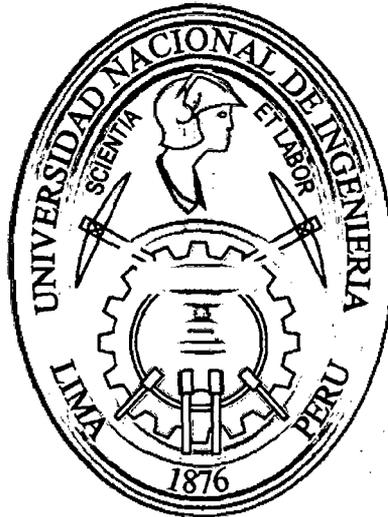


**UNIVERSIDAD NACIONAL DE INGENIERIA**  
**FACULTAD DE INGENIERIA CIVIL**



**CALCULO DE ESFUERZOS Y DEFORMACIONES EN ZONAS  
DE CONTACTO ENTRE CUERPOS ISOTROPICOS  
ESTRUCTURALES BIDIMENSIONALES POR EL MÉTODO  
DE ELEMENTOS FINITOS**

**TESIS**

**Para optar el Título Profesional de:**

**INGENIERO CIVIL**

**JULIO EDGAR GARCIA VILLANUEVA**

**Lima - Perú**

**2010**

**Digitalizado por:**

**Consortio Digital del  
Conocimiento MebLatam,  
Hemisferio y Dalse**

*Este trabajo es el fruto de un sacrificio y esfuerzo durante varios años y que cada vez se vino prolongando. Por tal motivo dedico esta Tesis a:*

A la mujer que me dio la vida y me regala mucho amor cada día.

*Para ti Mami, este reconocimiento por tu gran esfuerzo y dedicación en mi educación.*

Al hombre que comprende todos mis errores en la vida.

*Para ti Omar, que a pesar de ser mí hermano eres un padre para mí.*

A la mujer que me contagia su alegría.

*Para ti Chave, que desde que nací te veo como mi hermana.*

A mis primos, que en una etapa de mi vida me inculcaron valores y educación.

*Para ti Segundo, gracias por tu rectitud que ahora se deja extrañar.*

*Para ti Marco (En memoria), por todos los recuerdos que nos has dejado.*

A mis tíos que me abrieron las puertas de su familia y de su corazón cuando deje mí casa por venir a estudiar.

*Para ti Papa Marcial (En memoria), por enseñarme algo de la vida a través de tu historia cuando estamos lejos de casa.*

*Para ti Mama Balbina y Mama Aguedita, por darme ese calor familiar.*

## **AGRADECIMIENTO**

La gratitud es uno de los valores que me inculcaron, por eso quiero dejar testimonio de mi agradecimiento a:

A aquel ser espiritual que sé que me acompaña en la vida. Gracias Dios por estar allí.

A toda mi familia por su gran apoyo moral y que llevo una profunda gratitud.

A todas las personas que conocí y que me han acompañado en esta etapa de mi vida, de quienes llevo gratos recuerdos.

A todos mis profesores, quienes algunos resaltan más por su profunda dedicación en querer compartir sus conocimientos.

Al Dr. Carlos Zavala Toledo, por la confianza depositada en mí y cuyo asesoramiento sirvió para elaborar el presente trabajo.

Al Ing. Leonardo Flores Gonzales por su atención incondicional a mis consultas.

A los profesores de la sección de Post-Grado por su orientación académica, en especial al Dr. Daniel Huaco Oviedo, Dr. Luis Vásquez Chicata, Dr. Hugo Scaletti Farina.

**INDICE**

<b>RESUMEN</b>	<b>3</b>
<b>LISTA DE FIGURAS</b>	<b>5</b>
<b>INTRODUCCION</b>	<b>9</b>
<b>1.0.- ASPECTOS TEÓRICOS BÁSICOS .....</b>	<b>11</b>
1.1.- MECÁNICA DE MEDIOS CONTINUOS	11
1.1.1.- Análisis de Deformación	12
1.1.2.- Análisis de Tensiones	15
1.1.3.- Elasticidad	17
1.2.- MECANICA DEL CONTACTO	19
1.2.1.- Clasificación de Contacto	20
1.2.2.- Cargas de Contacto	22
1.2.3.- Modelo de Contacto de Hertz	27
1.3.- PRINCIPIO VARIACIONAL	28
1.3.1.- Funcional de Mínima Energía Potencial	30
1.3.2.- Funcional de Energía Potencial Complementaria	31
1.3.3.- Funcional de Reissner	32
1.4.- GENERADORES DE MALLAS	33
1.4.1.- Triangulación de Delaunay	35
1.5.- ALISADOS DE TENSIONES	38
1.5.1.- Métodos de Alisados	39
1.6.- DESARROLLO DE SISTEMAS DE ECUACIONES	40
<b>2.0.- MÉTODO DE ELEMENTO FINITO .....</b>	<b>42</b>
2.1.- DESARROLLO GEOMETRICO DEL MODELO	43
2.2.- FORMULACION DE ECUACIONES DE GOBIERNO	46
2.2.1.- Campo de Desplazamientos Asumidos	46
2.2.2.- Ecuaciones de Gobierno	48
2.3.- DISCRETIZACION DE ECUACIONES	49
2.3.1.- Funciones de Forma	50
2.3.2.- Elementos Isoparamétricos Bidimensionales	56
2.3.3.- Matriz de Deformación	58
2.3.4.- Integración Numérica	59

---

2.4.- SOLUCION DE ECUACIONES	60
2.5.- INTERPRETACION DE LOS RESULTADOS	62
<b>3.0.- MECANISMOS Y OPERADORES DE CONTACTO .....</b>	<b>65</b>
3.1.- MECANISMOS DE CONTACTO	65
3.1.1.- Condición de Contacto	65
3.2.- METODO DE REGULACIÓN	69
3.3.- ALGORITMOS DE INTERACCION	75
3.3.1.- Master-Slave	75
3.3.2.- Mortar Element	78
3.3.3.- Splitting Pinballs	78
3.4.- OPERADORES DE CONTACTO	79
<b>4.0.- EJEMPLOS APLICATIVOS .....</b>	<b>83</b>
4.1.- CONTACTO DE HERTZ	84
4.2.- SISTEMA APORTICADO	96
4.3.- VIGAS EN VOLADIZO	109
<b>CONCLUSIONES</b>	<b>118</b>
<b>RECOMENDACIONES</b>	<b>119</b>
<b>BIBLIOGRAFIA</b>	<b>120</b>
<b>ANEXOS</b>	<b>123</b>

## RESUMEN

La interacción o el simple apoyo de un cuerpo sobre otro, hace presenciar un tipo de análisis estructural. Las nuevas generaciones de metodologías para calcular desplazamientos y/o esfuerzos con mayor precisión hacen que el análisis estructural sea cada vez más confiable.

La presente tesis tiene como finalidad principal en analizar y cuantificar los valores de los esfuerzos y desplazamientos cuando dos cuerpos interactúan entre sí, originando de esta manera una zona de contacto. Las consideraciones a priori de estos cuerpos en estudio son: Deben ser analizados en un ambiente bidimensional, deben de tener un comportamiento isotrópico y elástico, no deben tener resistencia a la fricción y por último, las fuerzas externas no deben generar grandes deformaciones.

Para ello se ha implementado un programa computacional haciendo uso del Método del Elemento Finito (MEF) con mallas triangulares de tres y seis nudos, adicionando además el algoritmo de contacto llamado "Master-Slave", este elemento de contacto hace uso de una fuerza adicional cada vez que se incumple la condición de impenetrabilidad a través del operador de contacto "Nudo-Nudo", dicha fuerza está relacionada con una rigidez que tiene un valor variable. Existen otros operadores de contacto así como también otros algoritmos pero tienen cierto grado de dificultad en implementarlos.

La implementación complementaria del análisis No-Lineal Geométrico ayuda a comprender y a visualizar mejor el comportamiento del contacto.

El Principio Variacional usado en la implementación del elemento finito tiene cierta limitación en el cálculo de los esfuerzos, para ello se procedió a mejorar y reducir dicho error mediante el Alisado de Tensiones y calcular el Parámetro de Error Global del sistema.

El programa computacional implementado tiene tres rutinas bien definidas; el pre-procesado, el procesador y el post-procesador; codificadas en los lenguajes de programación del Visual C++ y Visual Basic. En el caso del Pre-Procesador, tiene como finalidad principal de generar el archivo data, en este archivo se describe las características geométricas y del material, las condiciones de borde, las cargas externas aplicadas y la descripción de los elementos de contacto. Cabe mencionar que en esta rutina resalta la función de la Generación de Mallas Triangulares a partir de una nube de puntos (variante de la Triangulación de

Delaunay). En el caso del Procesador, tiene la mayor importancia en todo el desarrollo de programa ya que en esta está los algoritmos de contacto así como también el algoritmo de resolución de ecuaciones, el algoritmo de alisado de tensiones (método promedio directo y método global) y el cálculo de error en los esfuerzos (parámetros de error global y de refinamiento de cada elemento triangular). Finalmente en el caso del Post-Procesador, su función es la visualización de los resultados de manera gráfica para interpretar mejor los resultados.

Para poder comprender mejor el efecto de contacto entre dos cuerpos, se presentarán tres ejemplos aplicativos en la que el primer ejemplo nos servirá para validar los resultados, los otros dos ejemplos restantes son de simple aplicación.

El uso de otros programas computacionales como el SAP2000 y el ANSYS que contienen mejores y/o distintos algoritmos matemáticos, hacen validar de alguna manera los resultados obtenidos en la presente tesis.

## LISTA DE FIGURAS

Figura 1.1: Estado de Deformación, Tensor Gradiente de Deformación	12
Figura 1.2: Estado de Deformación, Vector Desplazamiento	13
Figura 1.3: Estado de Deformación, Longitud Diferencial	13
Figura 1.4: Fuerzas Másicas y Superficiales	15
Figura 1.5: Tetraedro elemental alrededor de un punto P	15
Figura 1.6: Representación Gráfica y Matricial del Estado Tensional	16
Figura 1.7: Representación Gráfica y Matricial de la Elasticidad Plana	19
Figura 1.8: Tipos de Contacto	22
Figura 1.9: Fuerza sobre un punto de Contacto	22
Figura 1.10: Línea de Fuerza sobre un plano	23
Figura 1.11: Línea de Carga perpendicular al plano xy	25
Figura 1.12: Línea de Carga perpendicular al plano yz	25
Figura 1.13: Franja de Carga perpendicular al plano xy	26
Figura 1.14: Distribución de Carga Uniforme	26
Figura 1.15: Modelo de Hertz	28
Figura 1.16: Tipos de Generadores de Mallas	35
Figura 1.17: Triangulación de Delaunay	36
Figura 1.18: Acondicionamiento del Triangulo de Delaunay	37
Figura 1.19: Triangulación Delaunay: Envolvente Convexa	37
Figura 1.20: Alisado de Tensiones, antes de la evaluación	38
Figura 1.21: Alisado de Tensiones, resultado final	38
Figura 2.1: Triángulo de Pascal	46
Figura 2.2: Campos de Desplazamiento	46
Figura 2.3: Elemento Triangular de 3 y 6 nudos	47
Figura 2.4: Fuerzas Actuales sobre un elemento triangular	48
Figura 2.5: Coordenadas Naturales sobre un elemento triangular	50
Figura 2.6: Elemento Barra Típico lineal en coordenadas $\chi$ y $\xi$	50
Figura 2.7: Interpolación Lineal del Campo de Desplazamiento	51
Figura 2.8: Función de Interpolación Lineal	51
Figura 2.9: Función de Forma Lineal del campo de Desplazamiento	52
Figura 2.10: Elemento Barra Típico cuadrático en coordenadas $\square$ y $\xi$	52
Figura 2.11: Función de Interpolación Cuadrático	52
Figura 2.12: Función de Forma Cuadrático del campo de Desplazamiento	53

---

Figura 2.13: Coordenadas de Áreas en elemento triangular	54
Figura 2.14: Función de Interpolación Lineal en elemento triangular	55
Figura 2.15: Función de Interpolación Cuadrático en elemento triangular	56
Figura 2.16: Visualización de Resultados referente a los Desplazamiento	63
Figura 2.17: Visualización de Resultados referente a las Tensiones	63
Figura 2.18: Viga con Tensión Sin Alizar	64
Figura 2.19: Viga con Tensión Alisada (Promedio Directo)	64
Figura 2.20: Viga con Tensión Alisada (Método Global)	64
Figura 3.1: Visualización de dos cuerpos antes del contacto	66
Figura 3.2: Fuerza de Reacción del Contacto	67
Figura 3.3: Configuración antes y después del contacto	68
Figura 3.4: Consideraciones referentes al contacto	69
Figura 3.5: Ejemplo sobre el Método de Regulación	70
Figura 3.6: Representación del Cuerpo de Contacto y Cuerpo Objetivo	75
Figura 3.7: Representación de la Inclusión de un cuerpo sobre otro	75
Figura 3.8: Vector Desplazamiento en la Inclusión	77
Figura 3.9: Representación del Algoritmo Mortar Element	78
Figura 3.10: Representación del Algoritmo Splitting Pinballs	79
Figura 3.11: Representación del Algoritmo Splitting Pinballs	79
Figura 3.12: Representación del Operador Nudo-Nudo	81
Figura 3.13: Representación del Operador Nudo-Superficie	81
Figura 3.14: Representación del Operador Superficie- Superfi	82
Figura 4.1: Visualización del modelo	84
Figura 4.2: Estructura discretizada	85
Figura 4.3: Geometría durante los Estados del Proceso	86
Figura 4.4: Esfuerzos Alisados en cada sentido	86
Figura 4.5: Esfuerzos en el sentido X-X en zona de Contacto ( $\sigma_x$ )	87
Figura 4.6: Esfuerzos en el sentido Y-Y en zona de Contacto ( $\sigma_y$ )	87
Figura 4.7: Esfuerzos en el sentido X-Y en zona de Contacto ( $\tau_{xy}$ )	87
Figura 4.8: Escala de Colores para los Esfuerzos	87
Figura 4.9: Valor de Parámetros Locales	88
Figura 4.10: Valores de los Esfuerzos en cada elemento	89
Figura 4.11: Valores de los Esfuerzos Alisados	89
Figura 4.12: Comparativa entre mallas	90
Figura 4.13: Comparativa entre mallas, área localizada	91

---

---

Figura 4.14: Valor de Parámetros Locales	92
Figura 4.15: Esfuerzos en cada elemento triangular	92
Figura 4.16: Esfuerzos Alisados	93
Figura 4.17: Variación de la Rigidez de los Elementos de Contacto	94
Figura 4.18: Desplazamiento en los Nudos	95
Figura 4.19: Esfuerzos Alisados en Zona de Contacto	95
Figura 4.20: Visualización del modelo	96
Figura 4.21: Estructura Discretizada	97
Figura 4.22: Geometría durante los estados del proceso	98
Figura 4.23: Esfuerzos Alisados en cada sentido	98
Figura 4.24: Análisis NoLineal - Esfuerzos en X-X ( $\sigma_x$ )	99
Figura 4.25: Análisis NoLineal - Esfuerzos en Y-Y ( $\sigma_y$ )	100
Figura 4.26: Esfuerzos en cada elemento triangular	101
Figura 4.27: Esfuerzos Alisados	101
Figura 4.28: Valor de Parámetros Locales	102
Figura 4.29: Cambio de Orientación, Esfuerzos	102
Figura 4.30: Comparativa entre mallas	103
Figura 4.31: Comparativa entre mallas, área localizada	103
Figura 4.32: Esfuerzos Alisados en cada sentido	105
Figura 4.33: Valor de Parámetros Locales	105
Figura 4.34: Esfuerzos en cada elemento triangular	106
Figura 4.35: Esfuerzos Alisados	106
Figura 4.36: Variación de la Rigidez de los Elementos de Contacto	107
Figura 4.37: Esfuerzos en Zona de Contacto	108
Figura 4.38: Visualización del modelo	109
Figura 4.39: Estructura Discretizada	110
Figura 4.40: Geometría durante los estados del proceso	110
Figura 4.41: Esfuerzos Alisados en cada sentido	111
Figura 4.42: Esfuerzos en el sentido X-X ( $\sigma_x$ )	111
Figura 4.43: Esfuerzos en el sentido Y-Y ( $\sigma_y$ )	112
Figura 4.44: Esfuerzos en cada elemento triangular	112
Figura 4.45: Esfuerzos Alisados	113
Figura 4.46: Valor de Parámetros Locales	113
Figura 4.47: Comparativa entre mallas	114
Figura 4.48: Comparativa entre mallas, área localizada	114

Figura 4.49: Valor de Parámetros Locales	115
Figura 4.50: Esfuerzos en cada elemento triangular	116
Figura 4.51: Esfuerzos Alisados	116
Figura 4.52: Esfuerzos en Zona de Contacto	117

## INTRODUCCION

Los problemas que contacto son de gran interés para las aplicaciones industriales y como consecuencias para la Ingeniería Mecánica y Civil. Las aplicaciones abarcan desde los procesos de conformación de metales, uniones rígidas y como es en nuestro caso el contacto de edificaciones, otro mayor énfasis se está dando en la Ingeniería Biomecánica y las colisiones de vehículos. Cuando dos o más cuerpos están en contacto, la condición de impenetrabilidad entre los cuerpos debe de ser satisfecha, esto es, un cuerpo no debe penetrar a otro cuerpo, y como resultado, presiones de contacto iguales y opuestas surgen entre ellas a lo largo de la región de contacto.

Los problemas de contacto son predominantemente dinámicos y en menor escala son estáticos, o cuasi-estáticos propiamente hablando donde la variación de acontecimiento con el tiempo ocurre de forma lenta y gradual.

Usualmente la literatura emplea el término "problemas de contacto" para las situaciones generales donde ocurren interacciones entre dos o más cuerpos. Para el caso donde esas interacciones ocurren en intervalos de tiempo más pequeños el término "problemas de contacto-impacto" es más apropiado. Otra situación bastante común e interesante es cuando ocurren en contacto entre partes del mismo sólido, denominados "problemas de auto-contacto".

El análisis de problemas de contacto puede ser realizada a partir de modelos teóricos, experimentales y modelos numéricos. En gran parte las soluciones analíticas encuentran severas limitaciones ya sea por su complejidad física, geométrica o por las complejidades no lineales inherentes al problema y que son difíciles e incluso imposibles de ser resueltas. En los análisis experimentales presenta su mayor desventaja en el tiempo, el número de ensayos y el financiamiento necesario para el montaje de los mismos.

Una alternativa importante está en las técnicas de simulación numérica de los problemas de contacto. Dichas simulaciones presentan una gran ventaja de permitir una serie de ensayos sobre estructuras, pero no descartan el uso de los ensayos dado que dichos resultados experimentales pasarían a ser utilizados como datos de calibración para programas de análisis numéricos.

Entre los métodos más comunes de análisis estructural es el Método de Rigidez, teniendo como Principio Variacional la Teoría del Trabajo Virtual. Otro método

con mayor aceptación es el Método del Elemento Finito obteniendo mejores resultados cuando mayor sea su discretización pero a su vez hace que se desarrolle mejores técnicas de computo como por ejemplo, para el caso de las restricciones, son utilizados el método de Penalidad o el método de los Multiplicadores de Lagrange, teniendo como ventaja el de Penalidad sobre los Multiplicadores debido a que el número de incógnitas es menor pero a su vez es muy sensible al valor del parámetro de penalidad. En vista de todos estos procesos matemáticos y computacionales se está desarrollando otra técnica de análisis llamada Método de Contorno, tal como su nombre lo dice dicho método ya no necesita discretizar todo el elemento sino tan solo las partes que lo bordean. En términos generales se han ido creando nuevos algoritmos de solución y sistemas computacionales de alto desempeño para tratar de dar resultados más reales.

## CAPITULO I ASPECTOS TEORICOS BASICOS

En este capítulo revisaremos los principales tópicos básicos e importantes para el desarrollo de la tesis como son: Mecánica de Medios Continuos, Los Generadores de Mallas, Alisados de Tensiones y Desarrollo de Sistemas de Ecuaciones.

### 1.1.- MECÁNICA DE MEDIOS CONTINUOS.

La Mecánica de Medios Continuos es parte de la física que propone un modelo matemático que permita investigar las propiedades de los sólidos deformables, sólidos rígidos y fluidos sin tener una discontinuidad entre las partículas y que la descripción matemática y sus propiedades se puedan realizar mediante funciones continuas.

A pesar que la materia no es continua a nivel microscópica, podríamos suponer para su estudio que lo es, siendo de esta manera una aproximación muy buena para cuerpos cuyas dimensiones son mucho mayores que los átomos.

Para el desarrollo de problemas en este campo, es necesario determinar los campos de tensiones y deformaciones, siendo las ecuaciones necesarias para su estudio las ecuaciones de equilibrio (relación de tensiones internas del sólido con cargas aplicadas), las ecuaciones constitutivas (relaciona las tensiones y deformaciones, y otras magnitudes) y las ecuaciones de compatibilidad (desplazamientos en función de deformaciones).

Según sea la ecuación constitutiva desarrollada, se pueden clasificar tres tipos de comportamiento: Comportamiento Elástico, Comportamiento Plástico y Comportamiento Viscoso.

### 1.1.1.- ANÁLISIS DE DEFORMACIÓN

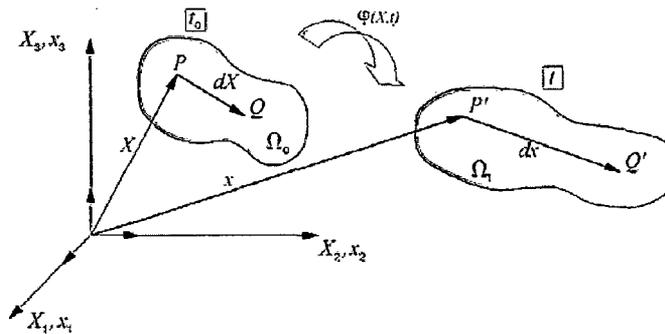


Figura 1.1: Estado de Deformación, Tensor Gradiente de Deformación

Sean las ecuaciones del movimiento de un punto.

