barras,4)
for u in range(numero_barras):
    transformada_union=Transformada(Coord_Barras[u][0],
    Coord_Barras[u][1],Coord_Barras[u][2],Coord_Barras[u][3]).T()
    T=T+[transformada_union]
for r in range(numero_barras):
    K_ultimo_1=dot(K_todos[r],T[r])
    K_ultimo_2=dot(transpose(T[r]),K_ultimo_1)
    K_Rtodos=K_Rtodos+[K_ultimo_2]
ordenamiento={}#Biblioteca de calculo
ordenamiento_primario=[]#ordenamiento final de las barras
#-----------------------------------------------------
#MATRIZ DE ORDENAMIENTO----->PI
#-----------------------------------------------------
    ordenamiento[i]=C[i]
print ordenamiento
for i in range(numero_datos):
    for j in range(len(C)):
        if ordenamiento[j][0]==nodos_espacio[i][0]
        and ordenamiento[j][1]==nodos_espacio[i][1]:
            ordenamiento_primario.append(j)
print ordenamiento_primario
ordenamiento2=array(ordenamiento_primario)
```

```
    numero_filas=len(ordenamiento2)/2
    ultimo_ordenamiento=ordenamiento2.reshape(numero_filas,2)
    P_01S=ultimo_ordenamiento+1
    P_02S=P_01S*3
    WQ=array([range(P_02S[i][0]-3,
    P_02S[i][0]) for i in range(len(P_02S))])
    WR=array([range(P_02S[i][1]-3,
    P_02S[i][1]) for i in range(len(P_02S))])
    O=[]
    for i in range(len(WQ)):
        for j in range(len(WR)):
            if i==j:
                O=O+[array([WQ[i],WR[j]]).reshape(6)]

    P_03S=array(O)
    matriz_ordenamiento=transpose(P_03S)
    print matriz_ordenamiento
#-----------------------------------------------------------------------
#
# ACOPLAMIENTO FINAL DE LA MATRIZ DE RIGIDEZ TOTAL DE LA  ESTRUCTURA
#       k[PI[i][n],PI[j][n]]<------------------K{n}[i][n]
#-----------------------------------------------------------------------
    KT=zeros((3*len(C),3*len(C)))
    for i in range(6):
        for j in range(6):
            for n in range(len(K_Rtodos)):
                KT[matriz_ordenamiento[i][n]]
                [matriz_ordenamiento[j][n]]=
                KT[matriz_ordenamiento[i][n]]
                [matriz_ordenamiento[j][n]]+K_Rtodos[n][i][j]
    print KT
#-----------------------------------------------------------------------
#
#                                CALCULO DE MATRIZ REDUCIDA --> Ki
#
```

```
#-----------------------------------------------------------------------
    GDL01=[]
    for i in range(len(C)):
        for j in range(len(tipo_apoyos1)):
            if ordenamiento[i][0]==tipo_apo
            yos1[j][0] and ordenamiento[i][1]==tipo_apoyos1[j][1]:
                GDL01.append(i)
    GDL02=array(GDL01)
    GDL03=GDL02+1
    GDL04=GDL03*3
    MQ=array([range(GDL04[i]-3,GDL04[i])
    for i in range(len(GDL04))])
    EL=MQ*codigo_apoyos1
    K_reducida_inicial=delete(KT,EL,axis=0)
    K_reducida_final=delete
    (K_reducida_inicial,EL,axis=1)#calculo de K_reducida
    print K_reducida_final
#-------------------------------------------------------
#
#Programacion de Calculo de Fuerzas Puntuales
#
#-------------------------------------------------------
    esquema_fuerzasp=[]
    for i in range(len(ordenamiento)):
        for j in range(len(Puntos_AFuerzasP1)):
            if ordenamiento[i][0]==Puntos_AFuerzasP1[j][0]
            and ordenamiento[i][1]==Puntos_AFuerzasP1[j][1]:
                esquema_fuerzasp.append(i)
    print esquema_fuerzasp
    esquema_fuerzasp1=array(esquema_fuerzasp)
    esquema_fuerzasp2=(esquema_fuerzasp1+1)*3
    WR=array([range(esquema_fuerzasp2[i]-3,
    esquema_fuerzasp2[i]) for i in range(len(esquema_fuerzasp2))])
    WQ=WR.reshape(len(WR)*3,1)
    Fuerzas_puntuales=zeros((len(ordenamiento)*3,1))
```