$\mathbf{x} = \varphi(\mathbf{X}, t) = \mathbf{x}(\mathbf{X}, t)$ $x_i = \varphi_i(X_1, X_2, X_3, t) = x_i(X_1, X_2, X_3, t)$ $i \in \{1, 2, 3\}$	$\mathbf{X} = \varphi(\mathbf{x}, t) = \mathbf{X}(\mathbf{x}, t)$ $X_i = \varphi_i(x_1, x_2, x_3, t) = X_i(x_1, x_2, x_3, t)$ $i \in \{1, 2, 3\}$
<p>Coordenadas Espaciales en función de materiales. (Formulación Lagrangiana)</p>	<p>Coordenadas Materiales en función de espaciales. (Formulación Euleriana)</p>

Si derivamos la ecuación del movimiento en coordenadas espaciales con respecto a las coordenadas materiales obtenemos:

$dx_i = \frac{\partial x_i}{\partial X_j} dX_j$ $d\mathbf{x} = \mathbf{F} \cdot d\mathbf{X}$ $i, j \in \{1, 2, 3\}$	$\mathbf{F} = \mathbf{x} \otimes \bar{\mathbf{V}}$ $F_{ij} = \frac{\partial x_i}{\partial X_j}$
<p>Ecuación Fundamental de la Deformación</p>	<p>Tensor Gradiente material de la Deformación</p>

Ahora, realizamos la derivación con respecto a las coordenadas espaciales, obtenemos:

$dX_i = \frac{\partial X_i}{\partial x_j} dx_j$ $d\mathbf{X} = \mathbf{F}^{-1} \cdot d\mathbf{x}$ $i, j \in \{1, 2, 3\}$	$\mathbf{F}^{-1} = \mathbf{X} \otimes \bar{\mathbf{V}}$ $F_{ij}^{-1} = \frac{\partial X_i}{\partial x_j}$
<p>Ecuación Fundamental de la Deformación</p>	<p>Tensor Gradiente espacial de la Deformación</p>

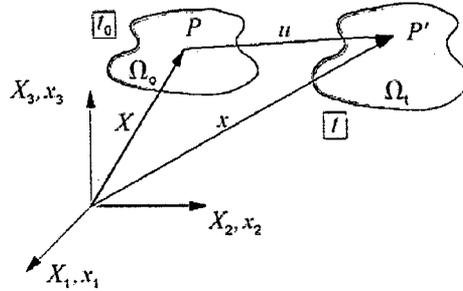


Figura 1.2: Estado de Deformación, Vector Desplazamiento

Sean los vectores de Desplazamientos en coordenadas respectivas:

$U(X,t) = x(X,t) - X$ $U_i(X,t) = x_i(X,t) - X_i$ $i \in \{1,2,3\}$	$u(x,t) = x - X(x,t)$ $u_i(x,t) = x_i - X_i(x,t)$ $i \in \{1,2,3\}$
Vector Desplazamiento en función de coordenadas materiales	Vector Desplazamiento en función de coordenadas espaciales

Si derivamos el vector desplazamiento con respecto a las coordenadas materiales y espaciales respectivamente obtenemos:

$\frac{\partial U_i}{\partial X_j} = \frac{\partial x_i}{\partial X_j} - \frac{\partial X_i}{\partial X_j} = F_{ij} - \delta_{ij} = J_{ij}$ $\delta_{ij} = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$	$\frac{\partial u_i}{\partial x_j} = \frac{\partial x_i}{\partial x_j} - \frac{\partial X_i}{\partial x_j} = \delta_{ij} - F_{ij}^{-1} = J_{ij}$ $\delta_{ij} = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$
Tensor Gradiente material de los Desplazamientos	Tensor Gradiente espacial de los Desplazamientos

Considerando ahora una partícula  $P$  del medio continuo y otra partícula  $Q$  de su entorno diferencial en ambas configuraciones (material y espacial) y separadas por el segmento  $dX$  (de longitud  $ds = \sqrt{dX \cdot dX}$ ) y  $dx$  (de longitud  $ds = \sqrt{dx \cdot dx}$ ) respectivamente.

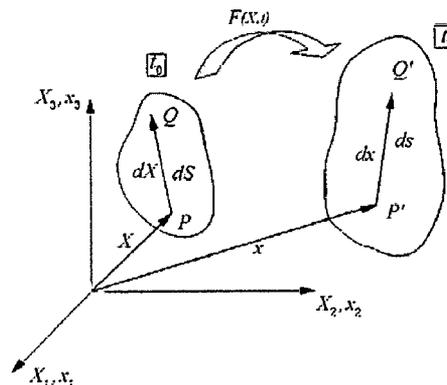


Figura 1.3: Estado de Deformación, Longitud Diferencial

Desarrollando la expresión de longitud:

$$(ds)^2 = dx \cdot dx = [dx]^T \cdot [dx] = [F \cdot dX]^T \cdot [F \cdot dX] = dX \cdot F^T \cdot F \cdot dX$$

$$(ds)^2 = dx_k \cdot dx_k = F_{ki} \cdot dX_i \cdot F_{kj} \cdot dX_j = dX_i \cdot F_{ki} \cdot F_{kj} \cdot dX_j = dX_i \cdot F_{ik}^T \cdot F_{kj} \cdot dX_j$$

$$(dS)^2 = dX \cdot dX = [dX]^T \cdot [dX] = [F^{-1} \cdot dx]^T \cdot [F^{-1} \cdot dx] = dx \cdot F^{-T} \cdot F^{-1} \cdot dx$$

$$(dS)^2 = dX_k \cdot dX_k = F_{ki}^{-1} \cdot dx_i \cdot F_{kj}^{-1} \cdot dx_j = dx_i \cdot F_{ki}^{-1} \cdot F_{kj}^{-1} \cdot dx_j = dx_i \cdot F_{ik}^{-T} \cdot F_{kj}^{-1} \cdot dx_j$$

$$\begin{aligned} (ds)^2 - (dS)^2 &= dX \cdot F^T \cdot F \cdot dX - dX \cdot dX = dX \cdot F^T \cdot F \cdot dX - dX \cdot 1 \cdot dX = \\ &= dX \cdot (F^T \cdot F - 1) \cdot dX = 2dX \cdot E \cdot dX \end{aligned}$$

Siendo:

$E(X, t) = \frac{1}{2} (F^T \cdot F - 1)$ $E_{ij}(X, t) = \frac{1}{2} (F_{ki} F_{kj} - \delta_{ij})$ $i, j \in \{1, 2, 3\}$	$e(x, t) = \frac{1}{2} (1 - F^{-T} \cdot F^{-1})$ $e_{ij}(x, t) = \frac{1}{2} (\delta_{ij} - F_{ki}^{-1} F_{kj}^{-1})$ $i, j \in \{1, 2, 3\}$
El Tensor material de Deformación (Green-Lagrange)	El Tensor espacial de Deformación (Green-Lagrange)

Reemplazando las ecuaciones se obtiene el Tensor Deformación en función de la Gradiente de Desplazamientos.

$E = \frac{1}{2} [(1 + J^T) \cdot (1 + J) - 1] = \frac{1}{2} [J + J^T + J^T \cdot J]$ $E_{ij} = \frac{1}{2} \left[ \frac{\partial U_i}{\partial X_j} + \frac{\partial U_j}{\partial X_i} + \frac{\partial U_k}{\partial X_i} \frac{\partial U_k}{\partial X_j} \right]$ $i, j \in \{1, 2, 3\}$	$e = \frac{1}{2} [1 - (1 - j^T) \cdot (1 - j)] = \frac{1}{2} [j + j^T - j^T \cdot j]$ $e_{ij} = \frac{1}{2} \left[ \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{\partial u_k}{\partial x_i} \frac{\partial u_k}{\partial x_j} \right]$ $i, j \in \{1, 2, 3\}$
El Tensor material de Deformación	El Tensor espacial de Deformación

Y bajo la Teoría de Deformación Infinitesimal o Teoría de Pequeñas Deformaciones, las ecuaciones quedan:

$\left[ \frac{\partial U_i}{\partial X_j} \right] \ll 1$ $E_{ij} = \frac{1}{2} \left[ \frac{\partial U_i}{\partial X_j} + \frac{\partial U_j}{\partial X_i} \right]$ $i, j \in \{1, 2, 3\}$	$\left[ \frac{\partial u_i}{\partial x_j} \right] \ll 1$ $e_{ij} = \frac{1}{2} \left[ \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right]$ $i, j \in \{1, 2, 3\}$
---	---

Bajo esta hipótesis se redefine el Tensor de Deformación Infinitesimal a la siguiente expresión:

$$E(x, t) = e(x, t) = \varepsilon(x, t)$$

### 1.1.2.- ANÁLISIS DE TENSIONES

Sean las fuerzas másicas (fuerzas que actúan sobre las partículas del interior del medio continuo) y las fuerzas de superficie (fuerzas que actúan sobre el contorno del volumen del material) las que consideraremos en este estudio.

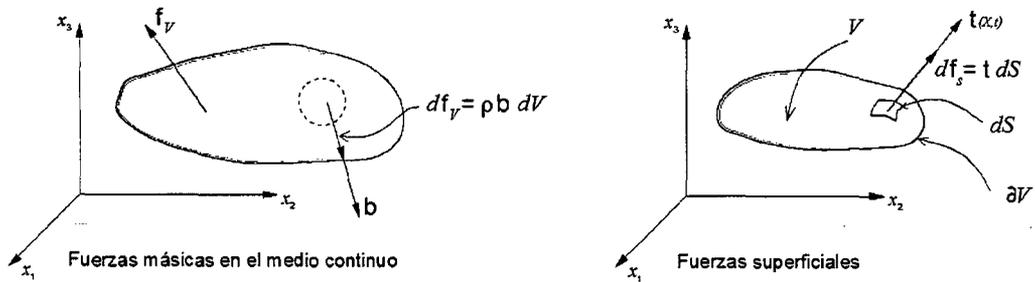


Figura 1.4: Fuerzas Másicas y Superficiales

Según los postulados de Cauchy y la 2ª Ley de Newton para un medio continuo se describe la siguiente ecuación:

$$R = \sum_i f_i = \sum_i m_i \cdot a_i$$

$$R = \int_{V_i} \rho \cdot b \cdot dV + \int_{\partial V_i} t \cdot dS = \int_M a \cdot dm = \int_{V_i} \rho \cdot a \cdot dV$$

Consideremos ahora, el volumen de un tetraedro situado alrededor de una partícula arbitraria P del interior del medio continuo.

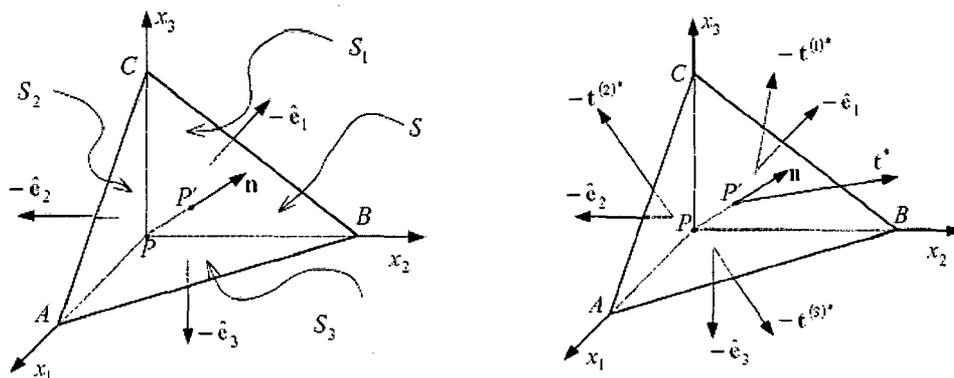


Figura 1.5: Tetraedro elemental alrededor de un punto P

De la ecuación anterior, desarrollando:

$$\int_V \rho \cdot b \cdot dV + \int_S t \cdot dS + \int_{S_1} (-t^{(1)}) \cdot dS + \int_{S_2} (-t^{(2)}) \cdot dS + \int_{S_3} (-t^{(3)}) \cdot dS = \int_V \rho \cdot a \cdot dV$$

$$\rho \cdot b \cdot V + t \cdot S - t^{(1)} \cdot S_1 - t^{(2)} \cdot S_2 - t^{(3)} \cdot S_3 = \rho \cdot a \cdot V ; V \cong 0$$

$$t(P, n) - t^{(1)} n_1 - t^{(2)} n_2 - t^{(3)} n_3 = 0$$

$$t(P, n) - t^{(i)} n_i = 0$$

Teniendo presente que el par de vectores de tracciones  $t^{(i)}$  y  $n_i$  define el estado de tensión en un punto, el vector tensión  $t^{(i)}$  pueden escribirse en función de sus componentes cartesianas:

$t^{(1)} = t_1^{(1)} \hat{e}_1 + t_2^{(1)} \hat{e}_2 + t_3^{(1)} \hat{e}_3 = t_j^{(1)} \hat{e}_j$ $t^{(2)} = t_1^{(2)} \hat{e}_1 + t_2^{(2)} \hat{e}_2 + t_3^{(2)} \hat{e}_3 = t_j^{(2)} \hat{e}_j$ $t^{(3)} = t_1^{(3)} \hat{e}_1 + t_2^{(3)} \hat{e}_2 + t_3^{(3)} \hat{e}_3 = t_j^{(3)} \hat{e}_j$ $t_j^{(i)} \equiv \sigma_{ij}$	$t^{(i)}(P) = \sigma_{ij} \hat{e}_j$ $\sigma_{ij}(P) = t_j^{(i)}(P)$ $i, j \in \{1, 2, 3\}$
Para cada componente	Para caso general

Regresando a la ecuación anterior y reemplazando:

$$t(P, n) = n_i \cdot t^{(i)} \Rightarrow t_j(P, n) = n_i \cdot t_j^{(i)}(P) = n_i \cdot \sigma_{ij}(P)$$

$$t(P, n) = n \cdot \sigma(P)$$

Obteniéndose de esta manera el Tensor de Tensiones de Cauchy:

$$\sigma = \sigma_{ij} \hat{e}_i \times \hat{e}_j$$

Siendo su representación gráfica y sus componentes la siguiente:

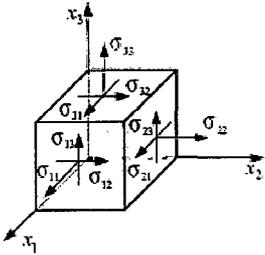
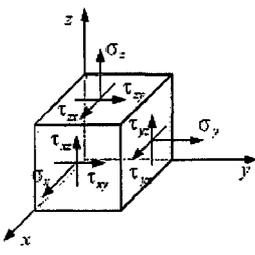
 $\sigma \equiv \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix}$	 $\sigma \equiv \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix}$
---	--

Figura 1.6: Representación Gráfica y Matricial del Estado Tensional

El tensor de tensiones, las fuerzas másicas y las aceleraciones tienen una relación llamada Ecuación de Cauchy

$$\nabla \cdot \sigma + \rho b = \rho a \Rightarrow \frac{\partial \sigma_{ij}}{\partial x_i} + \rho b_j = \rho a_j$$

Si el sistema está en equilibrio, la aceleración es nula ( $a = 0$ ), llamando de esta manera a la expresión anterior:

$$\begin{cases} \frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + \rho \cdot b_x = 0 \\ \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} + \frac{\partial \tau_{zy}}{\partial z} + \rho \cdot b_y = 0 \\ \frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \sigma_z}{\partial z} + \rho \cdot b_z = 0 \end{cases}$$

Ecuación de Equilibrio Interno

### 1.1.3.- ELASTICIDAD

La Teoría de la Elasticidad supone que los desplazamientos y los gradientes de desplazamientos son suficientemente pequeños de manera tal que no es necesaria ninguna distinción entre los descriptores lagrangiana y euleriana. Dicho de otra manera, La Teoría de la Elasticidad es el estudio del comportamiento de aquellas sustancias o cuerpos que poseen la propiedad de recuperar su tamaño y forma cuando la fuerza que produce deformaciones es retirada.

Una ley básica e importante para el estudio de este campo es la Ley de Hooke, que indica la existencia de una proporcionalidad entre la Tensión  $\sigma$  y la Deformación  $\varepsilon$ , a través de una constante de proporcionalidad llamada Módulo de Elasticidad.

$\sigma = E \cdot \varepsilon$	$\sigma_{(x,t)} = \mathbb{C} : \varepsilon_{(x,t)}$ $\sigma_{ij} = \mathbb{C}_{ijkl} \cdot \varepsilon_{kl}$ $\mathbb{C} = \text{Tensor de Constante Elastica}$
Para problemas unidimensionales	Para problemas multidimensionales (Ley Generalizada de Hooke)

Y bajo ciertas consideraciones como: la energía de deformación elástica, simetría de las propiedades elásticas, homogeneidad, Isotropía y elasticidad, el Tensor de Constantes Elásticas se describe de la siguiente manera:

$$C_{ijkl} = \lambda \delta_{ij} \delta_{kl} + \mu [\delta_{ik} \delta_{jl} + \delta_{il} \delta_{jk}]$$

$$i, j, k, l \in \{1, 2, 3\}$$

Donde  $\lambda$  y  $\mu$  son las Constantes de Lamé y que caracterizan el comportamiento elástico del material y son obtenidas mediante proceso experimental.

Reemplazando en la ley generalizada de Hooke, obtenemos:

$$\sigma_{ij} = C_{ijkl} \cdot \varepsilon_{kl} = \lambda \delta_{ij} \underbrace{\delta_{kl} \varepsilon_{kl}}_{\varepsilon_{ll}} + 2\mu \left[ \underbrace{\frac{1}{2} \delta_{ik} \delta_{jl} \varepsilon_{kl}}_{\varepsilon_{ij}} + \underbrace{\frac{1}{2} \delta_{il} \delta_{jk} \varepsilon_{kl}}_{\varepsilon_{ji} = \varepsilon_{ij}} \right]$$

$$\sigma_{ij} = \lambda \delta_{ij} \varepsilon_{ll} + 2\mu \varepsilon_{ij}$$

Siendo esta la Ecuación Constitutiva para un material elástico lineal isotrópico.

Realizando la inversión de la ecuación anterior, tenemos:

$$\varepsilon_{ij} = -\frac{\lambda}{2\mu(3\lambda + 2\mu)} \delta_{ij} \sigma_{kk} + \frac{1}{2\mu} \sigma_{ij}$$

Definiéndose de la ecuación anterior, las nuevas propiedades elásticas:

$E = \frac{\mu(3\lambda + 2\mu)}{\lambda + \mu}$ <p>Módulo de Young</p>	$\nu = \frac{\lambda}{2(\lambda + \mu)}$ <p>Coefficiente de Poisson</p>
---	---

$\lambda = \frac{\nu E}{(1 + \nu)(1 - 2\nu)}$	$\mu = \frac{E}{2(1 + \nu)} = G$
---	----------------------------------

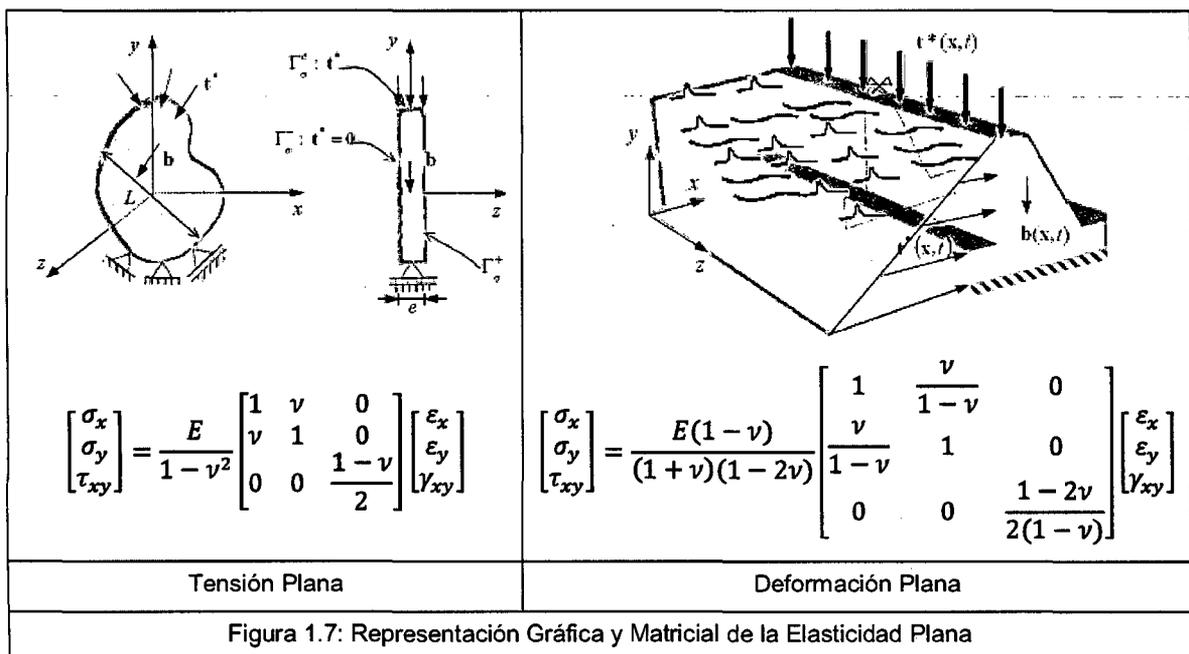
La Ley de Hooke se puede reescribir en función a estas constantes elásticas, como:

$$\sigma_{ij} = \frac{E}{1 + \nu} \left( \varepsilon_{ij} + \frac{\nu}{1 - 2\nu} \delta_{ij} \varepsilon_{kk} \right)$$

Existen unos casos especiales, en las que el análisis del elemento se realiza a nivel bidimensional llamado Elasticidad Plana. En este caso, los problemas se modelan como tensión plana y deformación plana.

En los problemas de *Tensión Plana*, la geometría del cuerpo es una lámina con una dimensión muy delgada en comparación de las otras dos, las cargas se aplican uniformemente sobre el espesor de la lámina y actúa sobre el plano de la misma.

En los problemas de *Deformación Plana*, la geometría del cuerpo es la de un cilindro prismático con una dimensión más grande en comparación a las otras dos, las cargas están uniformemente distribuidas con respecto a la dimensión mayor y actuando perpendicularmente a ella.



## 1.2.- MECÁNICA DEL CONTACTO

El objeto de la Mecánica de Contacto llega hacer conocido a la comunidad científica después de que Heinrich Hertz publicase su clásico artículo "The Contact of Elastic Solid" en 1882. Aunque su teoría fuera restringida a superficies de contacto friccionantes y sólidos no-conformes lineales elásticos, sus ideas y la importancia tecnológica despertó el interés considerable. Esto dio lugar al desarrollo rápido de más teoría de contacto general y condujo al establecimiento del concepto de mecánica de contacto.

Un análisis apropiado para los problemas de contacto podría ser más factible si el modelo pudiera ser simplificado; por ejemplo, si las fuerzas de fricción entre los cuerpos en la zona de contacto son muy pequeñas o el tamaño de la zona de contacto es pequeño en comparación con el radio de curvatura de los cuerpos en contacto suponiendo de esta manera que uno de los cuerpos puede ser sustituido por un medio elástico semi-infinito.

Para la mayor parte de problemas de contacto, el área de contacto y la distribución de los esfuerzos sobre el área son desconocidas antes del análisis. Estos esfuerzos son causados por la presión de un sólido sobre el otro sobre las áreas limitadas de contacto.

### 1.2.1.- CLASIFICACIÓN DE CONTACTO

Podría decirse que su clasificación está en base a las consideraciones hechas para su estudio. Se presentan a continuación las más importantes:

- **Contacto No Friccionante**

Es una idealización del modelo de contacto con limitaciones prácticas, es decir los cuerpos en contacto pueden deslizarse a lo largo de la dirección tangencial o área de contacto sin ningún tipo de resistencia, quedando de esta manera solo el estado de compresión en el sistema de equilibrio, iniciando de esta manera al proceso de deformación en algunas partes del cuerpo del área de contacto sin incurrir en ninguna incompatibilidad geométrica (impenetrabilidad).

- **Contacto Friccionante**

Siendo la fricción un fenómeno físico, está netamente enlazado a los problemas de contacto, aumentando el grado de dificultad para desarrollar este problema. Los efectos de fricción son caracterizados por el comportamiento dentro de la región de contacto; por ejemplo, el movimiento de desplazamientos en la dirección tangencial de un punto de contacto será restringido por la fuerza de fricción, lo que a su vez depende del componente de la fuerza normal que fue aplicado. La relación entre los componentes de las fuerzas normales y tangenciales imponen el comportamiento no-lineal entre el movimiento de desplazamiento en la superficie de contacto y la carga externa.

- **Contacto Conforme**

Un contacto se dice que es conforme si las superficies de ambos cuerpos coinciden exactamente en un estado no-cargado o están estrechamente unidos sin producir deformación inicial. Una de las principales características para este estado de contacto es que el tamaño del área de contacto es independiente de la carga aplicada, por esta razón la carga histórica aplicada a un cuerpo no es importante para el desarrollo del mismo.

- **Contacto No-Conforme**

Cuando dos cuerpos sólidos están inicialmente en un estado no-cargados y se ponen con contacto entre ellos, solo se pueden tocar en un punto o línea de contacto, como es el ejemplo de dos esferas que tienen un punto de contacto o el caso de dos cilindros que tiene una línea de contacto. En estas situaciones, donde el potencial del área de contacto de dos cuerpos que tienen diferentes perfiles de contorno pueden ser denominados contactos no-conformes. La característica importante en estos problemas de contacto es el tamaño del área de contacto, ya que cambiará a medida que los cuerpos sean cargados, dependiendo de factores importantes como los perfiles de contornos iniciales, las propiedades del material y la dirección de la carga aplicada.

- **Contacto Hertziano**

La forma de la distribución de las presiones de contacto, las áreas de contacto y las deformaciones entre dos cuerpos circulares lisos en contacto fue estudiada y presentada por Hertz. Su enfoque analítico del problema ha superado la prueba del tiempo y sigue siendo uno de los más importantes aportes para la solución de algunos problemas de ingeniería. Los detalles y consideraciones que propuso se verán con más detalle en los ítem's siguientes.

- **Contacto No-Hertziano**

La teoría de Hertz es un modelo idealizado que cubre solo un número pequeño de problemas de contacto. Los problemas prácticos inevitablemente violan algunas hipótesis de Hertz, como tales se les conocen como "Contacto No-Hertziano".

Como una aproximación, hay casos en los que la teoría de Hertz se puede utilizar para resolver problemas No-Hertziano, por ejemplo, los problemas de contacto elástico no-conforme con muy bajo coeficiente de fricción.



Figura 1.8: Tipos de Contacto

### 1.2.2.- CARGAS DE CONTACTO

En esta parte del contexto, determinaremos los esfuerzos que pueden ser ejercidas debido a las cargas externas de algunos casos.

- **Fuerzas transmitidas a un Punto de Contacto**

La fuerza resultante transmitida desde una superficie hacia otra a través de un punto de contacto se resuelve en una fuerza normal  $P$  actuando de la normal común entre ellos y que por lo general es en un estado de compresión. Y en una fuerza tangencial  $Q$  en el plano tangente bajo los efectos de la fricción, la magnitud de  $Q$  debe ser menor o igual que el límite de la fuerza de fricción.

$Q \leq \mu \cdot P$ $\mu = \text{Coeficiente de Fricción}$	$Q_x = -\frac{\Delta v_x}{ \Delta v } \cdot \mu P$ $Q_y = -\frac{\Delta v_y}{ \Delta v } \cdot \mu P$
--	---

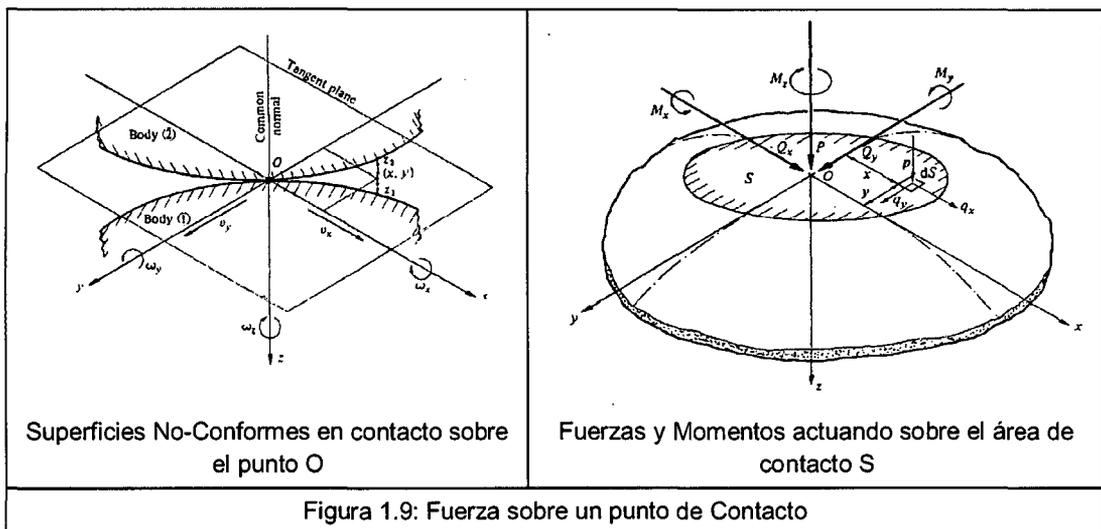


Figura 1.9: Fuerza sobre un punto de Contacto

Los momentos  $M_x$  y  $M_y$  son definidos como momentos de rodamiento *rolling moments* las que proporcionan la resistencia al movimiento rotacional y en la mayoría de problemas prácticos son significativamente pequeños los cuales son ignorados.

Las fuerzas que actúan sobre dicha región de contacto son expresadas a continuación:

$$P = \int_S p \cdot dS \quad ; \quad Q_x = \int_S q_x \cdot dS \quad ; \quad Q_y = \int_S q_y \cdot dS$$

$$M_x = \int_S p \cdot y \cdot dS \quad ; \quad M_y = - \int_S p \cdot x \cdot dS \quad ; \quad M_z = \int_S (q_y \cdot x - q_x \cdot y) \cdot dS$$

$$\sigma_r = \frac{P}{2\pi} (1-2\nu) \left[ \frac{1}{r^2} - \frac{z}{r^2(r^2+z^2)^{1/2}} \right] - \frac{3r^2z}{(r^2+z^2)^{5/2}}$$

$$\sigma_\theta = \frac{P}{2\pi} (1-2\nu) \left[ -\frac{1}{r^2} + \frac{z}{r^2(r^2+z^2)^{1/2}} + \frac{z}{(r^2+z^2)^{3/2}} \right]$$

$$\sigma_z = -\frac{3P}{2\pi} \frac{z^3}{(r^2+z^2)^{5/2}}$$

$$\tau_{rz} = -\frac{3P}{2\pi} \frac{rz^2}{(r^2+z^2)^{5/2}}$$

- **Línea carga en un semi-plano elástico**

Para el caso de un análisis bidimensional, la fuerza concentrada distribuida uniformemente a lo largo de una línea representaría el caso de un contacto entre cuerpos cilíndricos.

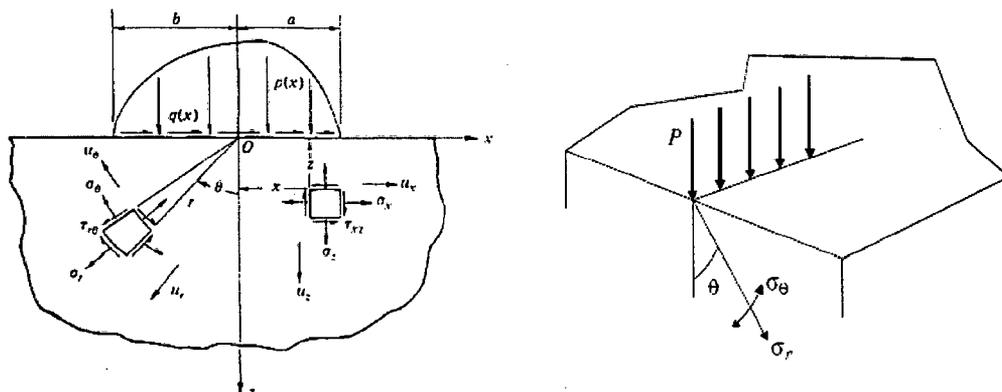


Figura 1.10: Línea de Fuerza sobre un plano

La distribución de esfuerzos dentro de un cuerpo producido por una carga distribuida a lo largo del eje  $y$  cuya intensidad por unidad de longitud es el valor de  $P$  es radialmente direccionado hacia el punto de contacto. Lo que sería conveniente trabajar en un sistema de coordenadas polares de la manera siguiente:

$$\begin{aligned}\sigma_r &= \frac{1}{r} \cdot \frac{\partial \phi}{\partial r} + \frac{1}{r^2} \cdot \frac{\partial^2 \phi}{\partial \theta^2} & \phi(r, \theta) &= A \cdot r \cdot \theta \cdot \sin \theta \\ \sigma_\theta &= \frac{\partial^2 \phi}{\partial r^2} & \sigma_r &= 2A \cdot \frac{\cos \theta}{r} \\ \tau_{r\theta} &= -\frac{\partial}{\partial r} \left( \frac{1}{r} \cdot \frac{\partial \phi}{\partial \theta} \right) & \sigma_\theta = \tau_{r\theta} &= 0\end{aligned}$$

En la superficie  $\theta = \pm\pi/2$ , el esfuerzo  $\sigma_\theta = 0$  excepto en el origen mismo. A medida que  $r \rightarrow \infty$  los esfuerzos tienden a cero, satisfaciendo de esta manera las condiciones de contorno. La constante ( $A$ ) puede ser desarrollada de la ecuación siguiente:

$$-P = \int_{-\pi/2}^{+\pi/2} \sigma_r \cdot \cos \theta \cdot r \cdot d\theta = \int_0^{\pi/2} 2A \cdot \cos^2 \theta \cdot d\theta = A \cdot \pi$$

Transformando los esfuerzos en coordenadas cartesianas:

$$\begin{aligned}\sigma_x &= \sigma_r \cdot \sin^2 \theta = -\frac{2P}{\pi} \cdot \frac{x^2 z}{(x^2 + z^2)^2} & \sigma_z &= \sigma_r \cdot \cos^2 \theta = -\frac{2P}{\pi} \cdot \frac{z^3}{(x^2 + z^2)^2} \\ \tau_{zx} &= \sigma_r \cdot \sin \theta \cdot \cos \theta = -\frac{2P}{\pi} \cdot \frac{xz^2}{(x^2 + z^2)^2}\end{aligned}$$

De la ley de Hooke, se puede hallar los desplazamientos:

$$\begin{aligned}u_r &= \frac{(1-\nu^2)}{\pi E} \cdot 2P \cdot \cos \theta \cdot \ln r - \frac{(1-2\nu)(1+\nu)}{\pi E} \cdot P \cdot \theta \cdot \sin \theta + C_1 \cdot \sin \theta + C_2 \cdot \cos \theta \\ u_\theta &= \frac{(1-\nu^2)}{\pi E} \cdot 2P \cdot \sin \theta \cdot \ln r + \frac{\nu(1+\nu)}{\pi E} \cdot 2P \cdot \sin \theta - \frac{(1-2\nu)(1+\nu)}{\pi E} \cdot 2P \cdot \theta \cdot \cos \theta + \\ &+ \frac{(1-2\nu)(1+\nu)}{\pi E} \cdot P \cdot \sin \theta + C_1 \cdot \cos \theta - C_2 \cdot \sin \theta + C_3 \cdot r\end{aligned}$$

Para  $\theta = \pm\pi/2$ :

$$[\bar{u}_r]_{\theta=\pi/2} = [\bar{u}_r]_{\theta=-\pi/2} = -\frac{(1-2\nu)(1+\nu)P}{2E}$$

$$[\bar{u}_\theta]_{\theta=\pi/2} = -[\bar{u}_\theta]_{\theta=-\pi/2} = -\frac{(1-\nu^2)}{\pi E} \cdot 2P \cdot \ln(r_0/r)$$

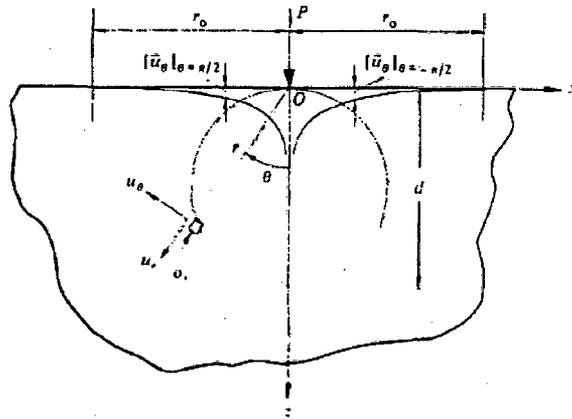


Figura 1.11: Línea de Carga perpendicular al plano xy

Para fuerzas concentradas  $Q$  por unidad de longitud a lo largo del eje  $y$ , produce esfuerzos radiales similares a los que actuarían en el plano normal pero rotado  $90^\circ$

$$\sigma_r = -2 \cdot \frac{Q}{\pi} \cdot \frac{\cos \theta}{r}$$

$$\sigma_\theta = \tau_{r\theta} = 0$$

$$\sigma_x = -\frac{2Q}{\pi} \cdot \frac{x^3}{(x^2 + z^2)^2}$$

$$\sigma_z = -\frac{2Q}{\pi} \cdot \frac{xz^2}{(x^2 + z^2)^2}$$

$$\tau_{zx} = -\frac{2Q}{\pi} \cdot \frac{x^2z}{(x^2 + z^2)^2}$$

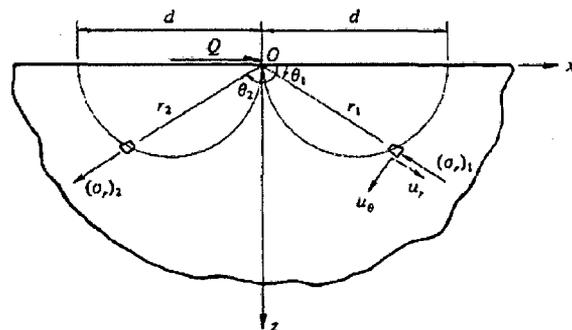


Figura 1.12: Línea de Carga perpendicular al plano yz

Para un caso más general, tomamos una franja de línea con un ancho  $(a + b)$  como se muestra en la figura  $(-b \leq x \leq a)$  y bajo las mismas condiciones descritas anteriormente, se llega a tener las siguientes expresiones:

$$\sigma_x = -\frac{2z}{\pi} \int_{-b}^a \frac{p(s) \cdot (x-s)^2}{\{(x-s)^2 + z^2\}^2} \cdot ds - \frac{2}{\pi} \int_{-b}^a \frac{q(s) \cdot (x-s)^3}{\{(x-s)^2 + z^2\}^2} \cdot ds$$

$$\sigma_z = -\frac{2z^3}{\pi} \int_{-b}^a \frac{p(s)}{\{(x-s)^2 + z^2\}^2} \cdot ds - \frac{2z^2}{\pi} \int_{-b}^a \frac{q(s) \cdot (x-s)}{\{(x-s)^2 + z^2\}^2} \cdot ds$$

$$\tau_{xz} = -\frac{2z^2}{\pi} \int_{-b}^a \frac{p(s) \cdot (x-s)}{\{(x-s)^2 + z^2\}^2} \cdot ds - \frac{2z}{\pi} \int_{-b}^a \frac{q(s) \cdot (x-s)^2}{\{(x-s)^2 + z^2\}^2} \cdot ds$$

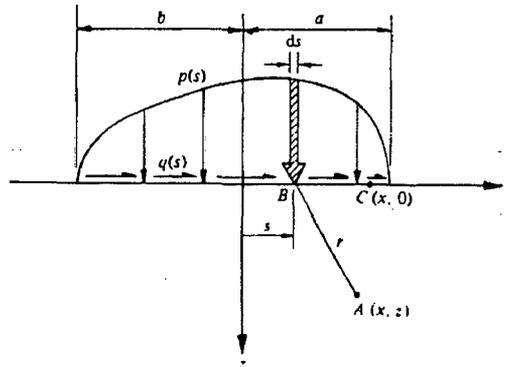


Figura 1.13: Franja de Carga perpendicular al plano xy

- **Distribución Uniforme de Tensión**

De la ecuación general anterior, tomamos una franja con un ancho de  $2a$   $(-a \leq x \leq a)$ , en las que se aplicará una carga uniforme distribuida.

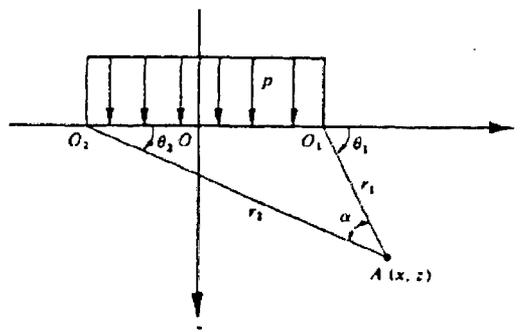


Figura 1.14: Distribución de Carga Uniforme

$\sigma_x = -\frac{P}{2\pi} \{2(\theta_1 - \theta_2) - (\sin 2\theta_1 - \sin 2\theta_2)\}$ $\sigma_z = -\frac{P}{2\pi} \{2(\theta_1 - \theta_2) + (\sin 2\theta_1 - \sin 2\theta_2)\}$ $\tau_{xz} = -\frac{P}{2\pi} (\cos 2\theta_1 - \cos 2\theta_2)$ $\tan \theta_{1,2} = z / (x \mp a)$	$\sigma_x = \frac{q}{2\pi} \{(4 \ln(r_1 / r_2) - (\cos 2\theta_1 - \cos 2\theta_2))\}$ $\sigma_z = \frac{q}{2\pi} (\cos 2\theta_1 - \cos 2\theta_2)$ $\tau_{xz} = -\frac{q}{2\pi} \{2(\theta_1 - \theta_2) + (\sin 2\theta_1 - \sin 2\theta_2)\}$ $r_{1,2} = \{(x \mp a)^2 + z^2\}^{1/2}$
Presión Normal	Tracción Tangencial

• **Otros casos**

Existen otros casos en las que pueden ser aplicadas las cargas como son: Distribución Triangular de Tensión, Tensión paralelo al eje  $y$ , Carga puntual sobre un plano, Presión aplicada sobre una región, Tensiones axi-simétricas, Cargas torsionantes, etc.

**1.2.3.- MODELO DE CONTACTO DE HERTZ**

Hertz desarrolló las primeras soluciones de contacto analíticas para problemas de contacto elásticos sólidos bajo ciertas suposiciones:

- El material de los cuerpos es homogéneo.
- Las cargas son estáticas.
- Se mantiene la validez de la ley de Hooke.
- La tensión de contacto se desvanece en el punto opuesto del cuerpo.
- El radio de la superficie de contacto es muy pequeño en comparación con el radio de curvatura de los cuerpos.
- La superficie de contacto entre los cuerpos es lo suficientemente liza para que no existan tensiones tangenciales (rozamiento).

Realizó un estudio del contacto entre una esfera rígida y una superficie plana, encontrando relación entre el radio de la circunferencia de contacto " $a$ ", la carga " $P$ ", el radio " $R$ " de la esfera y las propiedades de los materiales.

$a^3 = \frac{3}{4} \cdot \frac{P \cdot R}{E^*}$	$\frac{1}{E^*} = \frac{(1 - \nu^2)}{E} + \frac{(1 - \nu'^2)}{E'}$
---	---

Hertz también encontró la resistencia a la tensión máxima en el modelo y esta dado por:

$\sigma_{max} = (1 - 2\nu) \cdot \frac{P}{2\pi a^2}$	$\sigma_{max} = \left( \frac{1 - 2\nu}{2\pi} \right) \cdot \left( \frac{4E^*}{3} \right)^{2/3} \cdot P^{1/3} \cdot R^{-2/3}$
--	--

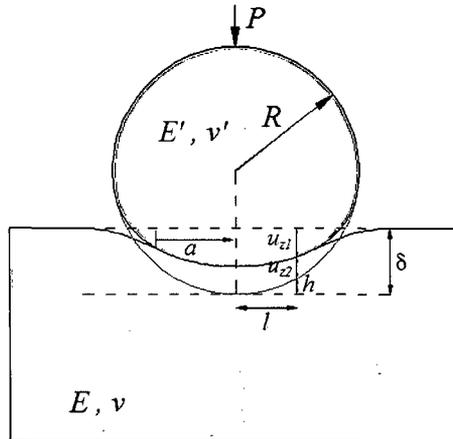


Figura 1.15: Modelo de Hertz

Posteriormente, de estos estudios elaborados por Hertz se obtuvieron algunas formulas para diversos casos. Para el caso de un cilindro apoyado sobre una superficie plana se tienen las siguientes formulas:

$\sigma_{(x)} = k \cdot \sqrt{(b^2 - x^2)}$ $\sigma_{max} = k \cdot b$	$b^2 = \frac{4 \cdot P' \cdot R}{\pi \cdot E^*}$ $k = \frac{2 \cdot P'}{\pi \cdot b^2}$
--	---

### 1.3.- PRINCIPIO VARIACIONAL

Los Principios Variacionales tienen gran importancia en la Mecánica, sirviendo como base para el desenvolvimiento de los métodos numéricos. Las Formulaciones Variacionales pueden ser usadas de tres maneras: La primera, algunos problemas de mecánica se plantean en términos de encontrar los extremos (mínimos o máximos) y por lo tanto, por su naturaleza, pueden ser formulados en términos de declaración variacional. La segunda, hay problemas

que pueden ser formulados por otros medios, como la mecánica vectorial (leyes de Newton) y estos también pueden ser formulados por medio variacional. Y la tercera, las formulaciones variacionales forman una sólida base para la obtención de soluciones a problemas prácticos. Por ejemplo, el Principio de la Energía Potencial Mínima Total, puede ser considerado como un sustituto de las Ecuaciones de Equilibrio de un cuerpo elástico para ser usado en el cálculo de desplazamientos y tensiones en el método de elemento finito. Las formulaciones variacionales pueden servir para unificar diversos campos, sugerir nuevas teorías y dar un poderoso medio para el estudio de la existencia y unicidad de soluciones a los problemas.