```
    Puntos_FuerzasP2=Puntos_FuerzasP1.reshape(len(Puntos_FuerzasP1)*3,1)
    for i in range(len(Puntos_FuerzasP1)*3):
        Fuerzas_puntuales[WQ[i]]=Fuerzas_puntuales[WQ[i]]+Puntos_FuerzasP2[i
    print Fuerzas_puntuales
#-----------------------------------------------------------
#
#   Programacion de calculo de Fuerzas Distribuidas
#
#-----------------------------------------------------------
    F5=F4.reshape(len(F4)*2,2)
    esquema_FuerzasD=[]
    for i in range(len(ordenamiento)):
        for j in range(len(F5)):
            if ordenamiento[i][0]==F5[j][0] and ordenamiento[i][1]==F5[j][1]
                esquema_FuerzasD.append(i)
    esquema_FuerzasD1=array(esquema_FuerzasD)
    esquema_FuerzasD2=(esquema_FuerzasD1+1)*3
    WU=array([range(esquema_FuerzasD2[i]-3,
    esquema_FuerzasD2[i]) for i in range(len(esquema_FuerzasD2))])
    WP=WU.reshape(len(WU)*3,1)
    print 'este es la clave'+str(WP)

    Fuerzas_Distribuidas=zeros((len(ordenamiento)*3,1))
    Fuerzas_DD=[]
    for i in range(len(longitudesFD)):
        Fuerzas_DD=Fuerzas_DD+[[0,Valores_FD[i]*(longitudesFD[i])/float(2),V
        [0,Valores_FD[i]*(longitudesFD[i])
        /float(2),-Valores_FD[i]*(longitudesFD[i]**2)/float(12)]]
    FuerzasDD1=array(Fuerzas_DD)
    FuerzasDD2=FuerzasDD1.reshape(len(FuerzasDD1)*3,1)
    for i in range(len(FuerzasDD2)):
        Fuerzas_Distribuidas[WP[i]]=Fuerzas_Distribuidas[WP[i]]+FuerzasDD2[i
#-----------------------------------------------------------------------
#
#                        Programacion de Fuerzas Nodales Equivalentes
```

```
#
#-------------------------------------------------------------------------
    Fuerzas_nequivalentes=Fuerzas_puntuales-Fuerzas_Distribuidas
    Fuerzas_NodD=delete(Fuerzas_nequivalentes,EL,axis=0)
    print Fuerzas_NodD
#-------------------------------------------------------------------------
#
#                          Calculo de las deformaciones Nodales
#
#-------------------------------------------------------------------------
    Deformaciones01=dot(inv(K_reducida_final),Fuerzas_NodD)
    print Deformaciones01
    WUP=range(0,len(Fuerzas_nequivalentes))
    WUP1=array(WUP)
    WUP2=WUP1.reshape(len(WUP),1)
    WUP3=delete(WUP2,EL,axis=0)
    Deformaciones_totales=zeros((len(ordenamiento)*3,1))
    for i in range(len(WUP3)):
        Deformaciones_totales[WUP3[i]]=Deformaciones_totales[WUP3[i]]+Deform
    print Deformaciones_totales
#-------------------------------------------------------------------------
#
#                          Calculo de Fuerzas Internasx-Barras
#
#-------------------------------------------------------------------------
    for i in range(len(us01)):
        sx=us01[i].reshape(1,6)
        ordenamiento_fuerzasU=ordenamiento_fuerzasU+[sx]
    ordenamiento_ff=array(ordenamiento_fuerzasU).reshape(len(ordenamiento_fu
    SMR=array([[0.000000,0.000000,0.000000,0.000000,0.000000,0.000000]
    for i in range(len(ordenamiento_ff))])
    for i in range(len(ordenamiento_ff)):
        for j in range(6):
            SMR[i][j]=SMR[i][j]+
            Deformaciones_totales[ordenamiento_ff[i][j]]
```

```
print SMR
resultados_fuerzapb=[]
for i in range(len(SMR)):
    resultados_fuerzapb=resultados_fuerzapb+
    [dot(K_Rtodos[i],SMR[i].reshape(6,1))]
print resultados_fuerzapb
restar_altotal=Fuerzas_nequivalentes-Fuerzas_puntuales
SMR1=array([[0.000000,0.000000,0.000000,0.000000,0.000000,0.000000]
for i in range(len(ordenamiento_ff))])
WNN=WP.reshape(len(WP)/6.00,6.00)
for i in range(len(ordenamiento_ff)):
    for j in range(len(WNN)):
        if ordenamiento_ff[i][0]==WNN[j][0] and
        ordenamiento_ff[i][1]==WNN[j][1] and
        ordenamiento_ff[i][2]==WNN[j][2] and
        ordenamiento_ff[i][3]==WNN[j][3] and
        ordenamiento_ff[i][4]==WNN[j][4] and ordenamiento_ff[i][5]==WNN[
            SMR1[i][0]=ordenamiento_ff[i][0]
            SMR1[i][1]=ordenamiento_ff[i][1]
            SMR1[i][2]=ordenamiento_ff[i][2]
            SMR1[i][3]=ordenamiento_ff[i][3]
            SMR1[i][4]=ordenamiento_ff[i][4]
            SMR1[i][5]=ordenamiento_ff[i][5]
        else:
            SMR1[i][0]=0.000000000
            SMR1[i][1]=0.000000000
            SMR1[i][2]=0.000000000
            SMR1[i][3]=0.000000000
            SMR1[i][4]=0.000000000
            SMR1[i][5]=0.000000000
SMR2=array([[0.000000,0.000000,0.000000,0.000000,0.000000,0.000000]
for i in range(len(ordenamiento_ff))])

for i in range(len(ordenamiento_ff)):
```

```
        for j in range(6):
            SMR2[i][j]=SMR2[i][j]+restar_altotal[SMR1[i][j]]



    fuerzas_bfinal=[]
    for i in range(len(ordenamiento_ff)):
        fuerzas_bfinal=fuerzas_bfinal+[resultados_fuerzapb[i]-SMR2[i].reshap
    print fuerzas_bfinal
#-----------------------------------------------------------------------
#
#                          Algoritmo de Graficas de u,a_{i},M_{i},V_{i}
#
#-----------------------------------------------------------------------
    X=[]
    for i in range(len(ordenamiento_ff)):
        xi=arange(0,longitudes[i],0.1)
        X=X+[xi]
    print X
    VVUW=nodos_espacio.reshape(len(nodos_espacio)/2,4)
    P0=[]
    for i in range(len(VVUW)):
        for j in range(len(F4)):
            if VVUW[i][0]==F4[j][0] and VVUW[i][1]==F4[j][1]
            and VVUW[i][2]==F4[j][2] and VVUW[i][3]==F4[j][3]:
                P0.append(-Valores_FD[j])
            else:
                P0.append(0)
    F_DD=array(P0).reshape(len(P0),1)
    Form=[]
    for i in range(len(ordenamiento_ff)):
        Form=Form+[Forma_GG(X[i],longitudes[i],
        C3_Materiales[i][1],C3_Materiales[i][0],F_DD[i])]
    DEFF=[]
    for i in range(len(ordenamiento_ff)):
        DEFF=DEFF+[Form[i].N_1()*SMR[i][0]+
```

```
        Form[i].N_2()*SMR[i][1]+Form[i].N_3()*SMR[i][2]
        +Form[i].N_4()*SMR[i][3]+Form[i].N_5()*SMR[i][4]+
        Form[i].N_6()*SMR[i][5]+Form[i].R_1()]
    print DEFF
    DEFG=[]
    for i in range(len(ordenamiento_ff)):
        DEFG=DEFG+[(Form[i].W_1()*SMR[i][0]+
        Form[i].W_2()*SMR[i][1]+Form[i].W_3()*SMR[i][2]+
        Form[i].W_4()*SMR[i][3]+Form[i].W_5()*SMR[i][4]
        +Form[i].W_6()*SMR[i][5]+Form[i].R_2())]
    print DEFG
    Momenx=[]
    for i in range(len(ordenamiento_ff)):
        Momenx=Momenx+[-1*(Form[i].M_1()*SMR[i][0]+
        Form[i].M_2()*SMR[i][1]+Form[i].M_3()*SMR[i][2]+
        Form[i].M_4()*SMR[i][3]+Form[i].M_5()*SMR[i][4]+
        Form[i].M_6()*SMR[i][5]+Form[i].R_3())]
    print Momenx
    Cort=[]
    for i in range(len(ordenamiento_ff)):
        Cort=Cort+[Form[i].V_1()*SMR[i][0]+
        Form[i].V_2()*SMR[i][1]+Form[i].V_3()*SMR[i][2]+
        Form[i].V_4()*SMR[i][3]+Form[i].V_5()*SMR[i][4]+
        Form[i].V_6()*SMR[i][5]+Form[i].R_4()]
    print Cort
    print longitudes


#-----------------------------------------------------------------------
#
#                        Algoritmo de Presentacion de Resultados.
#
#-----------------------------------------------------------------------
    def Diagramasfinales():
        def Diagrama():
            fiig=figure(1)
```

```
            TDef=fiig.add_subplot(221)
            title(r'$D_{i}$')
            grid(True)
            xlabel(r'$x_{i}$')
            ylabel(r'$\delta$')
            TDef.plot(X[m.get()],DEFF[m.get()],'r--')
            TGir=fiig.add_subplot(222)
            title(r'$\theta_{i}$')
            xlabel(r'$x_{i}$')
            ylabel(r'$\theta_{i}$')
            grid(True)
            TGir.plot(X[m.get()],DEFG[m.get()],'r--')
            TMom=fiig.add_subplot(223)
            title(r'$M_{i}$')
            xlabel(r'$x_{i}$')
            ylabel(r'$M_{i}$')
            grid(True)
            TMom.plot(X[m.get()],Momenx[m.get()],'r--')
            TCort=fiig.add_subplot(224)
            title(r'$V_{i}$')
            xlabel(r'$x_{i}$')
            ylabel(r'$V_{i}$')
            grid(True)
            TCort.plot(X[m.get()],Cort[m.get()],'r--')
            show()
        ven01=Toplevel()
        ven01.title('Diagramas de las Barras')
        textoMM=Label(ven01,text='Elija la Barra :',fg='blue').place(x=20,y=
        m=IntVar()
        EntradaMM=Entry(ven01,textvariable=m,width=10).place(x=100,y=20)
        BotonMM=Button(ven01,text='ejecuta',command=Diagrama).place(x=200,y=
        ven01.mainloop()
    def Fuer_Int():
        def VFuerzas01():
            ventana_alterna01=Toplevel()
```