### Personajes

Existen grandes personajes en la historia que dieron nuevas formulaciones al campo de la ciencia. Entre ellos tenemos a Jean Bernoulli, Isaac Newton, Leibniz, L'Hopital, D'Alembert's, Euler, Lagrange, Hamilton. En base a sus formulaciones de estos personajes y a medida del pasar del tiempo surgieron otros personajes como son Legendre, Jacobi, Weierstrass. A mediados del siglo XIX, el uso de las formulaciones variacionales fueron surgiendo en el campo de mecánica; entre ellos: Kirchhoff, Lamé, Green y Kelvin que trabajaron en el campo de la elasticidad; Betti, Maxwell, Castigliano, Menabrea y Engesser en sistemas estructurales discretos. Al inicio del siglo XX, las contribuciones a los métodos variacionales de aproximación y sus aplicaciones a problemas físicos fueron surgiendo; Rayleigh, Ritz y Galerkin. Con los trabajos de Hellinger y Reissner sobre principios variacionales mixtos en problemas elásticos dan origen de la los principios variacionales modernos a mediados del siglo XX.

En el estudio de la formulación variacional de problemas continuos, se encuentran funciones de variables dependientes que son a su vez funciones de otros parámetros. Así tenemos la función

$$F\left(y, \frac{dy}{dx}, x\right)$$

y la integral, comúnmente llamado funcional:

$$I = \int_a^b F\left(y, \frac{dy}{dx}, x\right) dx = I[y(x)]$$

Ahora, escogiendo la función  $y(x)$  dado que el funcional  $I[y(x)]$  es un máximo o mínimo o estacionario.

Considerando que:  $y(x) \rightarrow y(x) + \alpha \eta(x)$

$$\left. \frac{dI}{d\alpha} \right|_{\alpha=0} = 0$$

$$\begin{aligned} I(\alpha) &= \int_a^b F(y + \alpha\eta, y' + \alpha\eta', x) dx \\ &= I(0) + \alpha \int_a^b \left( \frac{\partial F}{\partial y} \eta + \frac{\partial F}{\partial y'} \eta' \right) dx + O(\alpha^2) \end{aligned}$$

en la que se cumple:

$$\int_a^b \left( \frac{\partial F}{\partial y} \eta + \frac{\partial F}{\partial y'} \eta' \right) dx = 0$$

e integrando por partes, tenemos:

$$0 = \eta \left. \frac{\partial F}{\partial y'} \right|_a^b + \int_a^b \left( \frac{\partial F}{\partial y} - \frac{d}{dx} \frac{\partial F}{\partial y'} \right) \eta dx$$

donde:

$$\frac{\partial F}{\partial y} - \frac{d}{dx} \frac{\partial F}{\partial y'} = 0$$

es conocida como la ecuación de Euler-Lagrange.

Muchas leyes de la mecánica y la física se reducen a la afirmación de que cierta funcional debe alcanzar su mínimo o su máximo en el proceso considerado, dichas leyes reciben el nombre de Principios Variacionales de la mecánica o la física.

### 1.3.1.- FUNCIONAL DE LA MÍNIMA ENERGÍA POTENCIAL

Las ecuaciones diferenciales que describen los desplazamientos y los esfuerzos para un caso lineal son las Ecuaciones de Equilibrio, Ecuaciones de Compatibilidad y las Ecuaciones de Esfuerzo-Deformación. El Funcional de la Mínima Energía Potencial es una expresión de las ecuaciones diferenciales del Equilibrio.

$$\pi_p(\mu_i) = \int_V \left( \frac{1}{2} \cdot \varepsilon_{ij} \cdot \varepsilon_{kl} \cdot C_{ijkl} - \mu_i \cdot \bar{b}_i \right) dV - \int_{S_\sigma} \mu_i \bar{T}_i \cdot dS$$

Para verificar la equivalencia del funcional con las ecuaciones de equilibrio, se aplica las ecuaciones de Euler-Lagrange.

En el Interior:

$$\frac{\partial F}{\partial \mu_i} - \frac{\partial}{\partial x_j} \cdot \frac{\partial F}{\partial \mu_{i,j}} = 0 \Rightarrow \sigma_{ij,j} + b_i = 0$$

En el Borde:

$$\mu_j \cdot \frac{\partial F}{\partial \mu_{i,j}} + \frac{\partial F}{\partial \mu_i} = 0 \Rightarrow \mu_j \cdot \sigma_{ij} + \bar{T}_i = 0$$

Los desplazamientos y las deformaciones son de la forma:

$\mu = N \cdot a$ $\varepsilon = B \cdot a$	<p><i>N</i>: Función de forma</p> <p><i>B</i>: Derivadas de las Funciones de Forma</p> <p><i>a</i>: Desplazamientos</p>
---	---

Reemplazando en el funcional:

$$\pi_p = \int_V \frac{1}{2} a^T \cdot B^T \cdot C \cdot B \cdot a \cdot dV - \int_V a^T \cdot N^T \cdot \bar{b} \cdot dV - \int_{S_\sigma} a^T \cdot N^T \cdot \bar{T} \cdot dS$$

Haciendo estacionario al funcional:

$$K \cdot a = f$$

donde:

$$K = \int_V B^T \cdot C \cdot B \cdot dV$$

$$f = \int_V N^T \cdot \bar{b} \cdot dV + \int_{S_\sigma} N^T \cdot \bar{T} \cdot dS$$

### 1.3.2.- FUNCIONAL DE LA ENERGÍA POTENCIAL COMPLEMENTARIA

Este funcional es una expresión de las ecuaciones diferenciales de la Compatibilidad y se puede expresar así:

$$\pi_c(\sigma_{ij}) = \int_V \left( \frac{1}{2} \cdot C_{ijkl} \cdot \sigma_{ij} \cdot \sigma_{kl} - \mu_i \cdot \bar{b}_i \right) \cdot dV - \int_{S_u} \bar{\mu}_i \cdot T_i \cdot dS$$

Las funciones incógnitas en este modelo son los esfuerzos unitarios, los cuales se supone en el interior y frontera de cada elemento mediante la relación tipo:

$$\{\sigma\} = [H].\{p\} + \{\Omega\}$$

los parámetros  $\{p\}$  definen a las funciones incógnitas y mediante esta relación se formula otra que relacionen los esfuerzos en el volumen con las fronteras:

$$\{T\} = [R].\{p\}$$

Ambas relaciones, reemplazadas en el funcional:

$$\pi_c = \frac{1}{2}.P^T.f.P - P^T.U + C$$

Haciendo estacionario el  $\pi_c$ , se construye un sistema de ecuaciones para calcular los  $\{P\}$ . Una desventaja de este modelo es que una vez calculado los parámetros  $\{P\}$ , se pueden calcular los esfuerzos en cualquier punto del sólido siempre y cuando se obtenga una función  $[H]$  adecuada y que cumpla las condiciones de equilibrio, luego las deformaciones y finalmente los desplazamientos, teniendo que integrar la ecuación de compatibilidad y obteniendo resultados variados según el método de integración.

Una forma de cumplir las condiciones de equilibrio, es representar los esfuerzos con la función de Airy.

### 1.3.3.- FUNCIONAL DE REISSNER

El funcional de Reissner, es un funcional mixto, pues involucra dos tipos de variables, los desplazamientos y los esfuerzos. Y establece que haciendo estacionario la expresión anterior, satisface las condiciones de compatibilidad y equilibrio.

$$\begin{aligned} \pi_p(\mu_i, \lambda_i) = & \int_V \left( \frac{1}{2} \cdot \varepsilon_{ij} \cdot \varepsilon_{kl} \cdot C_{ijkl} - \mu_i \cdot \bar{b}_i \right) \cdot dV + \int_V \left\{ \varepsilon_{ij} - \frac{1}{2} \cdot (\mu_{i,j} + \mu_{j,i}) \right\} \lambda_1 \cdot dV \\ & - \int_{S_\sigma} \mu_i \cdot \bar{T}_i \cdot dS + \int_{S_\mu} (\mu_i - \bar{\mu}_i) \lambda_2 \cdot dS \end{aligned}$$

donde:

$$\begin{aligned} \lambda_1 &= -\sigma \\ \lambda_2 &= T = \eta \cdot \sigma \end{aligned}$$

reformulando el funcional:

$$\pi_p(\mu_i, \sigma_{ij}) = \int_V \left( \frac{1}{2} \cdot \varepsilon_{ij} \cdot \varepsilon_{kl} \cdot C_{ijkl} - \mu_i \cdot \bar{b}_i \right) \cdot dV + \int_V \left\{ -\varepsilon_{ij} + \frac{1}{2} \cdot (\mu_{i,j} + \mu_{j,i}) \right\} \sigma_{ij} \cdot dV$$

$$- \int_{S_\sigma} \mu_i \cdot \bar{T}_i \cdot dS + \int_{S_\mu} (\mu_i - \bar{\mu}_i) \cdot \eta_j \cdot \sigma_{ij} \cdot dS$$

Existen otros procedimientos matemáticos para tratar el problema estructural, tenemos entre los más destacados al Funcional de Hu-Washizu  $(\mu, \sigma, \varepsilon)$ , Modelo Híbrido de Esfuerzos  $(\mu, \sigma)$  y Energía Complementaria Modificada  $(\mu, \sigma)$ .

#### 1.4.- GENERADORES DE MALLAS

Este tema es de suma necesidad en cualquier problema computacional en la que la geometría juega un papel importante como es el caso del análisis del Elemento Finito y tiene lugar como respuesta al problema de la generación automática de mallas triangulares.

La generación automática de mallas de polígonos es tan solo uno de los aspectos de un campo de la computación llamado Geometría Computacional. A dicho campo compete el análisis, diseño, implementación y evolución de procedimientos destinados a la ejecución de tareas de índole geométrica, requeridos por ciertas áreas de la ingeniería.

Los sistemas físicos continuos son generalmente modelados usando ecuaciones diferenciales parciales, teniendo la necesidad de ser discretizados en un número finito de puntos. Una malla es un conjunto de puntos con coordenadas en 2D o 3D que representan una superficie y una estructura que describe como estos puntos son conectados en triángulos. Existen dos tipos de mallas que pueden caracterizarse por la forma en que los nodos están conectados; las Mallas Estructuradas y las Mallas No Estructuradas, siendo este último el más usado por el refinamiento selectivo en zonas diferentes de la malla de forma más sencilla. Por la necesidad de tener mejores resultados, usando el método de elementos finitos, surgieron otros métodos complementarios a los mencionados, siendo el más conocido las mallas adaptivas.

- **Mallas Estructurales**

Estos tipos de malla tienen una conectividad regular, es decir, que cada punto tiene el mismo número de puntos vecinos.

Son de fácil construcción pero tiene el inconveniente cuando se requiere construir mallas sobre regiones irregulares.

- **Mallas No-Estructurales**

Estas mallas tienen una conectividad irregular, es decir, que cada punto puede tener un número diferente de vecinos.

Estas mallas pueden controlar la densidad en zonas específicas y los ángulos interiores de los triángulos deben ser aproximadamente iguales para evitar problemas de estabilidad numérica.

- **Mallas Adaptivas**

Las mallas Adaptivas es un método de refinamiento automático de la malla con la tendencia de llegar a la "exactitud" de los resultados. Este método está basado en la Teoría de Errores del sistema, específicamente en los errores procedente de la discretización de la malla. Uno de los factores que contribuyen al éxito de cualquier proceso de refinamiento adaptable es disponer de un generador de mallas eficientes que permita generar elementos de diferentes tamaños a través de la información obtenida de la estimación del error (condición de error global y condición de malla óptima).

Las ideas básicas para la generación de mallas pueden clasificarse en tres grandes grupos, teniendo presente sus limitaciones e inconvenientes tanto geométricos como computacionales.

**Método de Frente de Avance:** Consiste en construir el mallado elemento a elemento empezando desde un frente inicial, creando nuevos puntos y los conecta con los puntos del frente actual y construye los elementos del mallado. Así, el espacio que aun no está mallado es gradualmente disminuido cuando el frente va avanzando a través del dominio.

**Método de Delaunay:** Consiste en generar puntos de alguna manera y luego conectarlos formando triángulos que satisfagan la condiciones de optimización. Este método se describirá posteriormente.

**Método de Árboles Cuaternarios:** Consiste en superponer al dominio una cuadrícula en la que las celdas exteriores son descartadas y el resto son subdivididas de acuerdo al grado de discretización deseada, mientras que la zona que se encuentra en la frontera son distretizados usando cualquier método mencionado anteriormente.

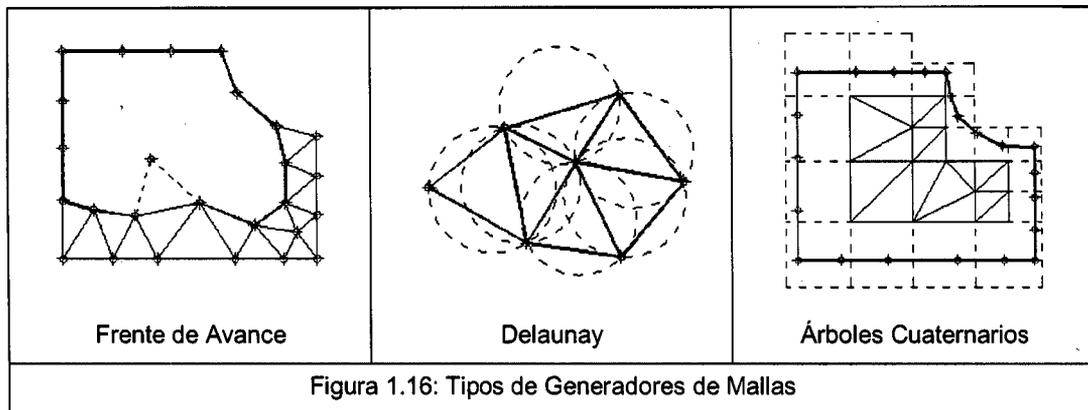


Figura 1.16: Tipos de Generadores de Mallas

#### 1.4.1- TRIANGULACIÓN DE DELAUNAY

Primero definiremos el concepto de triangulación. Una triangulación es una subdivisión de una determinada área en triángulos. Una triangulación de una nube de puntos del plano es una familia de triángulos de interiores disjuntos cuyos vértices son puntos de la nube y en cuyo interior no hay ningún punto de la nube. Puede obtenerse una triangulación añadiendo, mientras sea posible, segmentos rectilíneos que unan puntos de la nube que no atraviesen a los segmentos considerados anteriormente.

Para una misma cantidad y posición de puntos, existen varias formas de triangular siendo los triángulos más regulares las que aparentemente tendrían mejores comportamientos al análisis planteado.

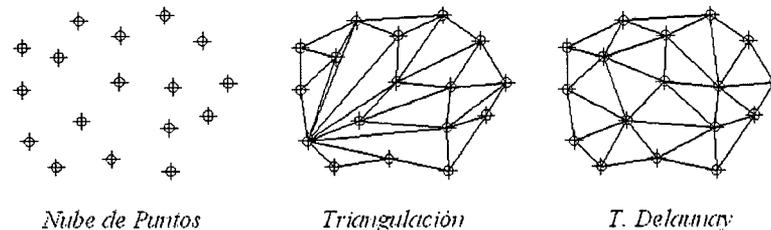


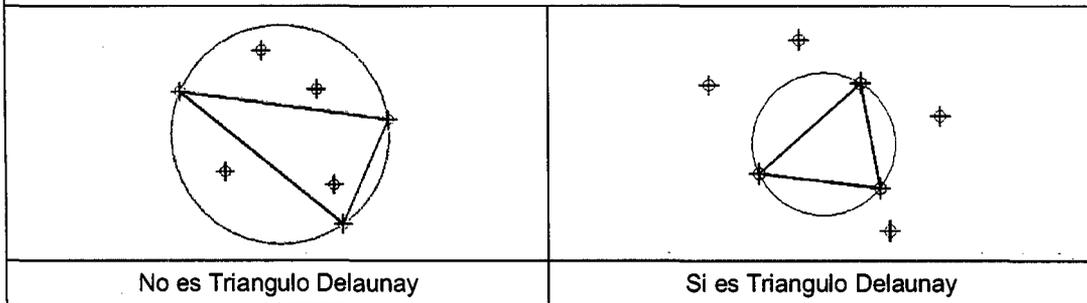
Figura 1.17: Triangulación de Delaunay

La triangulación de Delaunay es una estructura geométrica de gran popularidad en las generaciones de mallas triangulares. Consiste en un conjunto de triángulos cuyos vértices no se interceptan con ningún otro; es decir, que para cada triángulo debe existir un circuncírculo que no contenga ningún otro vértice.

Proposición 1.

$P = \{p_1, p_2, p_3, \dots, p_n\}$  puntos en el plano.

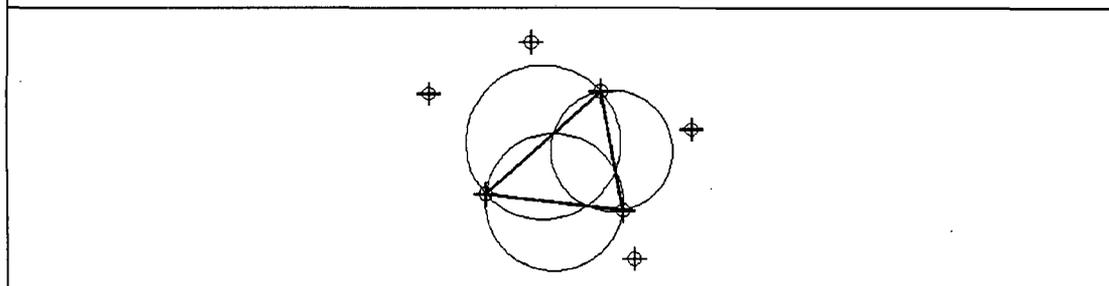
$T(p_i, p_j, p_k)$  es un triángulo de Delaunay si y sólo si  $C(p_i, p_j, p_k)$  no contiene a ningún punto de  $P$  en su interior.



Proposición 2.

$P = \{p_1, p_2, p_3, \dots, p_n\}$  puntos en el plano.

$(p_i, p_j)$  es una arista del triángulo de Delaunay si y sólo si existe un círculo a través de  $(p_i, p_j)$  que no contiene a ningún punto de  $P$  en su interior.



Existen diversas y variadas técnicas para la construcción de la triangulación de Delaunay. La más común es el método incremental, Este procedimiento consiste inicialmente, en calcular un triángulo que contenga en su interior todos los puntos que contendrá la malla, para luego subdividirlo a medida que los puntos son insertados incrementalmente en la triangulación. Esta subdivisión es realizada de acuerdo a su posición con respecto a la triangulación en la que se encuentre el punto insertado, desde esta perspectiva un punto puede caer en dos posiciones generales; en el interior de cualquier triángulo ó sobre la arista de

uno de ellos, para cada uno de estos casos se realiza un procedimiento de intercambio de arista para permitir el ingreso del punto si es necesario.

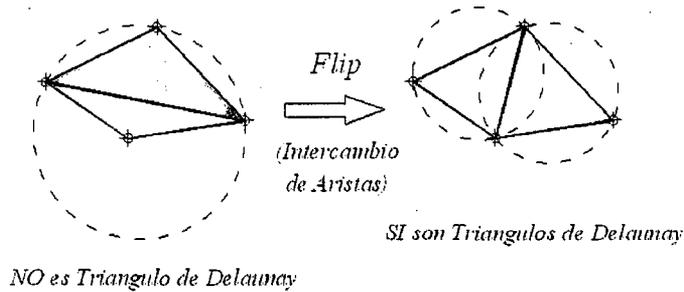


Figura 1.18: Acondicionamiento del Triangulo de Delaunay,

### Implementación Computacional

Para implementación Computacional de la Triangulación de Delaunay se tiene como base a una variante del algoritmo mencionado y que se presenta a continuación: En primer lugar, transformar el dominio bidimensional donde esta contenido los puntos en un plano espacial (hiperboloide) a través de la transformación  $(x, y) \rightarrow (x, y, z)$  en que  $z = x^2 + y^2$ . Esto corresponde a proyectar todos los puntos en un hiperboloide centrado en el origen en coordenadas globales.

Luego se escoge un punto  $i$  cualquiera y se verifica junto con otros dos puntos  $j$  y  $k$  si forman un triangulo Delaunay. Se dice que es un triangulo Delaunay si el resto de los puntos de la nube de puntos analizada están por encima del plano formado por los puntos  $i, j, k$ . Se analiza de esta manera para todos los puntos del sistema, convirtiéndose en una secuencia repetitiva.

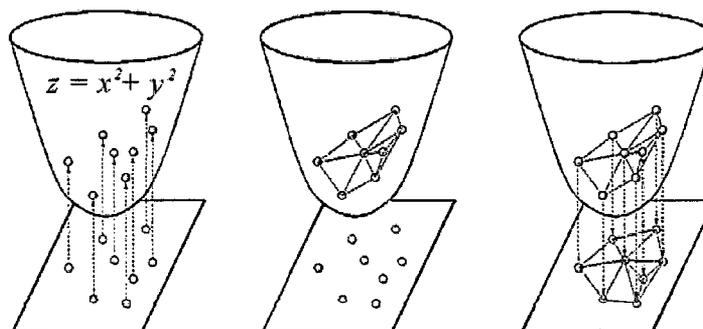


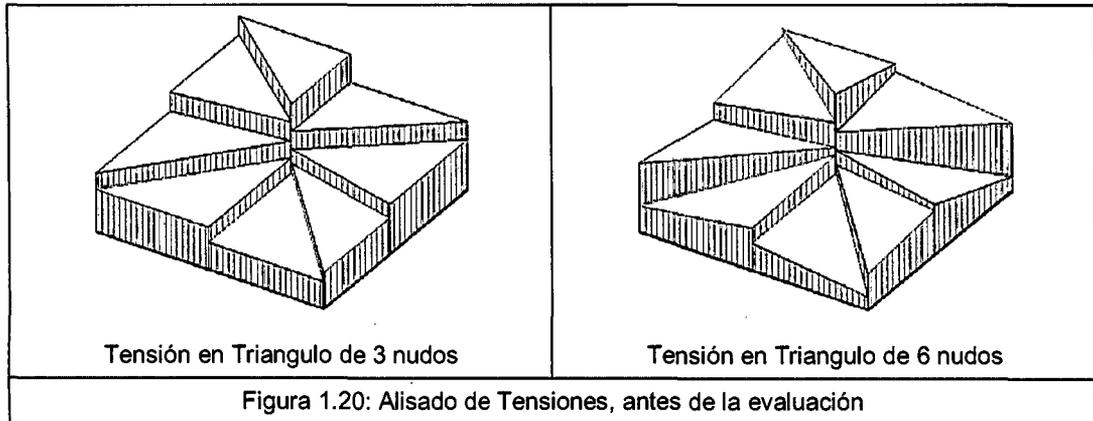
Figura 1.19: Triangulación Delaunay: Envoltente Convexa

### 1.5.- ALISADOS DE TENSIONES

El cálculo de las Tensiones en el nudo de cada elemento es de forma indirecta a través de la formula:

$$\sigma = D\varepsilon = DB\delta^e$$

Originando de esta manera una discontinuidad de los valores de las tensiones entre cada elemento adyacente tal como se muestra en la grafica.



Esto se debe a que el Principio Variacional empleado se tiene como variable al campo de desplazamiento y no a las tensiones.

Una medida de mejorar estas variaciones de las tensiones en los nudos es refinando la malla ya sea de manera automática u de forma manual en las zonas adecuadas, pero siempre existirá esta variación, quedando la alternativa de alisar estos valores nodales.

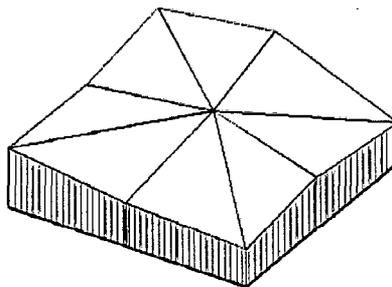


Figura 1.21: Alisado de Tensiones, resultado final

Como la integración de la rigidez está hecha en los puntos de integración de Gauss, por consecuencia las tensiones también están hechas en esos mismos puntos y que para elementos triangulares de tres nudos solo basta hacer un

alisado directo, pero para el caso de los elementos triangulares de seis nudos, estos deberán ser extrapolados de los puntos de integración a los puntos nodales.

### 1.5.1.- MÉTODOS DE ALISADOS

Existen tres métodos de Alisar las Tensiones, las cuales se describen a continuación:

- **Método Promedio Directo**

Este método realiza el cálculo promediando las Tensiones en el nudo  $i$  entre todos los elementos que llegan a él o realizando el promedio Ponderado con las áreas. Este método es de simple aplicación e implementación computacional.

$$\bar{\sigma}_i = \frac{\sum_e \sigma_i^e}{n_i^e} ; \bar{\sigma}_i = \frac{\sum_e A^e \sigma_i^e}{\sum_e A^e}$$

$n$  = Número de elementos que llegan al nudo  $i$

$A$  = Área del elemento

- **Método Global**

Se calcula estimando el error de las tensiones alisadas y las tensiones original no alisadas. Se calcula el error cuadrático en un elemento cualquiera y luego se buscan las tensiones alisadas en los nudos que minimicen el error cuadrático. Al realizar estas operaciones se obtienen términos matriciales. Este método es más eficaz pero tiene más costo computacional.

Error entre la tensión alisada y la original  $\xi = \sigma_a - \sigma = N \cdot \bar{\sigma}^e - D \cdot B \cdot \delta^e$

Error cuadrático en un elemento cualquiera:  $\xi_2 = \int (\sigma_a - \sigma)^2 dv$

Minimizando el error cuadrático:  $\frac{\partial \xi_2}{\partial \bar{\sigma}^e} = 0 \Rightarrow \int 2 \cdot \frac{\partial \xi}{\partial \bar{\sigma}^e} \cdot \xi \cdot dv = 0$

Para el estudio de una componente  $\sigma$  cualquiera de la tensión:

$$\sigma = D_f \cdot \varepsilon = D_f \cdot B \cdot \delta^e$$

Siendo:  $D_f$  la fila correspondiente a  $\sigma$  en la matriz  $D$  (matriz constitutiva del material)

$$f = 1 \Rightarrow \sigma_x ; f = 2 \Rightarrow \sigma_y ; f = 3 \Rightarrow \tau_{xy}$$

Reemplazando y sustituyendo:

$$\int N_f^T (N_f \bar{\sigma}^e - \sigma) dv = 0$$

$$\int N_f^T \cdot N_f \cdot dv \cdot \bar{\sigma}^e = \int N_f^T \cdot \sigma \cdot dv$$

$$M^e \bar{\sigma}^e = R^e$$

$$M_{ij}^e = \int N_i \cdot N_j \cdot dv$$

$$R_i^e = \int N_i \cdot D_f \cdot B \cdot dv \cdot \delta^e$$

Siendo  $N_f$  el vector fila con las funciones de Interpolación del elemento

- **Método Local**

Este método calcula la tensión alisada reduciendo el error entre las tensiones en los puntos de integración antes de extrapolar a los nudos en cada elemento, una vez obtenido la tensión en los nudos, se procede a promediar dichas tensiones como antes para obtener las tensiones definitivas.

$$E_G = \sum_G (W_G (\sigma_{aG} - \sigma_G))^2 = \sum_G W_G^2 (N_{jG} \bar{\sigma}^e - D_f B_G \delta^e)^2$$

$W_G$  = Factores de peso en cada punto

Minimizando el error respecto a  $\bar{\sigma}^e$ :

$$\sum_G H_G N_{jG}^T N_{jG} \bar{\sigma}^e = \sum_G H_G N_{jG}^T D_f B_G \delta^e$$

Esta ecuación tiene una similitud al método anterior ( $M^e \bar{\sigma}^e = R^e$ ), pero de manera individual del elemento y no de forma global.

Para la presente tesis se ha utilizado dos métodos (método promedio directo y método global) con la finalidad de ver las variaciones entre ambos métodos.

Ver el procedimiento de cálculo en el capítulo de anexos.

## 1.6.- DESARROLLO DE SISTEMAS DE ECUACIONES

El desarrollo de sistemas de ecuaciones es un tema tratado en el campo de los Métodos Numéricos, las cuales existen varios métodos de desarrollo para estas ecuaciones.

Para el caso del desarrollo del Elemento Finito, es un tema importante dado que es la parte principal del método, teniendo presente que el MEF se concluye en calcular las variables de un sistema de ecuaciones ( $K \cdot u = F$ ); además, en los cálculos de las Tensiones Alisadas también concluye en desarrollar el sistema

de ecuaciones ( $M \cdot \sigma = R$ ). Según sea el método a usar para el desarrollo del sistema de ecuaciones variarían los resultados, pero no obstante a nivel de ingeniería estas variaciones no son de importancia pero sí el tiempo de procesamiento, salvo en algunos casos.

Los métodos más comunes para el desarrollo de estos sistemas de ecuaciones son: Eliminación de Gauss, Descomposición LU, Gradiente Conjugada, entre otros.

Dado que se está aplicando los dos Métodos de Regulación (Penalidad y Multiplicadores de Lagrange) se usarán dos métodos de Resolución del Sistema de Ecuaciones que son: Eliminación de Gauss y una variante del método de Descomposición LU.

El método de Eliminación de Gauss usado, tiene una adecuación importante en el sentido de optimizar los recursos de memoria computacional, se trata en usar arreglos unidimensionales, teniendo dos variables (vector diagonal y vector fila o columna).

A medida que se estaba desarrollando la tesis, en especial cuando se trató de usar el método de Multiplicadores de Lagrange, el método de eliminación de gauss no funcionaba por el hecho que debería de desarrollar una matriz inversa con algunos ceros en la diagonal debido a las condiciones de borde de la estructura analizada. Motivo por el cual se optó por usar el método de Descomposición LU con cierta variación en su algoritmo.

## CAPITULO II METODO DE ELEMENTO FINITO

El Método de Elemento Finito (MEF) es una herramienta popular para cálculo en la mecánica estructural así como también para otros campos de la ciencia. Siendo un método de aproximación de problemas continuos a través de sistemas de ecuaciones complejas que dependen de varios grados de libertad, de tal forma que:

- El cuerpo se divide en un número finito de partes (discretización) pasando de esta manera a ser un problema discreto.
- En que las incógnitas del problema dejan de ser funciones matemáticas y pasan a ser resultados de los sistemas matriciales.
- El comportamiento en el interior de cada elemento está en función del comportamiento de los nodos mediante las adecuadas funciones de interpolación o funciones de forma.

Por lo tanto, el Método de Elemento Finito se basa en transformar un cuerpo de naturaleza continua en un modelo discreto aproximado, denominándose de esta manera discretización del modelo. Y lo que sucede en el interior de este modelo del cuerpo aproximado, se obtiene mediante la interpolación de los valores conocidos en los nudos.

El MEF consiste en los siguientes pasos:

1. Desarrollo del modelo.
2. Formulación de las ecuaciones que lo gobiernan.
3. Discretización de las ecuaciones.
4. Solución de las ecuaciones.
5. Interpretación de los resultados.

En esta primera etapa, el modelado es un término que tiende a ser usado para identificar el modelo más simple que puede reproducir el comportamiento de interés y que pueda proporcionar las predicciones cualitativas y cuantitativas de los resultados. El esfuerzo para este paso es importante ya que dependerá de los valores que se obtenga a partir de este modelo.

Para esta segunda y tercera etapa, la Formulación de las Ecuaciones que lo Gobiernan y su Descritización son de suma importancia para los usos que se quieran hacer, es decir en esta etapa se ven los Principios Variacionales o bajo que teoría matemática se basa el problema a analizar teniendo en cuenta la

sensibilidad de los resultados y la magnitud de los errores que se puedan ocasionar.

En la cuarta etapa, la Solución de las Ecuaciones discretas, presenta muchas opciones en el sentido de que algoritmo matemático se puede usar ya que esto puede generar alguna variación en los resultados así como también el costo del tiempo en el proceso del ordenador.

Y por último, esta etapa es de mucha responsabilidad por parte del usuario en el sentido de cómo interpretar los resultados, siendo sensibles a muchos factores y los resultados pueden depender de parámetros del tipo de material. Siendo el usuario consciente de estos eventos, teniendo bastante la posibilidad de tener una mala interpretación de los resultados.

A pesar de algunas desventajas, la utilidad y potencial del MEF lineal u no lineal son muy optimistas, siendo las empresas industriales las que mayor provecho sacan sobre este método reduciendo de alguna manera las pruebas de prototipos.

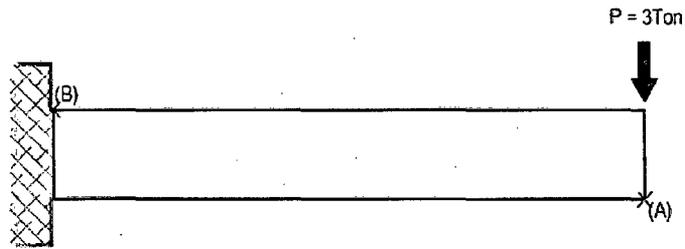
Como se ha visto en el capítulo anterior, el MEF está basado en un Principio Variacional según el tipo y alcance del problema a analizar, para nuestro caso usaremos el Principio Variacional de Energía Potencial para cumplir la condición de Equilibrio, teniendo en cuenta las Condiciones de Continuidad y Relaciones Constitutivas, obteniendo de esta manera el modelo más común en el análisis estructural llamado Modelo de Desplazamiento.

## 2.1.- DESARROLLO GEOMETRICO DEL MODELO

Existe gran variedad de modelos geométricos que pueden reproducir el mejor comportamiento a un modelo. Que por su geometría podemos citar a los Elementos Triangulares y Elementos Cuadriláteros. Cada uno de estos elementos tiene a su vez varios tipos de clase como son elementos triangulares de 3 y 6 nudos, elementos cuadriláteros Lagrangianos de 4, 9 y 16 nudos, elementos cuadriláteros Serendípitos de 4, 8 y 12 nudos y elementos cuadriláteros Incompatibles.

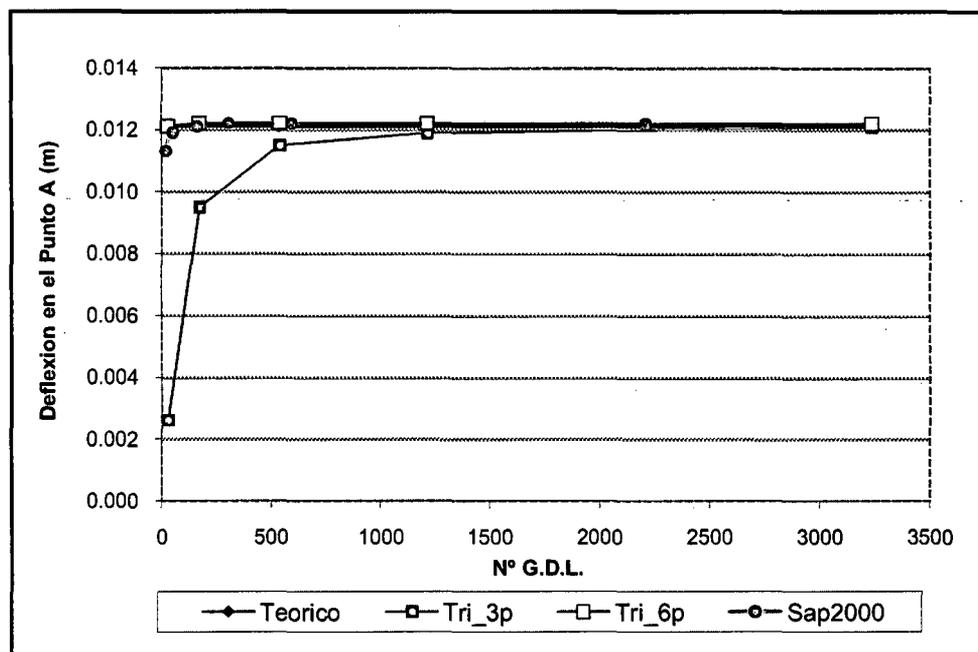
Por tratarse el método como un elemento discreto, una tendencia a mejorar los resultados es discretizando más el elemento en la zona de estudio, así podemos citar algunos resultados obtenidos según la tabla adjunta.

Para el caso de una viga empotrada en un extremo.



Máxima Deflexión Teórica = 0.0121m

$$\delta = \frac{P.L^3}{3.E.I}$$

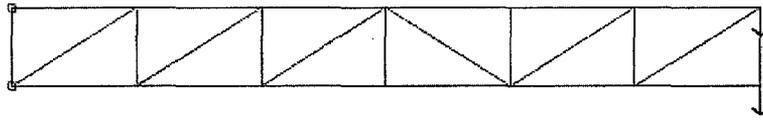


Modelo	# G.D.L	$\delta_y$ 3pto Penalty	% Error1	$\delta_y$ 6pto Penalty	% Error2
V1-1	28	0.0026	21.5	0.0121	100
V1-2	172	0.0095	78.5	0.0122	100.8
V1-3	540	0.0115	95	0.0122	100.8
V1-4	1216	0.0119	98.3	0.0122	100.8
V1-5	3238	0.0121	100	0.0122	100.8

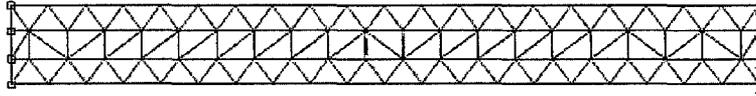
$$\%Error1 = \frac{\delta_y 3p}{\delta_{TEORICO}}$$

$$\%Error2 = \frac{\delta_y 6p}{\delta_{TEORICO}}$$

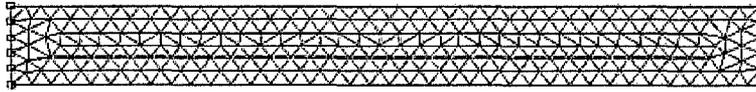
Modelos Discretizados:



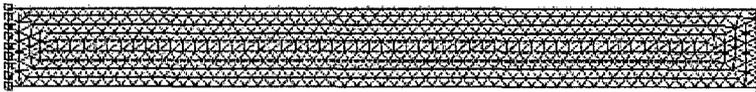
V1-1: 14Nudos, 12Triangulos, 28GDL



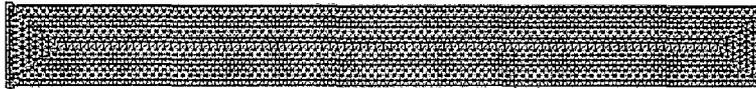
V1-2: 86Nudos, 124Triangulos, 172GDL



V1-3: 270Nudos, 452Triangulos, 540GDL

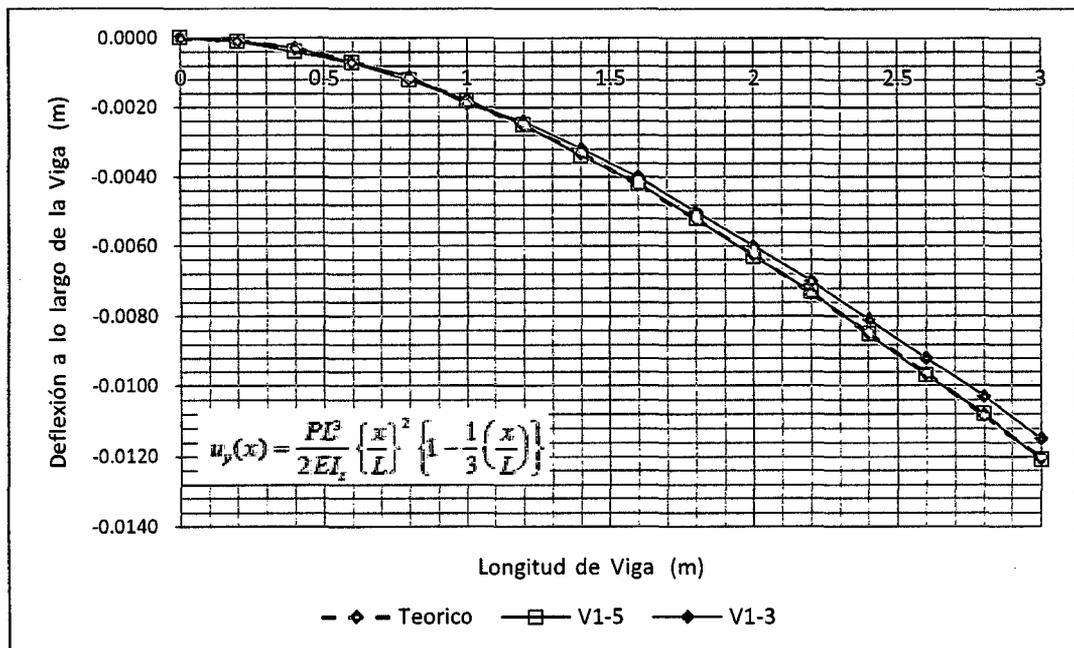


V1-4: 608Nudos, 1078Triangulos, 1216GDL



V1-5: 1619Nudos, 3010Triangulos, 3238GDL

Deflexión Vertical a lo Largo de la Viga (Teórico y Modelos)



## 2.2.- FORMULACION DE ECUACIONES DE GOBIERNO

### 2.2.1.- CAMPO DE DESPLAZAMIENTOS ASUMIDOS

Un importante principio que se tiene en cuenta en el MEF es el Campo de Desplazamiento, que es por lo general una función de las coordenadas espaciales que definen la forma del desplazamiento de un elemento, estas funciones son polinomios que por lo general son polinomios básicos del Triángulo de Pascal siendo recomendable usar polinomios completos ya que así se incrementa la convergencia. Teniéndose presente es que el campo de desplazamiento es sólo un campo “supuesto” o “asumido” que puede o no ser la forma exacta del desplazamiento del elemento.

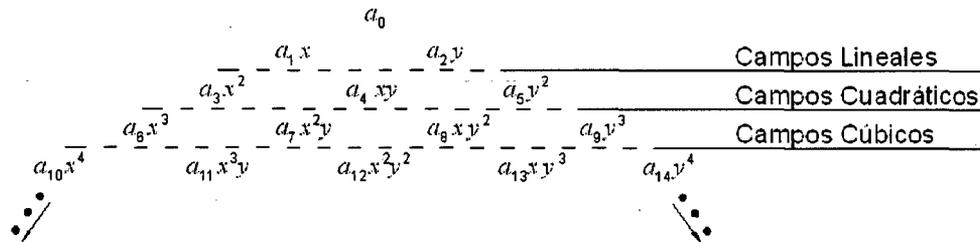


Figura 2.1: Triángulo de Pascal

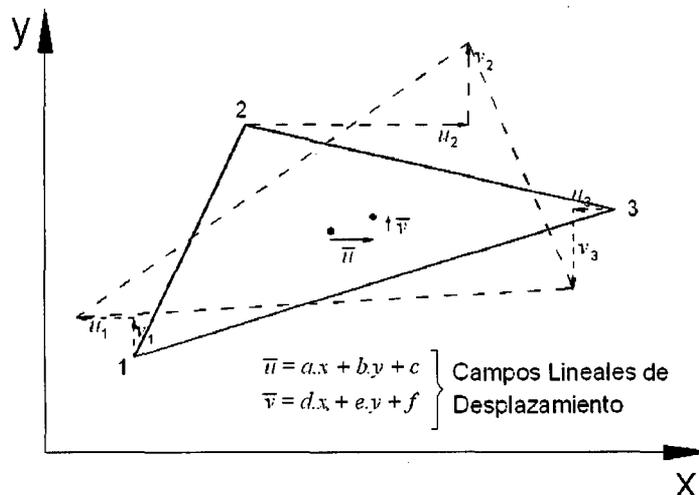


Figura 2.2: Campos de Desplazamiento

- **Discretización del Campo de Desplazamiento**

Siendo la primera etapa del MEF la discretización en elementos finitos, ésta puede darse en elementos triangulares de tres o seis nudos tal como se muestra

en el gráfico, teniendo en cuenta que el análisis reproduce el comportamiento de la malla escogida y no el de la estructura real.