```
            texto=Label(ventana_alterna01,text=fuerzas_bfinal[n.get()]).place
            ventana_alterna01.title('Fuerzas Int por Barras.')
            ventana_alterna01.mainloop()
        ventana_fuerzasint=Toplevel()
        texto_01=Label(ventana_fuerzasint,text='Elegir Barra :').place(x=20,
        n=IntVar()
        entrada01=Entry(ventana_fuerzasint,textvariable=n,width=5).place(x=1
        Boton01=Button(ventana_fuerzasint,
        text='ejecuta',command=VFuerzas01).place(x=150,y=20)
        ventana_fuerzasint.mainloop()
    def Deformaciones_FFU():
        ventana_Deformaciones=Toplevel()
        for i in range(len(Deformaciones_totales)):
            Label(ventana_Deformaciones,text='Para el Punto :').place(x=20,y
            Label(ventana_Deformaciones,text=i+1).place(x=150,y=i*20+20)
            Label(ventana_Deformaciones,
            text=Deformaciones_totales[i]).place(x=180,y=i*20+20)


        ventana_Deformaciones.mainloop()
    def Matriz_RRIG():
        ventana_matrix=Toplevel()
        ventana_matrix.geometry('600x600+0+0')
        ventana_matrix.title('Matriz de Rigidez')
        textos=Canvas(ventana_matrix,width=400,
        height=400,scrollregion=(0,0,2000,2000),
        highlightcolor='black',relief='solid',background='white')
        for i in range(len(KT)):
            for j in range(len(KT)):
                textos.create_text(100*i+40,100*j+40,text=KT[i][j])
        for i in range(len(KT)):
            textos.create_text(100*i+40,10,text=i+1,fill='red')
            textos.create_text(10,100*i+40,text=i+1,fill='red')
        textos01=Label(ventana_matrix,text='CALCULO DE MATRIZ GENERAL',fg='r
        textos01.grid(row=1,column=3)
```

```
        scrollY=Scrollbar(ventana_matrix,orient=VERTICAL,command=textos.yviev
        scrollY.grid(row=1,column=2,sticky=N+S)
        scrollX=Scrollbar(ventana_matrix,orient=HORIZONTAL,command=textos.xv
        scrollX.grid(row=2,column=1,sticky=E+W)
        textos['xscrollcommand']=scrollX.set
        textos['yscrollcommand']=scrollY.set
        textos.grid(row=1,column=1)

        ventana_matrix.mainloop()
    def Matriz_DMM():
        ventana_matrixD=Toplevel()
        ventana_matrixD.geometry('600x600+0+0')
        ventana_matrixD.title('Matriz de Rigidez Simplificada')
        textos1=Canvas(ventana_matrixD,width=400,height=400,
        scrollregion=(0,0,2000,2000),
        highlightcolor='black',relief='solid',background='white')
        for i in range(len(K_reducida_final)):
            for j in range(len(K_reducida_final)):
                textos1.create_text(100*i+40,100*j+40,text=K_reducida_final[
        for i in range(len(K_reducida_final)):
            textos1.create_text(100*i+40,10,text=i+1,fill='red')
            textos1.create_text(10,100*i+40,text=i+1,fill='red')
        textos02=Label(ventana_matrixD,text='CALCULO DE MATRIZ SIMPLIFICADA
        textos02.grid(row=1,column=3)
        scrollY=Scrollbar(ventana_matrixD,orient=VERTICAL,command=textos1.y
        scrollY.grid(row=1,column=2,sticky=N+S)
        scrollX=Scrollbar(ventana_matrixD,orient=HORIZONTAL,command=textos1
        scrollX.grid(row=2,column=1,sticky=E+W)
        textos1['xscrollcommand']=scrollX.set
        textos1['yscrollcommand']=scrollY.set
        textos1.grid(row=1,column=1)

        ventana_matrixD.mainloop()
    ventana_Resultados=Toplevel()
    ventana_Resultados.geometry('600x400+0+0')
```