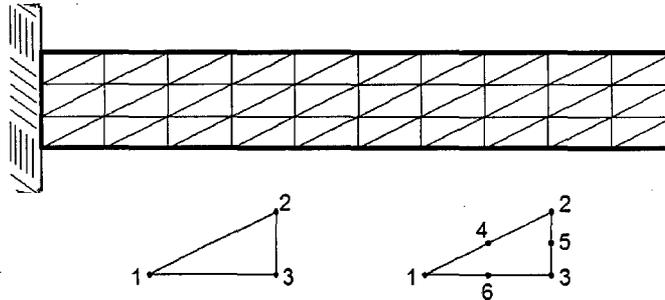


Figura 2.3: Elemento Triangular de 3 y 6 nudos

Si analizamos un elemento de la estructura en estudio, podemos expresar los desplazamientos cartesianos de un punto cualquiera del interior del elemento en función de los desplazamientos de sus nudos como se muestra:

$$u = N_1 \cdot u_1 + N_2 \cdot u_2 + N_3 \cdot u_3$$

$$v = N_1 \cdot v_1 + N_2 \cdot v_2 + N_3 \cdot v_3$$

Donde:

$(u_i, v_i)$  = Desplazamientos horizontal y vertical de cada nudo.

$N_i$  = Función de Forma del nudo  $i$  del elemento.

Escrito de manera matricial:

$$\mathbf{u} = \begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{bmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{Bmatrix}$$

$$\mathbf{u} = \mathbf{N} \cdot \mathbf{a}^{(e)}$$

Donde:

$\mathbf{u} = \begin{Bmatrix} u \\ v \end{Bmatrix}$  Vector desplazamiento de un punto del elemento.

$\mathbf{N} = [N_1, N_2, N_3]$  Matriz de Función de Forma del elemento.

$$N_i = \begin{bmatrix} N_i & 0 \\ 0 & N_i \end{bmatrix} \text{ Matriz de Función de Forma del nudo } i \text{ de elemento.}$$

$$a^{(e)} = \begin{Bmatrix} a_1^{(e)} \\ a_2^{(e)} \\ a_3^{(e)} \end{Bmatrix} \text{ y } a_i^{(e)} = \begin{Bmatrix} u_i \\ v_i \end{Bmatrix} \text{ Vector de Desplazamiento nodal del elemento y del}$$

nudo  $i$

### 2.2.2.- ECUACIONES DE GOBIERNO

Las ecuaciones de gobierno son obtenidas del Principio de Trabajos Virtuales aplicadas a un elemento aislado como se muestra a continuación:

$$\iint_{A^{(e)}} \delta \varepsilon^T \cdot \sigma \cdot t \cdot dA = \iint_{A^{(e)}} \delta u^T \cdot b \cdot t \cdot dA + \oint_{(e)} \delta u^T \cdot t \cdot t \cdot ds + \sum_{i=1}^3 \delta u_i \cdot U_i + \sum_{i=1}^3 \delta v_i \cdot V_i$$

Despejando la ecuación y teniendo en cuenta la arbitrariedad de los desplazamientos virtuales, se deduce a lo siguiente:

$$\iint_{A^{(e)}} B^T \cdot \sigma \cdot t \cdot dA - \iint_{A^{(e)}} N^T \cdot b \cdot t \cdot dA - \oint_{(e)} N^T \cdot t \cdot t \cdot ds = q^{(e)}$$

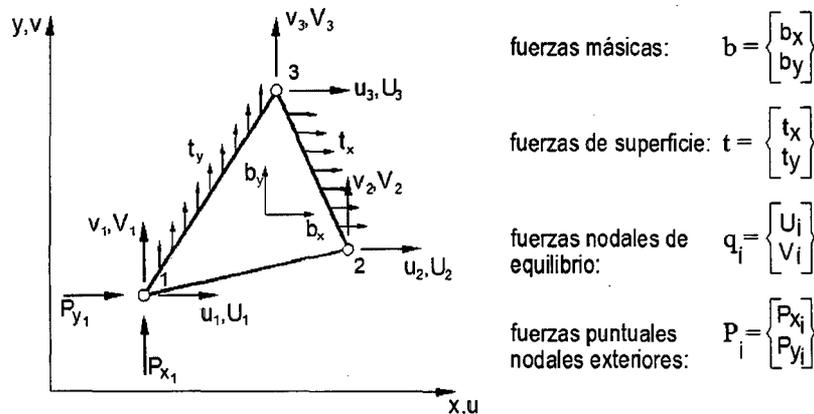


Figura 2.4: Fuerzas Actuantes sobre un elemento triangular

Expresando esta ecuación el equilibrio entre las fuerzas nodales de equilibrio y las fuerzas debidas a la deformación, a las fuerzas másicas y a las de superficie. Reemplazando el vector de Tensiones en la ecuación anterior:

$$\sigma = D(\varepsilon - \varepsilon^0) + \sigma^0 ; \varepsilon = B \cdot a^{(e)}$$

$$\left[ \iint_{A^{(e)}} \mathbf{B}^T \cdot \mathbf{D} \cdot \mathbf{B} \cdot t \cdot dA \right] \mathbf{a}^{(e)} - \iint_{A^{(e)}} \mathbf{B}^T \cdot \mathbf{D} \cdot \boldsymbol{\varepsilon}^0 \cdot t \cdot dA + \iint_{A^{(e)}} \mathbf{B}^T \cdot \boldsymbol{\sigma}^0 \cdot t \cdot dA - \iint_{A^{(e)}} \mathbf{N}^T \cdot \mathbf{b} \cdot t \cdot dA - \oint_{\Gamma^{(e)}} \mathbf{N}^T \cdot \mathbf{t} \cdot t \cdot ds = \mathbf{q}^{(e)}$$

o

$$\mathbf{K}^{(e)} \cdot \mathbf{a}^{(e)} - \mathbf{f}^{(e)} = \mathbf{q}^{(e)}$$

donde:

$$\mathbf{K}^{(e)} = \iint_{A^{(e)}} \mathbf{B}^T \cdot \mathbf{D} \cdot \mathbf{B} \cdot t \cdot dA$$

es la matriz de rigidez del elemento, y

$$\mathbf{f}^{(e)} = \mathbf{f}_{\varepsilon}^{(e)} + \mathbf{f}_{\sigma}^{(e)} + \mathbf{f}_b^{(e)} + \mathbf{f}_t^{(e)}$$

es el vector de fuerzas nodales equivalentes del elemento.

Estas ecuaciones pueden ser expresadas en términos que relacionen los nodos i y j del elemento de la siguiente manera:

Matriz de Rigidez	$\mathbf{K}_{ij}^{(e)} = \iint_{A^{(e)}} \mathbf{B}_i^T \cdot \mathbf{D} \cdot \mathbf{B}_j \cdot t \cdot dA$
Fuerzas Repartidas por unidad de área en el nudo i	$\mathbf{f}_{b,i}^{(e)} = \iint_{A^{(e)}} \mathbf{N}_i^T \cdot \mathbf{b} \cdot t \cdot dA$
Fuerzas Repartidas sobre el contorno en el nudo i	$\mathbf{f}_{t,i}^{(e)} = \oint_{\Gamma^{(e)}} \mathbf{N}_i^T \cdot \mathbf{t} \cdot t \cdot ds = \oint_{\Gamma^{(e)}} \begin{Bmatrix} \mathbf{N}_i \cdot t_x \\ \mathbf{N}_i \cdot t_y \end{Bmatrix} \cdot t \cdot ds$
Fuerzas debidas a Deformaciones iniciales en el nudo i	$\mathbf{f}_{\varepsilon,i}^{(e)} = \iint_{A^{(e)}} \mathbf{B}_i^T \cdot \mathbf{D} \cdot \boldsymbol{\varepsilon}^0 \cdot t \cdot dA$
Fuerzas debidas a Tensiones iniciales en el nudo i	$\mathbf{f}_{\sigma,i}^{(e)} = - \iint_{A^{(e)}} \mathbf{B}_i^T \cdot \boldsymbol{\sigma}^0 \cdot t \cdot dA$

### 2.3.- DISCRETIZACION DE ECUACIONES

La discretización de ecuaciones para el desarrollo del MEF se basa principalmente en las ecuaciones matemáticas que ayudan a obtener la matriz de rigidez.

### 2.3.1.- FUNCIONES DE FORMA

- **Sistema de Coordenadas Naturales**

Antes de describir la importancia de las Funciones de Forma, se mencionara el uso del Sistema de Coordenadas Naturales. Todas las soluciones del elemento finito requieren una evaluación de integrales, algunos son de fácil evaluación como otros son imposibles evaluarlos analíticamente cuando se hace uso del sistema de coordenadas globales. Todo eso se hace más fácil si se transforma el sistema de coordenadas globales a naturales. De tal manera que la geometría de un aspecto irregular inicial del elemento triangular pasaría ser un triángulo rectángulo con ejes coordenados  $\alpha$  y  $\beta$  de manera que el elemento tenga los lados sobre los ejes  $\alpha = 0$ ,  $\beta = 0$  y  $1 - \alpha - \beta = 0$  como se muestra en la figura.

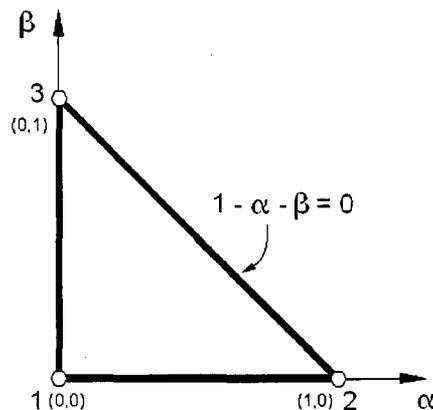


Figura 2.5: Coordenadas Naturales sobre un elemento triangular

- **Funciones de Forma en Elementos tipo Barra**

Las funciones de Forma o de Interpolación se usan para definir el campo de desplazamiento dentro de un elemento.

Considerando un elemento finito de dos nudos como se muestra en el grafico.

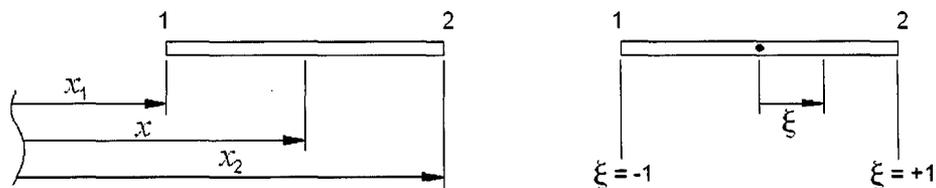


Figura 2.6: Elemento Barra Típico lineal en coordenadas  $x$  y  $\xi$

Ahora definamos un sistema de coordenadas naturales como se indica.

$$\xi = \frac{2}{x_2 - x_1}(x - x_1) - 1$$

Lo que queda es que el campo de desplazamiento desconocido dentro de un elemento será interpolado por una distribución lineal para este caso.

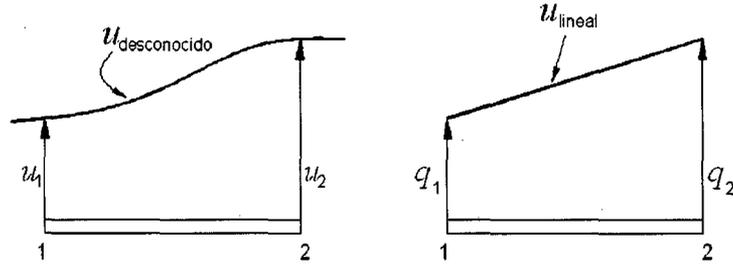


Figura 2.7: Interpolación Lineal del Campo de Desplazamiento

Para implementar esta interpolación lineal, se introducirán funciones de forma lineales como:

$$N_1(\xi) = \frac{1 - \xi}{2} \quad N_2(\xi) = \frac{1 + \xi}{2}$$

Donde sus gráficas son de la manera siguiente:

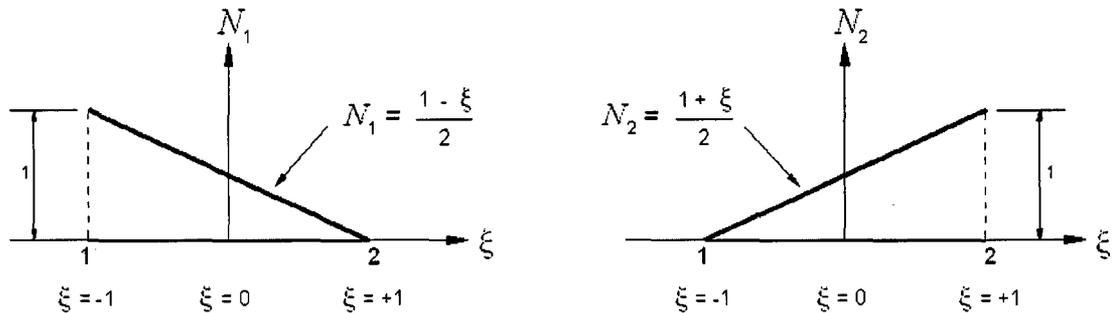


Figura 2.8: Función de Interpolación Lineal

Quedando definida las funciones de forma, el campo de desplazamiento lineal dentro del elemento puede escribirse en términos de desplazamientos nodales  $q_1$  y  $q_2$  como:

$$u = N_1 \cdot q_1 + N_2 \cdot q_2$$

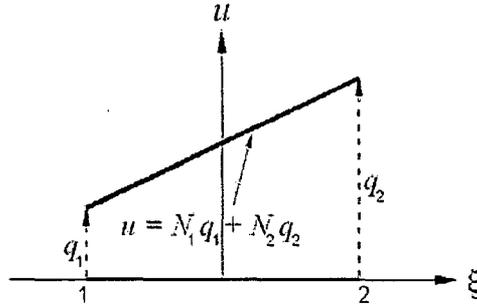


Figura 2.9: Función de Forma Lineal del campo de Desplazamiento

Sin embargo, en algunos problemas, el uso de una interpolación cuadrática conduce a resultados más exactos.

Consideremos un elemento típico cuadrático de tres nodos, tal como se muestra en el gráfico.

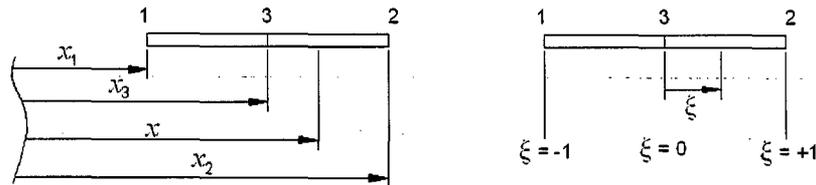


Figura 2.10: Elemento Barra Típico cuadrático en coordenadas  $x$  y  $\xi$

Y las Funciones de Forma Cuadrática son:

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

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

$$N_3(\xi) = (1 + \xi) \cdot (1 - \xi)$$

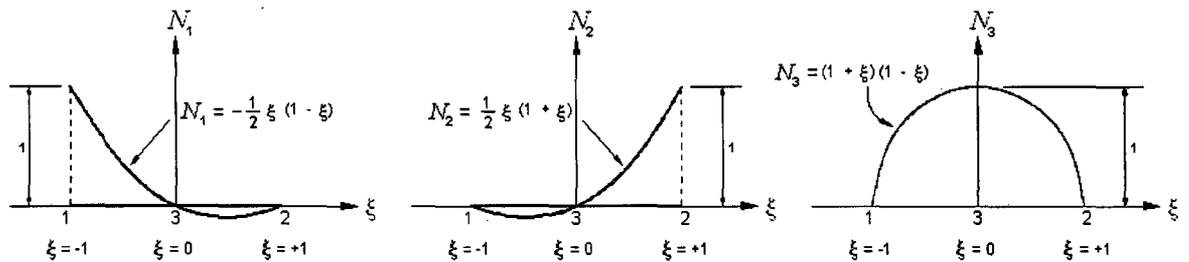


Figura 2.11: Función de Interpolación Cuadrático

El campo de desplazamientos dentro del elemento se escribe en términos de los desplazamientos nodales como

$$u = N_1 \cdot q_1 + N_2 \cdot q_2 + N_3 \cdot q_3$$

Siendo de esta manera una interpolación cuadrática que pasa por  $q_1$ ,  $q_2$  y  $q_3$

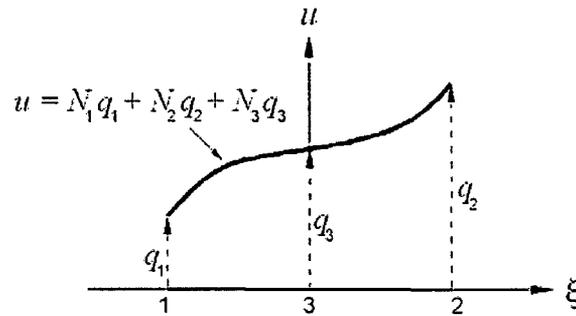


Figura 2.12: Función de Forma Cuadrático del campo de Desplazamiento

- **Funciones de Forma en Elementos Bidimensionales**

Para el caso de elementos triangulares, existen varios tipos de funciones de forma, las cuales solo consideraremos los elementos triangulares de tres y seis nudos que definen aproximaciones completas de primer y segundo grado.

Elementos de tres nudos

$$\phi = \alpha_0 + \alpha_1 x + \alpha_2 y$$

Elementos de seis nudos

$$\phi = \alpha_0 + \alpha_1 x + \alpha_2 y + \alpha_3 xy + \alpha_4 x^2 + \alpha_5 y^2$$

Los valores de  $\alpha_i$  de las ecuaciones anteriores pueden calcularse usando el método de las Coordenadas de Áreas.

- **Coordenadas de Áreas**

Si se une un punto interior P de un triángulo con un área A con los vértices se obtienen tres subáreas  $A_1$ ,  $A_2$  y  $A_3$  tales que  $A_1 + A_2 + A_3 = A$ . Las coordenadas de áreas se define en:

$$L_1 = \frac{A_1}{A} \quad ; \quad L_2 = \frac{A_2}{A} \quad ; \quad L_3 = \frac{A_3}{A}$$

$$L_1 + L_2 + L_3 = 1$$

$$x = L_1 x_1 + L_2 x_2 + L_3 x_3 \quad ; \quad y = L_1 y_1 + L_2 y_2 + L_3 y_3$$

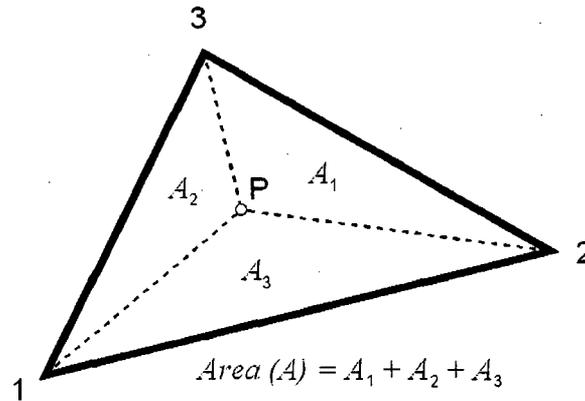


Figura 2.13: Coordenadas de Áreas en elemento triangular

Las Funciones de Forma de elementos triangulares que contienen polinomios completos para el nodo  $i$  viene dada por:

$$N_i = l_i^i(L_1)l_j^i(L_2)l_k^i(L_3)$$

Donde  $l_i^i(L_1)$  es el polinomio de Lagrange de grado  $I$  en  $L_1$

$$l_i^i(L_1) = \prod_{\substack{j=1, I+1 \\ j \neq i}} \frac{(L_1 - L_j^I)}{(L_1^i - L_j^I)}$$

Con idénticas expresiones para  $l_j^i(L_2)$  y  $l_k^i(L_3)$

### Funciones de Forma del elemento triangular lineal de tres nodos

Las funciones de forma son polinomios de primer grado ( $M=1$ ).

Nodo1

Posición  $(I, J, K)$ : (1,0,0)

Coordenadas de área: (1,0,0)

$$N_1 = l_1^1(L_1) = L_1$$

De igual manera:  $N_2 = L_2$  y  $N_3 = L_3$

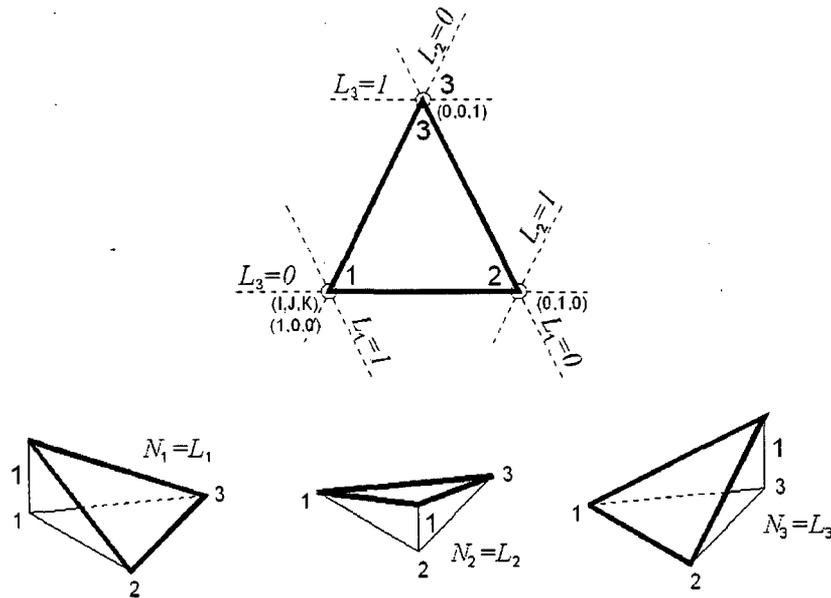


Figura 2.14: Función de Interpolación Lineal en elemento triangular

### Funciones de Forma del elemento triangular cuadrático de seis nodos

Las funciones de forma son polinomios de segundo grado ( $M=2$ ).