```
    ventana_Resultados.title('Resultados de Analisis')
    texto_01=Label(ventana_Resultados,text='Deformaciones Internas :',fg='bl
    Boton01=Button(ventana_Resultados,text='Ejecutar',
    fg='red',command=Deformaciones_FFU).place(x=180,y=20)
    texto_02=Label(ventana_Resultados,
    text='Matriz de rigidez',fg='blue').place(x=20,y=50)
    Boton02=Button(ventana_Resultados,text='Ejecutar'
    ,fg='red',command=Matriz_RRIG).place(x=180,y=50)
    texto03=Label(ventana_Resultados,text='Matriz de Rigidez Simplificada',f
    Boton03=Button(ventana_Resultados,text='Ejecutar',
    command=Matriz_DMM,fg='red').place(x=180,y=80)
    texto04=Label(ventana_Resultados,
    text='Fuerzas Internas',fg='blue').place(x=20,y=110)
    Boton04=Button(ventana_Resultados,
    text='Ejecutar',
    command=Fuer_Int,fg='red').place(x=180,y=110)
    texto05=Label(ventana_Resultados,
    text='Diagramas',fg='blue').place(x=20,y=140)
    Boton05=Button(ventana_Resultados,text='ejecuta',
    command=Diagramasfinales,fg='red').place(x=180,y=140)
    ventana_Resultados.mainloop()

def grilla():
    global Ubicaciones_Universales
    Ubicaciones_Universales=[]
    def dibujo_grilla():
        if nx.get()!=0 and ny.get()!=0:
            for i in range(nx.get()+1):
                dibujo.create_line(50+i*mx.get()*50,550,50+
                i*mx.get()*50,550-ny.get()*my.get()*50,
                fill='gray',activefill='red',dash=(3,5),tag=('Recta01'))
            for i in range(ny.get()+1):
                dibujo.create_line(50,550-i*my.get()
                *50,50+(nx.get())*mx.get()*50,
                550-i*my.get()*50,fill='gray',activefill='red'
```

```
                ,dash=(3,5),tag=('Recta02'))
            for u in range(nx.get()+1):
                for m in range(ny.get()+1):
                    Ubicaciones_Universales.append([50+u*mx.get()
                    *50,550-m*my.get()*50])
                    dibujo.create_oval(50+u*mx.get()*50-2,
                    550-m*my.get()*50-2,50+u*mx.get()
                    *50+2,550-m*my.get()*50+2,
                    fill='red',activefill='blue')


        else:
            tkMessageBox.showinfo(message='Por Favor poner Datos')

    ventana_grilla=Toplevel()
    ventana_grilla.title('DIBUJAR GRILLA P/ESQUEMA')
    ventana_grilla.minsize(500,200)
    ventana_grilla.geometry('500x200+0+0')
    #-----------------------------------------------------
    nx=IntVar()
    mx=DoubleVar()
    ny=IntVar()
    my=DoubleVar()
    #-----------------------------------------------------
    texto_principal=Label(ventana_grilla,
    text='COORDENADAS EN EL EJE X:Y').place(x=100,y=0)
    Texto_01=Label(ventana_grilla,text='N:Ejes-X').place(x=10,y=20)
    texto_02=Label(ventana_grilla,text='Dist-Ejes-X').place(x=250,y=20)
    texto03=Label(ventana_grilla,text='N:Ejes-Y').place(x=10,y=60)
    texto04=Label(ventana_grilla,text='Dist-Ejes-Y').place(x=250,y=60)
    #-----------------------------------------------------
    Ejes_Horizontales=Entry(ventana_grilla,textvariable=nx).place(x=80,y=20)
    Distancias_Horizontales=Entry(ventana_grilla,textvariable=mx).place(x=32
    #-----------------------------------------------------
    Ejes_Verticales=Entry(ventana_grilla,textvariable=ny).place(x=80,y=60)
```

```
    Distancias_Verticales=Entry(ventana_grilla,textvariable=my).place(x=320,
    Boton01=Button(ventana_grilla,text='APLICAR',
    command=dibujo_grilla).place(x=300,y=150)
    #-----------------------------------------------
    Boton02=Button(ventana_grilla,text='CANCELAR',
    command=ventana_grilla.destroy).place(x=400,y=150)
    #-------------------------------------------------------------------
    #VENTANA DE GRILLAS-LOKITOSAMAX-ELEMENTOS DE VARIABLE
    #-------------------------------------------------------------------
    ventana_grilla.mainloop()



def Borrador():
    dibujo.delete(ALL)
def dibuja_barra():
    tkMessageBox.showinfo(message='Poner el Primero y Segundo punto.')
    global puntos
    global longitudes
    global u2
    global u
    start =None
    def punto(evento):
        global start
        global u

        start=[evento.x,evento.y]

    def punto2(evento):
        global puntos
        global longitudes
        global start
        global u
        global u2
        if start is not None:
```

```
            x=start[0]
            y=start[1]
            dibujo.create_line(x,y,evento.x,evento.y,width=4.5,
            arrow=LAST,activefill='gray',tag=('linea1'))
            dibujo.itemconfig('linea1', fill='red')
            u=u+[(x-50)/50,(550-y)/50,(evento.x-50)/50,(550-evento.y)/50]
            longitudes=longitudes+[norm(array([(evento.x-50)/50,
            (550-evento.y)/50])-array([(x-50)/50,(550-y)/50]))]
            start=None


    dibujo.bind('<Button-1>',punto)
    dibujo.bind('<Button-3>',punto2)
    u1=array(u)
    l1=len(u1)
    u2=u1.reshape(l1/2,2)
    print u2
    print longitudes


def kill_XY(evento=None):
    dibujo.delete('no')
def Kill_2(evento=None):
    dibujo.delete('cuadradito')


X,Y=None,None
def coordenadas(evento):
    global Ubicaciones_Universales
    global X,Y
    kill_XY()
    X=dibujo.create_line(evento.x,0,evento.x,1000,dash=[3,2],tags='no',fill=
    Y=dibujo.create_line(0,evento.y,1000,evento.y,dash=[3,2],tags='no',fill=

    global cuadrado
```

```
    Coordenada_mause['text']='x->
    '+str((evento.x-50)/50)+'y->'+str((550-evento.y)/50)
    m=array([evento.x,evento.y])

    if Ubicaciones_Universales is not None:
        for s in range(len(Ubicaciones_Universales)):
            if evento.x==
            Ubicaciones_Universales[s][0] and
            evento.y==Ubicaciones_Universales[s][1]:
                dibujo.create_rectangle
                (Ubicaciones_Universales[s][0]-10,Ubicaciones_Universales[s]
                -10,Ubicaciones_Universales[s][0]
                +10,Ubicaciones_Universales[s][1]
                +10,outline='blue',tag='Cuadrado_Identificado'+str(s))
            else:
                dibujo.delete('Cuadrado_Identificado'+str(s))


#-------------------------------------------------------------------
ventana=Tk()
ventana.title('FEMAX')
ventana.minsize(1200,800)
ventana.geometry('1200x800+0+0')
barra_dibujo=Frame(ventana,width=200,height=100)

barra_dibujo.grid(row=0,column=4)

#-------------------------------------------------------------------
dibujo=Canvas(ventana,width=950,height=600,scrollregion=(-2000,-2000,2000,20
highlightcolor='black',relief='solid',background='white')



#-------------------------------------------------------------------
dibujo.bind('<Motion>',coordenadas)
```

```
flecha_X=dibujo.create_line(50,550,100,550,arrow=LAST,fill='blue')
X_sumate=dibujo.create_text(125,550,text='Eje_X',fill='blue')
flecha_Y=dibujo.create_line(50,550,50,500,arrow=LAST,fill='blue')
X_sumate=dibujo.create_text(50,480,text='Eje_Y',fill='blue')
#-----------------------------------------------------------------------
dibujo.grid(row=1,column=1)

scrollY=Scrollbar(ventana,orient=VERTICAL,command=dibujo.yview)
scrollY.grid(row=1,column=2,sticky=N+S)
scrollX=Scrollbar(ventana,orient=HORIZONTAL,command=dibujo.xview)
scrollX.grid(row=2,column=1,sticky=E+W)
dibujo['xscrollcommand']=scrollX.set
dibujo['yscrollcommand']=scrollY.set
#-----------------------------------------------------------------------
#
#  BOTONES Y CONTROLES GENERALES DEL PROGRAMA EN GENERAL----------->LO
#
#-----------------------------------------------------------------------
'''TITULO DEL PROGRAMA EN GENERAL'''
Texto_General=Label(ventana,
text='PROGRAMA DE ESTRUCTURAS -HECHO EN 01/2014-UNIVERSIDAD NACIONAL DE CAJA
fg='blue')
Texto_General.grid(row=0,column=1)
#-----------------------------------------------------------------------
Im_barra=PhotoImage(file='frame.gif')
Im_grilla=PhotoImage(file='grilla.gif')
Im_Borrador=PhotoImage(file='borrador.gif')
Im_Materiales=PhotoImage(file='materiales_damier.gif')
Im_Procesos=PhotoImage(file='proceso.gif')
Im_Apoyosgeneral=PhotoImage(file='movil.gif')
Im_Fuerzas1=PhotoImage(file='FP.gif')
Im_Fuerzas2=PhotoImage(file='FD.gif')
#-----------------------------------------------------------------------
barra_nodal=Button(barra_dibujo,image=Im_barra,
command=dibuja_barra,width=30,height=30).place(x=20,y=20)
```