Nodo 1

Posición  $(I,J,K)$ :  $(2,0,0)$

Coordenadas de área:  $(1,0,0)$

$$N_1 = l_2^1(L_1) = \frac{(L_1 - 1/2) \cdot L_1}{(1 - 1/2) \cdot 1} = (2L_1 - 1) \cdot L_1$$

Nodo 4

Posición  $(I,J,K)$ :  $(1,1,0)$

Coordenadas de área:  $(1/2, 1/2, 0)$

$$N_4 = l_1^2(L_1) \cdot l_1^2(L_2) = \frac{L_1}{1/2} \cdot \frac{L_2}{1/2} = 4L_1L_2$$

De igual manera, se obtiene para todos los nodos:

$$N_1 = (2L_1 - 1) \cdot L_1 \quad ; \quad N_2 = (2L_2 - 1) \cdot L_2 \quad ; \quad N_3 = (2L_3 - 1) \cdot L_3$$

$$N_4 = 4L_1L_2 \quad ; \quad N_5 = 4L_2L_3 \quad ; \quad N_6 = 4L_1L_3$$

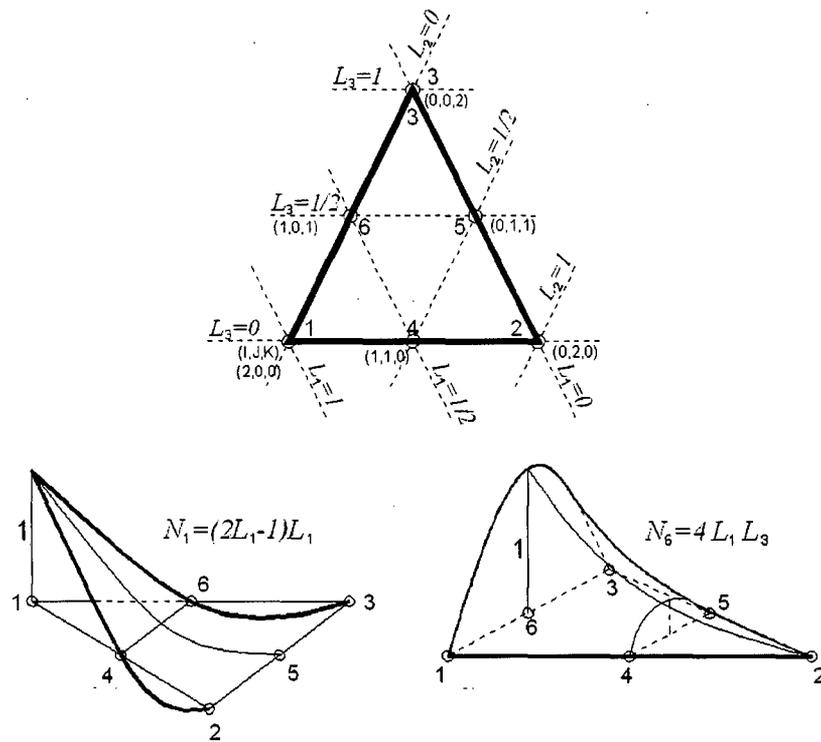


Figura 2.15: Función de Interpolación Cuadrática en elemento triangular

Las Funciones de Forma de los elementos triangulares en el sistema de coordenadas naturales serían:

$$L_1 = 1 - \alpha - \beta ; L_2 = \alpha ; L_3 = \beta$$

Elementos triangulares de tres nudos:

$$N_1 = 1 - \alpha - \beta ; N_2 = \alpha ; N_3 = \beta$$

Elementos triangulares de seis nudos:

$$N_1 = (1 - 2\alpha - 2\beta)(1 - \alpha - \beta) ; N_2 = (2\alpha - 1)\alpha ; N_3 = (2\beta - 1)\beta$$

$$N_4 = 4(1 - \alpha - \beta)\alpha ; N_5 = 4\alpha\beta ; N_6 = 4(1 - \alpha - \beta)\beta$$

### 2.3.2.- ELEMENTOS ISOPARAMETRICOS BIDIMENSIONALES

Para un elemento bidimensional, las coordenadas de un elemento isoparamétrico  $x, y$  también pueden representarse en términos de coordenadas nodales usando las mismas funciones de forma usadas para interpolar la geometría y los desplazamientos. Es decir, se llaman elementos isoparamétricos a aquellos elementos en que las funciones asociadas a los nodos que definen la

posición de sus puntos interiores y las funciones asociadas a los nodos que definen los desplazamientos son las mismas.

$$x = \sum_{i=1}^n N_i(L_1, L_2, L_3) \cdot x_i \quad ; \quad y = \sum_{i=1}^n N_i(L_1, L_2, L_3) \cdot y_i$$

$$L_1 = 1 - \alpha - \beta \quad ; \quad L_2 = \alpha \quad ; \quad L_3 = \beta$$

Estos elementos, tienen desde el punto de vista operacional una gran importancia.

$$\frac{\partial x}{\partial \alpha} = \sum_{i=1}^n \frac{\partial N_i(\alpha, \beta)}{\partial \alpha} \cdot x_i \quad ; \quad \frac{\partial x}{\partial \beta} = \sum_{i=1}^n \frac{\partial N_i(\alpha, \beta)}{\partial \beta} \cdot x_i$$

$$\frac{\partial y}{\partial \alpha} = \sum_{i=1}^n \frac{\partial N_i(\alpha, \beta)}{\partial \alpha} \cdot y_i \quad ; \quad \frac{\partial y}{\partial \beta} = \sum_{i=1}^n \frac{\partial N_i(\alpha, \beta)}{\partial \beta} \cdot y_i$$

$$\frac{\partial N_i}{\partial \alpha} = \frac{\partial N_i}{\partial x} \cdot \frac{\partial x}{\partial \alpha} + \frac{\partial N_i}{\partial y} \cdot \frac{\partial y}{\partial \alpha}$$

$$\frac{\partial N_i}{\partial \beta} = \frac{\partial N_i}{\partial x} \cdot \frac{\partial x}{\partial \beta} + \frac{\partial N_i}{\partial y} \cdot \frac{\partial y}{\partial \beta}$$

$$\begin{bmatrix} \frac{\partial N_i}{\partial \alpha} \\ \frac{\partial N_i}{\partial \beta} \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{\partial x}{\partial \alpha} & \frac{\partial y}{\partial \alpha} \\ \frac{\partial x}{\partial \beta} & \frac{\partial y}{\partial \beta} \end{bmatrix}}_{J^{(e)}} \begin{bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{bmatrix} \quad \begin{bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{bmatrix} = \frac{1}{|J^{(e)}|} \begin{bmatrix} \frac{\partial y}{\partial \beta} & -\frac{\partial y}{\partial \alpha} \\ -\frac{\partial x}{\partial \beta} & \frac{\partial x}{\partial \alpha} \end{bmatrix} \begin{bmatrix} \frac{\partial N_i}{\partial \alpha} \\ \frac{\partial N_i}{\partial \beta} \end{bmatrix}$$

donde  $J^{(e)}$  es la Matriz Jacobiano de la transformación de coordenadas naturales a cartesianas y  $|J^{(e)}|$  es su determinante del Jacobiano, que expresa el diferencial de área en coordenadas naturales.

$$dx \cdot dy = |J^{(e)}| \cdot d\alpha \cdot d\beta$$

Los términos del Jacobiano se calculan usando la transformación isoparamétrica, quedando:

$$J^{(e)} = \sum_{i=1}^n \begin{bmatrix} \frac{\partial N_i}{\partial \alpha} \cdot x_i & \frac{\partial N_i}{\partial \alpha} \cdot y_i \\ \frac{\partial N_i}{\partial \beta} \cdot x_i & \frac{\partial N_i}{\partial \beta} \cdot y_i \end{bmatrix}$$

n=3: elementos triangulares de 3 nudos.

n=6: elementos triangulares de 6 nudos.

### 2.3.3.- MATRIZ DE DEFORMACION

De las ecuaciones básicas de la Elasticidad vistas en el capítulo inicial, el campo de deformaciones para cada tipo de elemento triangular es:

$$\varepsilon_x = \frac{\partial u}{\partial x} = \sum_{i=1}^n \frac{\partial N_i}{\partial x} \cdot u_i \quad \varepsilon_y = \frac{\partial v}{\partial y} = \sum_{i=1}^n \frac{\partial N_i}{\partial y} \cdot v_i \quad \gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} = \sum_{i=1}^n \left( \frac{\partial N_i}{\partial y} \cdot u_i + \frac{\partial N_i}{\partial x} \cdot v_i \right)$$

En forma matricial

$$\varepsilon = \begin{Bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \end{Bmatrix} = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \dots & \frac{\partial N_n}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial y} & \dots & 0 & \frac{\partial N_n}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & \dots & \frac{\partial N_n}{\partial y} & \frac{\partial N_n}{\partial x} \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ \vdots \\ u_n \\ v_n \end{bmatrix}$$

$$\varepsilon = B \cdot a^{(e)}$$

donde:

$$B = [B_1, B_2, \dots, B_n] \quad B_i = \begin{bmatrix} \frac{\partial N_i}{\partial x} & 0 \\ 0 & \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} \end{bmatrix}$$

Matriz de Deformación del Elemento

Matriz de Deformación del Nodo i

n=3: elementos triangulares de 3 nudos.

n=6: elementos triangulares de 6 nudos.

La matriz B también puede expresarse en términos de coordenadas naturales usando las funciones isoparamétricas:

$$B_i(\alpha, \beta) = \frac{1}{|J^{(e)}|} \begin{bmatrix} \bar{b}_i & 0 \\ 0 & \bar{c}_i \\ \bar{c}_i & \bar{b}_i \end{bmatrix} \quad \begin{aligned} \bar{b}_i &= \frac{\partial y}{\partial \beta} \cdot \frac{\partial N_i}{\partial \alpha} - \frac{\partial y}{\partial \alpha} \cdot \frac{\partial N_i}{\partial \beta} \\ \bar{c}_i &= \frac{\partial x}{\partial \alpha} \cdot \frac{\partial N_i}{\partial \beta} - \frac{\partial x}{\partial \beta} \cdot \frac{\partial N_i}{\partial \alpha} \end{aligned}$$

Por lo tanto, la Matriz de Rigidez de un elemento puede escribirse en términos de las coordenadas naturales:

$$K_{ij}^{(e)} = \iint_{A^{(e)}} B_i^T \cdot D \cdot B_j \cdot t \cdot dx \cdot dy = \int_0^1 \int_0^{1-\beta} B_i^T(\alpha, \beta) \cdot D \cdot B_j(\alpha, \beta) \cdot |J^{(e)}| \cdot t \cdot d\alpha \cdot d\beta$$

### 2.3.4.- INTEGRACION NUMERICA

El cálculo de la matriz de rigidez y las de los vectores de fuerzas nodales equivalentes de cada elemento queda finalmente en el desarrollo algunas integrales. Y ésta aun hubiera sido algo laborioso si no se hubiera utilizado las formulaciones isoparametricas en las que se pueden transformar los dominios de las integrales a un espacio de coordenadas naturales.

Tales cálculos pueden hacerse por integración numérica como el método de Gauss, las cuales proporcionan resultados aproximados suficientemente buenos.

La cuadratura de Gauss para elementos triangulares se escribe como:

$$\int_0^1 \int_0^{1-L_3} f(L_1, L_2, L_3) \cdot dL_2 \cdot dL_3 = \sum_{p=1}^{n_p} f(L_{1p}, L_{2p}, L_{3p}) \cdot W_p$$

donde:

$n_p$  es el número de puntos de integración.

$L_{1p}, L_{2p}, L_{3p}$  son los valores de las coordenadas de área

$W_p$  es el peso en el punto de integración

## 2.4.- SOLUCION DE ECUACIONES

Una vez formulado las ecuaciones de gobierno  $K.a = f$  en forma matricial de cada elemento, la etapa siguiente es el ensamblaje de dichas matrices en la ecuación de equilibrio global y su solución del sistema de ecuaciones simultáneas para calcular los movimientos nodales. Teniendo presente que la eficacia total de los resultados del sistema depende de gran medida sobre los procedimientos numéricos usados para la solución de estas ecuaciones y del refinamiento óptimo de la malla usada.

Pero las variables incógnitas no solo son los desplazamientos, sino también los esfuerzos y eso de acuerdo al Principio Variacional que se está utilizando. Los cuales estos esfuerzos tienen un tratamiento posterior a esta etapa llamada alisado de tensiones y que se verá posteriormente, dicho tratamiento también tiene que ver con el desarrollo de ecuaciones  $M^{(e)} \bar{\sigma}^{(e)} = R^{(e)}$ .

El tiempo de procesamiento de estas ecuaciones es otro factor pero no tan importante como lo era antes ya que en la actualidad existen computadoras con procesadores mucho más rápido y de mayor capacidad operacional. Este factor depende de la cantidad de variables a calcular y esta a la vez depende de la discretización geométrica de la estructura, la optimización del ensamblaje de las ecuaciones y el algoritmo matemático usado para desarrollar este sistema.

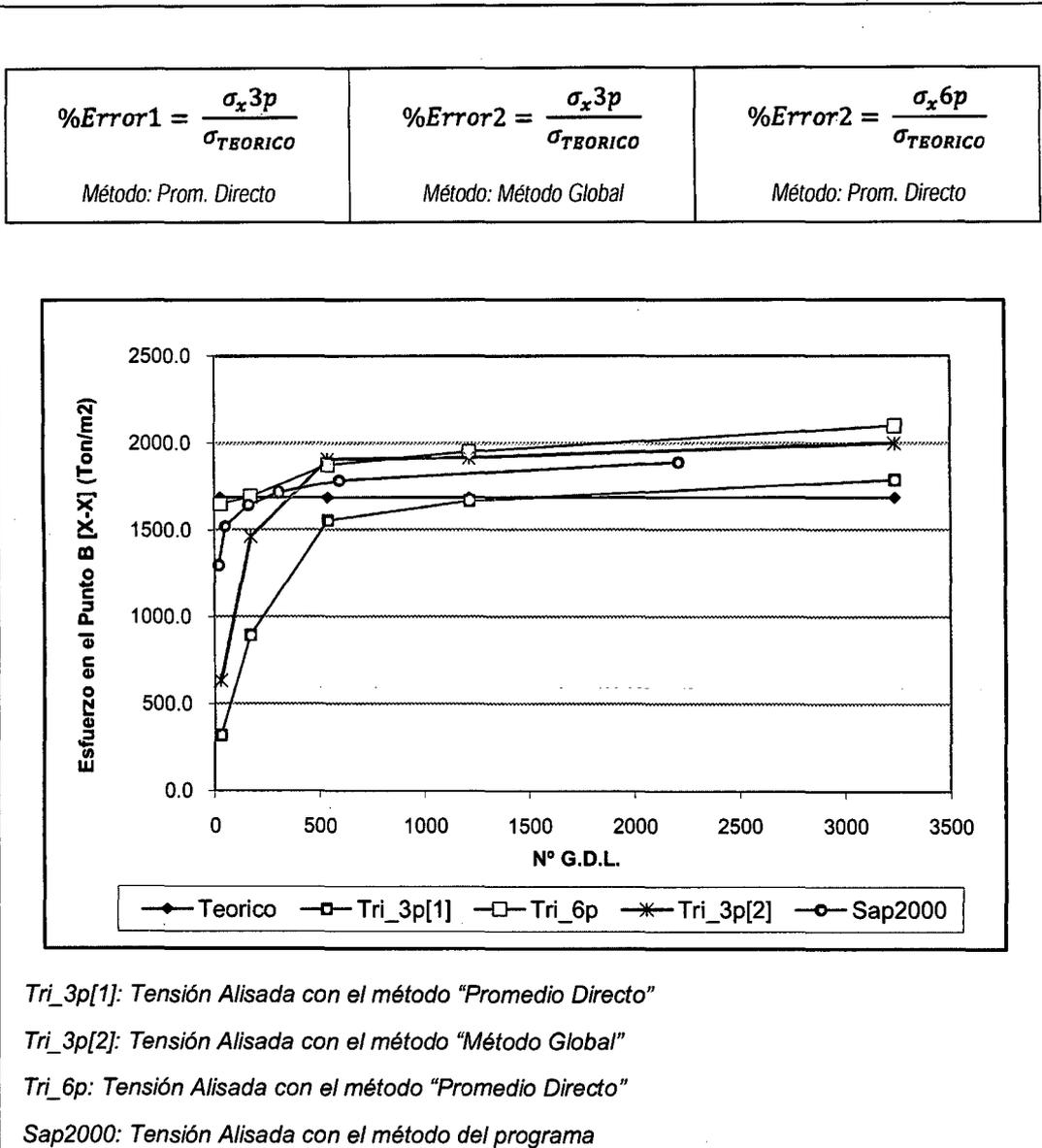
En la siguiente tabla se podrá observar la variación de los resultados en referencia a los esfuerzos.

En referencia al ejemplo mostrado en el capítulo 2.1

Máxima Tensión Teórica = 1687.50 Ton/m<sup>2</sup>

$$\sigma = M \cdot \frac{6}{b \cdot h^2}$$

# G.D.L	$\sigma_x$ 3pto Penalty	% Error1	$\sigma_x$ 3pto Penalty	% Error2	$\sigma_x$ 6pto Penalty	% Error3
28	318.807	18.9	633.312	37.5	1,649.840	97.8
172	896.924	53.2	1,464.720	86.8	1,693.640	100.4
540	1,556.620	92.2	1,904.880	112.9	1,874.320	111.1
1216	1,667.590	98.8	1,915.910	113.5	1,951.240	115.6
3238	1,788.390	106	1,999.120	118.5	2,098.280	124.3



Para el desarrollo del sistema de ecuaciones se ha utilizado dos métodos para observar la variación entre ellos, dichos desarrollos han hecho que se elabore dos programas computacionales alternativos, los métodos son los conocidos métodos de Penalidad y el método de Lagrange. Independientemente sea el elemento triangular de 3 o 6 nudos a usar o talvez la cantidad de elementos a discretizar, siempre se tiene que desarrollar el sistema de ecuaciones.

En el caso del Método de Penalidad, se ha usado arreglos unidimensionales con el fin de optimizar el algoritmo. De esta manera se reduce la cantidad de operaciones innecesarias debido que algunos elementos de la matriz ( $i \neq j$ ) contiene el valor de cero.

En el caso del Método de Lagrange, se ha usado arreglos bidimensionales utilizando una variante del algoritmo numérico LU. Se optó este método distinto al anterior ya que contienen valores de cero en su diagonal de la matriz ( $i = j$ ).

En referencia al ejemplo mostrado en el capítulo 2.1

# G.D.L	$\delta_y$ 3pto Penalty	$\delta_y$ 3pto Lagrange	% Error1	$\sigma_x$ 3pto Penalty	$\sigma_x$ 3pto Lagrange	% Error2
28	0.0026	0.00259	99.6	318.807	318.790	100
172	0.0095	0.00951	100.1	896.924	894.429	99.7
540	0.0115	0.01119	97.3	1,556.620	1,512.660	97.2
1216	0.0119	0.01117	93.9	1,667.590	1,556.700	93.4
3238	0.0121	0.00942	77.9	1,788.390	1,324.830	74.1

$$\%Error1 = \frac{\delta_y 3p [Lagrange]}{\delta_y 3p [Penalty]}$$

$$\%Error1 = \frac{\sigma_x 3p [Lagrange]}{\sigma_x 3p [Penalty]}$$

Los valores de las Tensiones corresponden a las Tensiones Alisadas con el método Promedio Directo.

## 2.5.- INTERPRETACION DE LOS RESULTADOS

Una desventaja no muy significativa del método del elemento finito es la cantidad de resultados que se obtienen. Es decir por cada elemento discretizado se obtienen los desplazamientos nodales, los esfuerzos en cada dirección y los esfuerzos principales, a mayor necesidad de precisión mayor sería la cantidad de elementos discretizados. Pero esta desventaja puede ser subsanada si los resultados son vistos de una manera gráfica.

Otra observación en esta etapa ya antes mencionada es el alisado de las tensiones. Cuando se calcula estas tensiones, estos se hacen de manera independiente al sistema originando una discontinuidad entre cada elemento triangular.

Para la presente tesis, se ha elaborado un programa de postproceso, las cuales se visualizan de manera gráfica los resultados de los desplazamientos y las tensiones.

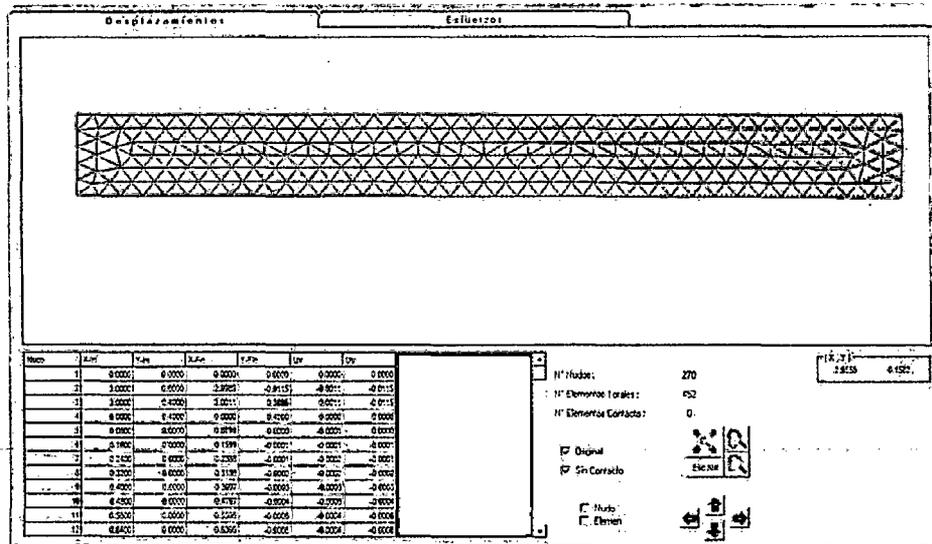


Figura 2.16: Visualización de Resultados referente a los Desplazamientos. Tabla de Coordenadas y Desplazamientos nodales

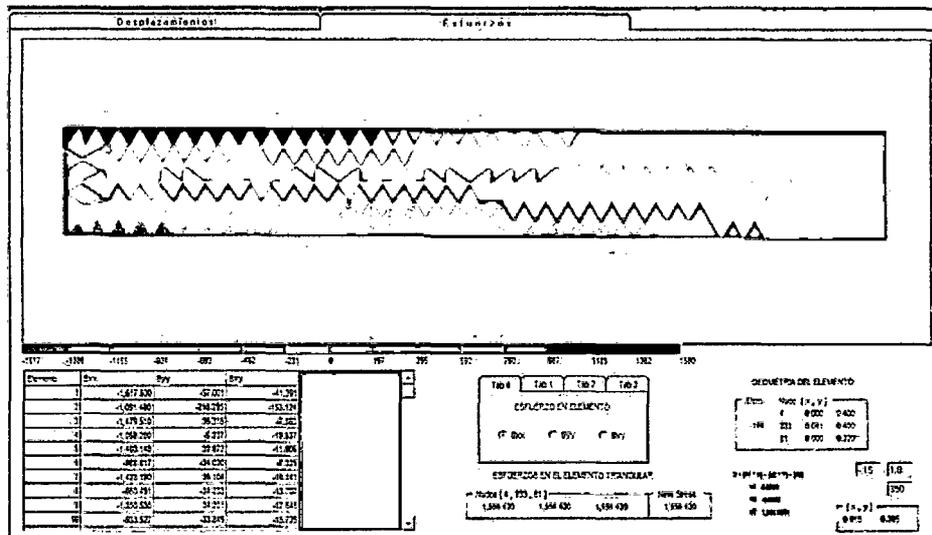


Figura 2.17: Visualización de Resultados referente a las Esfuerzos. Esfuerzos en cada elemento triangular. Tabla de Esfuerzos en cada elemento, Esfuerzos alisados nodales según en el método usado, Cuadro de Esfuerzo en cualquier posición del elemento, Cuadro de Escalas de Colores.