```
grillas_nodal=Button(barra_dibujo,image=Im_grilla,
command=grilla,width=30,height=30).place(x=55,y=20)
Borrar=Button(barra_dibujo,image=Im_Borrador,
command=Borrador,width=30,height=30).place(x=20,y=60)
Materiales_Elementos=Button(barra_dibujo,
image=Im_Materiales,width=30,height=30,
command=Materiales_Calculo).place(x=55,y=60)
Prueba_ultima=Button(barra_dibujo,image=Im_Procesos,
command=nodos_calculo,width=30,height=30).place(x=90,y=20)
Boton_Apoyos=Button(barra_dibujo,image=Im_Apoyosgeneral,
command=apoyos_universales,width=30,height=30).place(x=90,y=60)
Boton_Fuerzas1=Button(barra_dibujo,image=
Im_Fuerzas1,width=30,height=30,
command=Fuerzas_puntuales).place(x=130,y=20)
Boton_Fuerzas2=Button(barra_dibujo,image=
Im_Fuerzas2,width=30,height=30,
command=Fuerzas_distribuidas).place(x=130,y=60)
#------------------------------------------------------------
'''MODELADO DEL MAUSE PRIMEROS PROPIEDADES DEL MOUSE'''

Coordenada_mause=Label(ventana)
Coordenada_mause.grid(row=4,column=1)
ventana.mainloop()
```

Capítulo 11

Figuras y Otros.

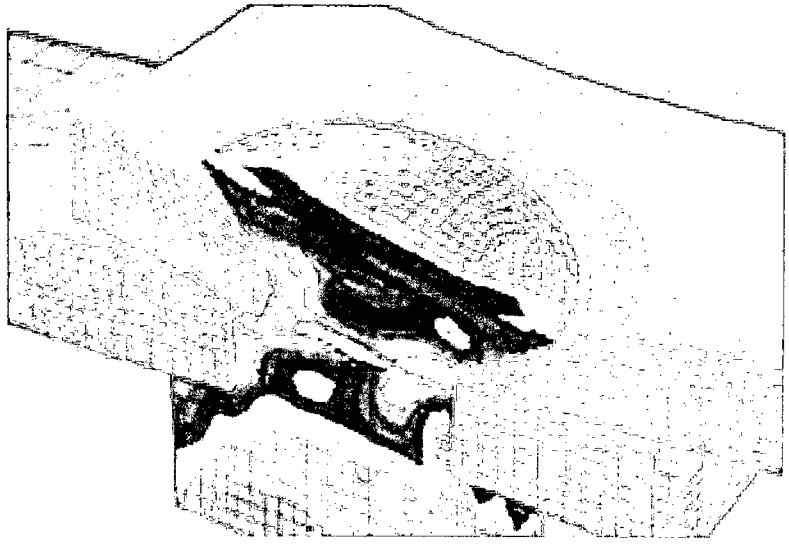


Figura 11.1: Interacción suelo Estructura.

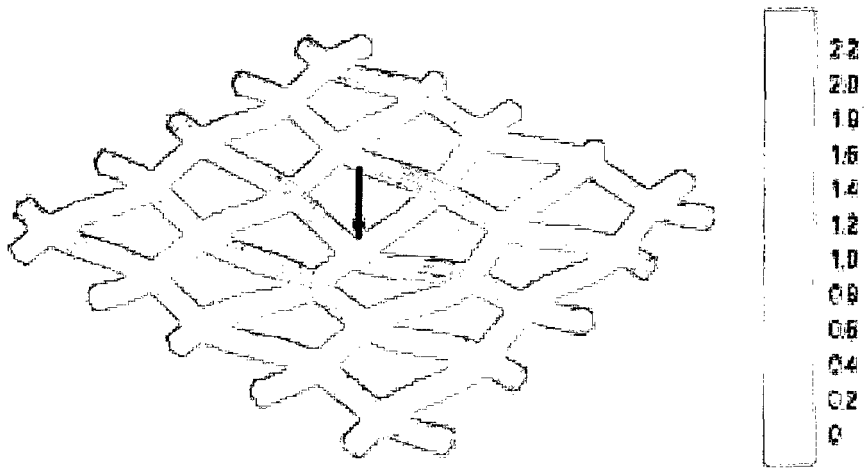


Figura 11.2: Aplicaciones del FEM(Finite Element Method.)

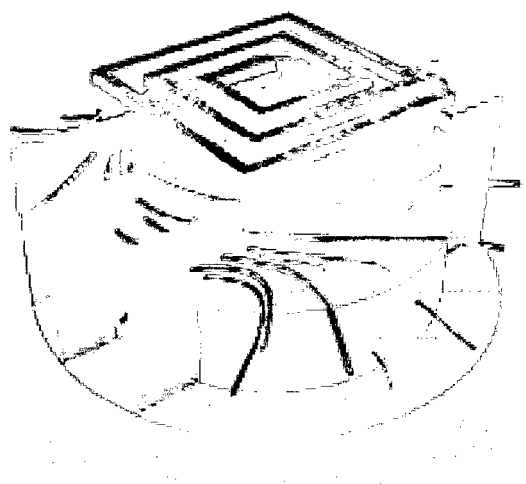


Figura 11.3: Mecánica de Fluidos FEM(Finite Element Method)

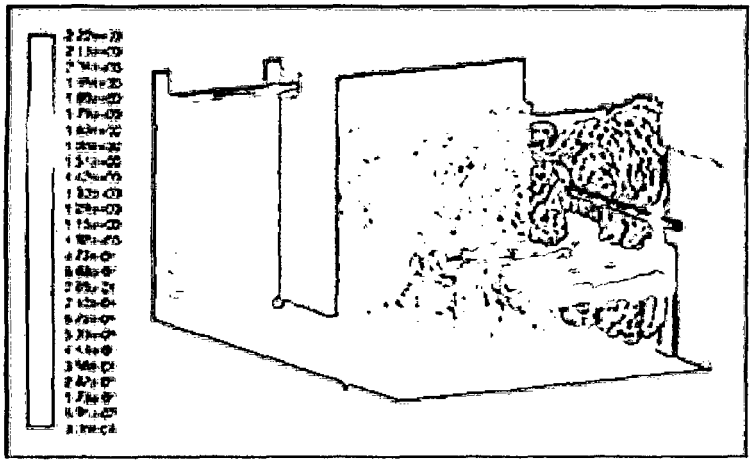


Figura 11.4: Idealización de un sistema de Fluidos.

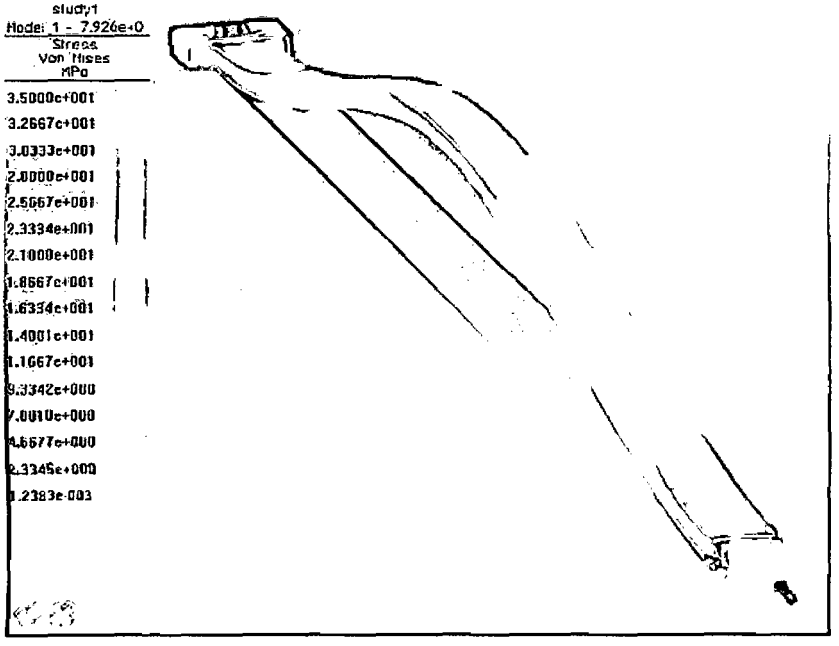


Figura 11.5: Puente Idealizado con Elementos Finitos.

Bibliografía

[1] H. Partl: Perez Villar Luis Alberto,Tesis" Análisis de Estructuras por Método de Elementos finitos Asistido por computadora" IPN,(2003)

[2] H. partl: O.CZienkiewichz-R.LTaylor,El Método de Elementos Finitos,McGraw-Hill,(1998)

[3] H. partII: Eugenio Oñate,Cálculo de Estructuras por el método de elementos finitos Análisis Estático Lineal,C.I de Métodos Numéricos en Ingeniería UPC,(1992)

[4] H. partII: Sergio Gallegos Cázares,Análisis de Sólidos y Estructural mediante el método de Elementos finitos,Tecnológico de Monterrey-LIMUSA,(2008)

[5] H. partII: Sandro Tosi,Matplotlib for Python Develops,Birmingham-Noviembre 2009.

[6] H. partII: Egor P.Popov(University of California-Berkeley),Mecánica de Sólidos,Educational-Person,Edition(2000)

[7] H. partII: Manuel Vasquez,Eloiza Lopez,El Método de Elementos Finitos Aplicado al análisis Estructural.

[8] H. partII: Arturo Tenna Colunga ,Análisis de Estructuras con Métodos Matriciales,Limusa México-2007.

[9] H. partII: Manuel Casado Martin,Lenguaje de Programación Python Orientado a Objetos,Santiago Gualadajara Perez,Universidad de Salamanca.

[10] H. partII: Gabriel Valiente Feruglio,Composición de textos científicos con LaTeX