A continuación se presenta los valores de los Esfuerzos en cada elemento y sus respectivos alisados

$$S_x \text{ max} = +1788.39$$

$$S_x \text{ min} = -1739.12$$



Figura 2.18: Viga con Esfuerzo Sin Alisar.

$$S_x \text{ max} = +1556.62$$

$$S_x \text{ min} = -1385.21$$

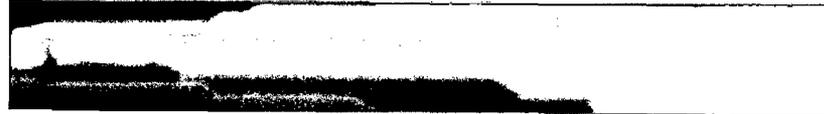


Figura 2.19: Viga con Esfuerzo Alisado (Promedio Directo).

$$S_x \text{ max} = +1904.88$$

$$S_x \text{ min} = -1518.75$$



Figura 2.20: Viga con Esfuerzo Alisado (Método Global).

## CAPITULO III MECANISMO Y OPERADORES DE CONTACTO

### 3.1.- MECANISMOS DE CONTACTO

El análisis de los problemas de contacto es uno de los tópicos más difíciles en la Mecánica de los Sólidos. La colisión es un evento altamente no-lineal y en este fenómeno la no-linealidad no acontece solamente en la no-linealidad geométrica y del material sino también en las condiciones de contorno, y es en estas condiciones cuando el problema es difícil de analizar.

La no-linealidad geométrica surge en las condiciones de contorno cinemáticas desconocidas, no se sabe a priori cual es la región de contacto de los cuerpos. La no-linealidad del material es originada por el agotamiento y envuelve procesos de naturaleza irreversibles.

El problema de contacto abarca desde contacto con pequeños desplazamientos hasta contactos con grandes deformaciones inelásticas.

Existen varios algoritmos matemáticos que describen dicho comportamiento, como el Método Master-Slave, Método Mortar, Método Splitting Pinballs, entre otros.

#### 3.1.1.- CONDICIÓN DE CONTACTO

La condición de impenetrabilidad del contacto entre cuerpos debe ser descrito matemáticamente y esto puede hacerse si los desplazamientos y rotaciones son suficientemente pequeños y la superficie en contacto son bastantes lisos. Otro aspecto importante de contacto es la iteración entre cuerpos de contacto, en la cual para la presente tesis se asumirá como si fuese perpendicular a la superficie en contacto dando así lugar a que los esfuerzos de corte puedan ser determinados con la ayuda de un modelo de fricción como el conocido modelo Ley de Coulomb.

Consideremos ahora que los cuerpos  $\Omega_A$  y  $\Omega_B$  son dos cuerpos impenetrables y con la facultad de poder tener contacto entre ellos. Los Puntos  $P_A(m_A)$  y  $P_B(m_B)$  (donde  $m_A = (m_{1A}, m_{2A})$  componentes de las coordenadas)

pertencientes a la superficie  $\Gamma_A$  y  $\Gamma_B$  respectivamente son llamados puntos de contacto posibles si en un estado  $t$  estas posiciones coinciden.

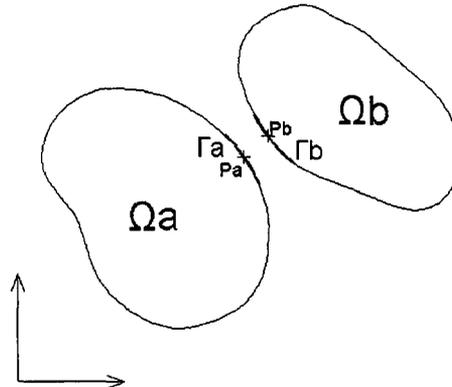


Figura 3.1: Visualización de dos cuerpos antes del contacto

La superficie de contacto común  $\Gamma_C(t)$  es la intersección de  $\Gamma_A(t)$  y  $\Gamma_B(t)$  y como los contornos son asumidos como superficies lisas, las normales  $\vec{n}_A(m_A, t)$  y  $\vec{n}_B(m_B, t)$  son opuestas a los puntos de contacto común

$$\vec{X}_A(m_A, t) = \vec{X}_B(m_B, t) \quad ; \quad \vec{n}_A(m_A, t) = -\vec{n}_B(m_B, t)$$

Los puntos de contacto en común  $P_A(m_A)$  y  $P_B(m_B)$ , usando la ley de acción y reacción son:

$$\vec{P}_A + \vec{P}_B = \vec{0}$$

$$\vec{\sigma}_{nA} + \vec{\sigma}_{nB} = \vec{0} \quad ; \quad \vec{\sigma}_{tA} + \vec{\sigma}_{tB} = \vec{0} \quad \text{en } \Gamma_C \quad (\text{Tensor normal y tangencial})$$

Por lo tanto, el esfuerzo de contacto normal  $\sigma_{nA}$  y  $\sigma_{nB}$  son iguales, justificando la presión de contacto  $\sigma_n$  como:

$$\sigma_n = \sigma_{nA} = \sigma_{nB} \quad \text{en } \Gamma_C$$

La adhesión no es permitida en los puntos de contacto en común, por lo tanto la presión de contacto debe ser menos o igual a cero

$$\sigma_n \leq 0 \quad \text{en } \Gamma_C$$

Los esfuerzos de corte, pueden ser analizados con la ayuda de las ecuaciones constitutivas, lo cual este tema no es detallado en esta tesis.

Un aspecto importante de la descripción de la condición de contacto es la formulación matemática de la impenetrabilidad de ambos cuerpos.

Llamemos a  $\Gamma_{CA}(t)$  a los posibles candidatos del área de contacto de la región  $\Gamma_A(t)$  que puede contactar a  $\Gamma_B(t)$ , y  $\Gamma_{CB}(t)$  a los posibles candidatos del área de contacto de la región  $\Gamma_B(t)$  que puede contactar a  $\Gamma_A(t)$ . En cada punto  $P_A(m_A)$  de  $\Gamma_{CA}(t_0)$  la separación inicial  $S_0(m_A)$  entre  $P_A(m_A)$  y  $\Gamma_B(t_0)$  medida a lo largo de  $\vec{n}_{A0}$  puede ser evaluado viendo la siguiente figura. Donde  $\vec{\mu}_\alpha(m_\alpha, t)$  es el desplazamiento de  $P_\alpha(m_\alpha)$  en un tiempo  $t$  con respecto a una configuración referencial ( $\alpha = A$  o  $\alpha = B$ ).

$$\vec{\mu}_\alpha(m_\alpha, t) = \vec{X}_\alpha(m_\alpha, t) - \vec{X}_\alpha(m_\alpha, t_0)$$

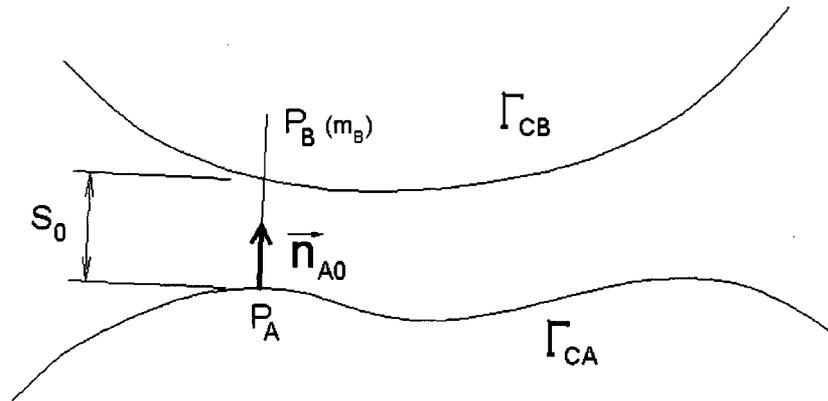


Figura 3.2: Fuerza de Reacción del Contacto

Si la rotación y/o los desplazamientos son pequeños y la superficie de contorno es lisa, ninguna penetración ocurre si:

$$(\vec{\mu}_A - \vec{\mu}_B) \cdot \vec{n}_{A0} - S_0 \leq 0 \text{ en } \Gamma_{CA}(t)$$

Si las rotaciones y/o desplazamientos **no son pequeñas** y las superficies de contorno **no son lisas**, estas inecuaciones no representan las condiciones de impenetrabilidad correctamente. Un ejemplo de esta situación se ve en la siguiente figura:

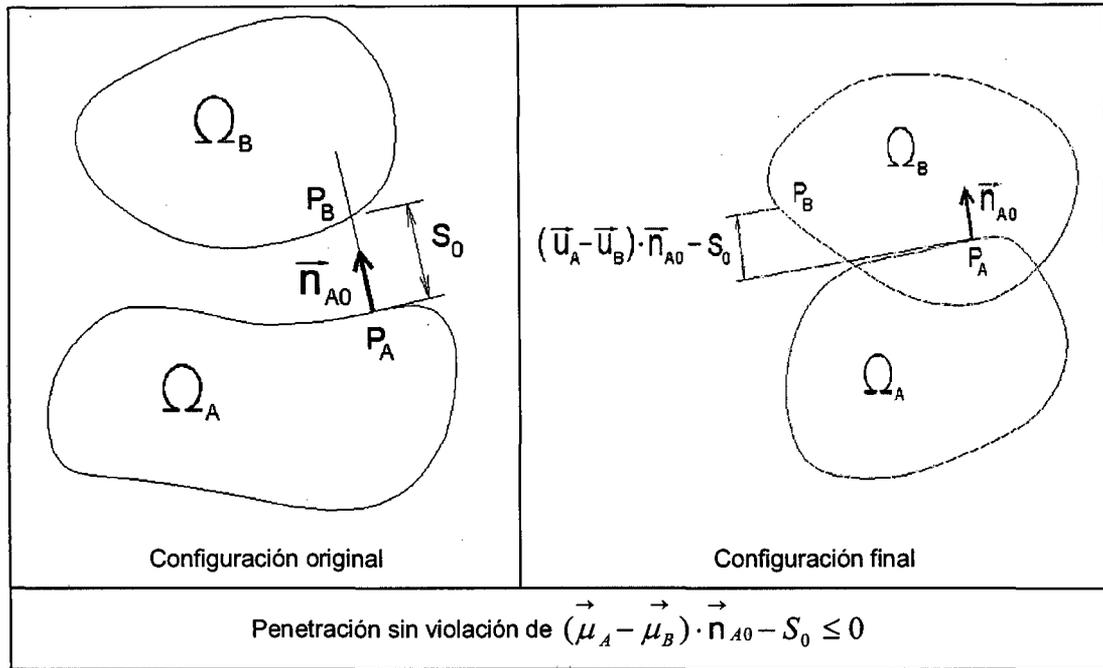


Figura 3.3: Configuración antes y después del contacto

Para este propósito, esto es asumido que en cada estado  $t$  para todo  $\vec{x} \in \Gamma_{CA}(t)$

una cantidad escalar  $g = g(\vec{x}, \Gamma_B(t))$ , pudiendo ser definido como:

$$g(\vec{x}, \Gamma_B(t)) < 0 \text{ si } \vec{x} \notin \Omega_B(t)$$

$$g(\vec{x}, \Gamma_B(t)) = 0 \text{ si } \vec{x} \in \Gamma_B(t) \dots\dots (\alpha)$$

$$g(\vec{x}, \Gamma_B(t)) > 0 \text{ si } \vec{x} \in \Omega_B(t)$$

La notación  $g = g(\vec{x}, \Gamma_B(t))$  indica que  $g$  depende del vector  $\vec{x}$  y la forma de  $\Gamma_B(t)$ . Por lo tanto,  $g = g(\vec{x}, \Gamma_B(t))$  podría ser considerado como una funcional más bien que función. De las ecuaciones anteriores, se ve que ninguna penetración ocurre en el estado  $t$  si y solo si

$$g(\vec{x}, \Gamma_B(t)) \leq 0 \quad \forall \vec{x} \in \Gamma_{CA}(t)$$

Esta desigualdad es un apoyo a la condición de impenetrabilidad, igual en el caso de largos desplazamientos y/o superficies no lisas.

Existen otras consideraciones que se deben de tener en cuenta, como las que pueden ocurrir cuando un punto  $A$  con vector posición común  $\vec{x} \in \Gamma_{CA}(t)$ , con

un vector unitario  $\vec{e} = \vec{e}(x, t)$  intercepta (o se dirigen hacia) a varios puntos pertenecientes a la superficie  $\Gamma_B(t)$ .

Otra consideración a tener en cuenta es cuando el vector unitario no tiene una orientación adecuada al punto  $A$  establecido ( $\vec{e}$  es dirigido hacia dentro o hacia fuera de  $\Omega_A(t)$ ).

Ambas consideraciones últimas satisfacen la ecuación ( $\alpha$ ).

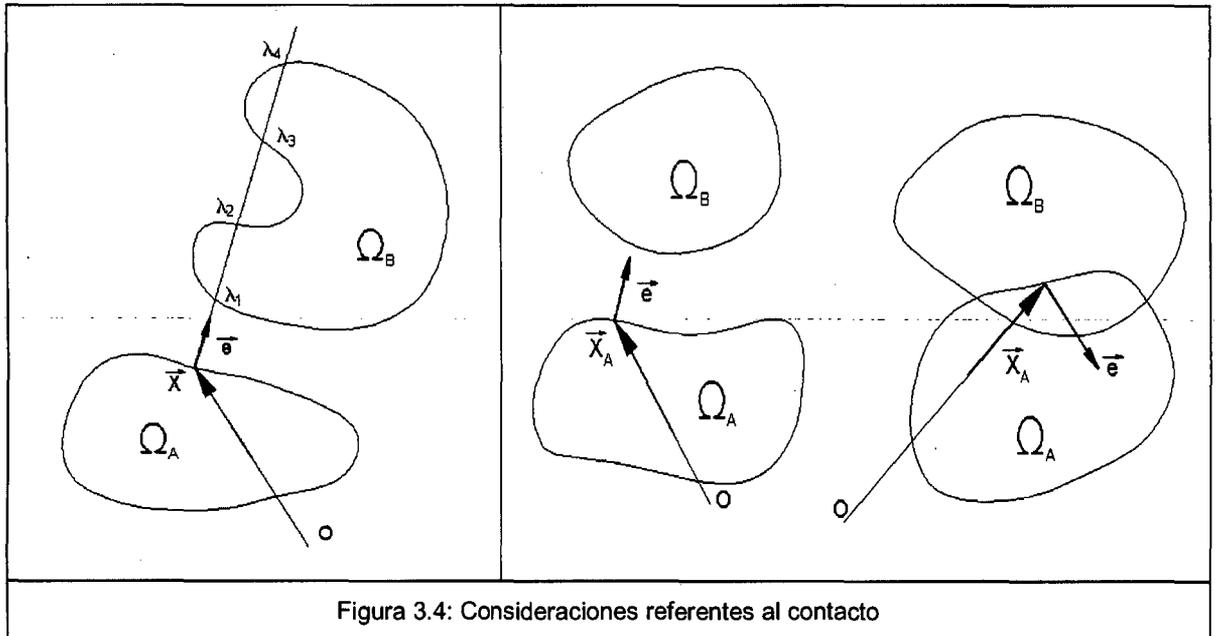


Figura 3.4: Consideraciones referentes al contacto

### 3.2.- METODO DE REGULACIÓN

Con el fin de garantizar la impenetrabilidad de un nudo de un cuerpo a otro cuerpo, la inclusión de este nudo deberá ser igual a cero, satisfaciendo de esta manera las restricciones del problema a lo largo de la superficie de contacto.

Existen dos métodos bien establecidos y de mayor uso en programas comerciales para el caso de la restricción de movimientos nodales, estamos hablando del Método de Multiplicadores de Lagrange y el Método de Penalidad.

Existen otros métodos de uso menos frecuente como los Multiplicadores Aumentados de Lagrange, el método de Barrier y el uso de Ecuaciones Algebraicas.

### Método de los Multiplicadores de Lagrange

Este método obliga que las restricciones de contacto se verifiquen de forma exacta a través de sus multiplicadores de Lagrange. Este método tiene como inconveniente el aumento del número de incógnitas y la aparición de ceros en la diagonal principal de la matriz de rigidez asociada a los multiplicadores y que puede traer dificultades numéricas en el proceso de solución directa.

### Método de Penalidad

El método de Penalidad impone que las condiciones de contacto se verifiquen de forma aproximada, por medio de coeficientes de penalidad. Este método es una alternativa sin que sea necesario aumentar nuevas variables y es de fácil implementación computacional. La desventaja de este método es la posibilidad de hacer un mal acondicionamiento del sistema de ecuaciones dados que los coeficientes de penalidad deberían tender al infinito.

A continuación se presentan dos ejemplos analizados por los dos métodos anteriormente comentados.

#### Ejemplo 01:

Datos del material 1:

$$A_1 = 2400 \text{ mm}^2$$

$$E_1 = 70 \times 10^9 \text{ N/m}^2$$

Datos del material 2:

$$A_2 = 600 \text{ mm}^2$$

$$E_2 = 200 \times 10^9 \text{ N/m}^2$$

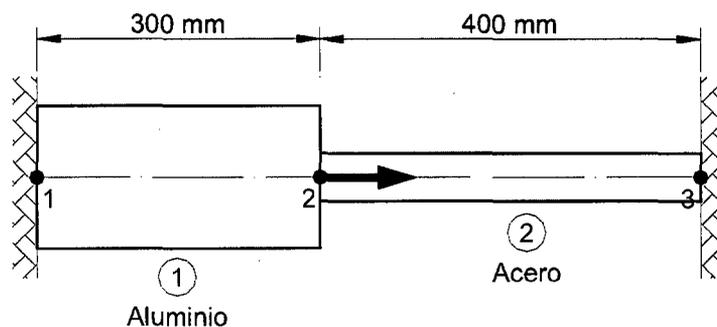


Figura 3.5: Ejemplo sobre el Método de Regulación

Las matrices de rigidez del elemento son:

$$k^1 = \frac{70 \times 10^3 \times 2400}{300} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad y \quad k^2 = \frac{200 \times 10^3 \times 600}{400} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

La matriz de rigidez estructural global que se ensambla a partir de  $k^1$  y  $k^2$  es:

$$K = 10^6 * \begin{bmatrix} 1 & 2 & 3 \\ 0.56 & -0.56 & 0 \\ -0.56 & 0.86 & -0.30 \\ 0 & -0.30 & 0.30 \end{bmatrix}$$

El vector de carga global es:

$$F = [0 \quad 200 \times 10^3 \quad 0]^T$$

y los valores de los desplazamientos prescritos son:

$$u_1 = u_3 = 0$$

La expresión del Funcional es de la Energía Potencial Total de la Estructura

$$\Pi = \frac{1}{2} a^T K a - a^T f$$

Utilizando el **método de Penalidad**: Para mantener la restricción de los nudos 1 y 3 según grafico anterior, se deberá agregar un número tan grande  $\alpha$  a los elementos diagonales primero y tercero de la matriz de rigidez global, así como también se deberá de agregar el número  $(\alpha \times u_1)$  y  $(\alpha \times u_3)$  al primer y tercer componente del vector de carga global.

$$\bar{\Pi}_p = \Pi_p + \frac{1}{2} r^T * \alpha * r$$

$\alpha$  = Matriz diagonal de coeficientes de penalización

$$r = A \cdot u - b ; r = 0$$

La ecuación de equilibrio resultante para valores de  $\alpha$  es:

$$[K + A^T \alpha A] u = f + A^T \alpha b$$

Donde:

$$\left. \begin{array}{l} \alpha = \max. |K_{ij}| \times 10^4 \\ 1 \leq i \leq N ; 1 \leq j \leq N \end{array} \right\} \alpha = [0.86 \times 10^6] \times 10^4$$

Entonces la matriz de rigidez estructural global modificada es:

$$K = 10^6 * \begin{bmatrix} 8600.56 & -0.56 & 0 \\ -0.56 & 0.86 & -0.30 \\ 0 & -0.30 & 8600.30 \end{bmatrix}$$

Resolviendo el sistema:  $K u = F$

$$10^6 * \begin{bmatrix} 8600.56 & -0.56 & 0 \\ -0.56 & 0.86 & -0.30 \\ 0 & -0.30 & 8600.30 \end{bmatrix} * \begin{bmatrix} u1 \\ u2 \\ u3 \end{bmatrix} = \begin{bmatrix} 0 \\ 200 * 10^3 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} u1 \\ u2 \\ u3 \end{bmatrix} = \begin{bmatrix} 15.1432 \times 10^{-6} \\ 0.23257 \\ 8.1127 \times 10^{-6} \end{bmatrix} \text{ mm}$$

Como se puede observar, los resultados de los desplazamientos  $u1$  y  $u2$  aproximadamente tienen el valor de cero y eso dependerá del valor de  $\alpha$  se considere.

Utilizando el método de **Multiplicador de Lagrange** : Este método se basa en añadir al funcional a minimizar un término igual al producto de las condiciones por un conjunto de multiplicadores y que pasan a ser nuevas incógnitas del problema.

$$\begin{aligned} \bar{\Pi}_p &= \Pi_p + \lambda^T (A u - b) & \lambda &= \text{Vector que contiene número de} \\ & & & \text{multiplicadores} \\ A \cdot u &= b \end{aligned}$$

La ecuación de equilibrio resultante es:

$$\begin{bmatrix} K & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} u \\ \lambda \end{bmatrix} = \begin{bmatrix} f \\ b \end{bmatrix}$$

Entonces la matriz de rigidez estructural global modificada es:

$$K = 10^6 * \begin{bmatrix} 0.56 & -0.56 & 0 & 1 & 0 \\ -0.56 & 0.86 & -0.30 & 0 & 0 \\ 0 & -0.30 & 0.30 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Resolviendo el sistema:  $K u = F$

$$10^6 * \begin{bmatrix} 0.56 & -0.56 & 0 & 1 & 0 \\ -0.56 & 0.86 & -0.30 & 0 & 0 \\ 0 & -0.30 & 0.30 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} u1 \\ u2 \\ u3 \\ \lambda1 \\ \lambda2 \end{bmatrix} = \begin{bmatrix} 0 \\ 200 * 10^3 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} u1 \\ u2 \\ u3 \\ \lambda1 \\ \lambda2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.23256 \\ 0 \\ 0.13023 \\ 0.06977 \end{bmatrix} \text{ mm}$$

**Ejemplo 02:**

Datos

$k_{11} = k_{77} = 100$

$f_i = i ; i = 1 .. 7$

$k_{22} = \dots = k_{66} = 200$

$u_2 - u_6 = 0.20$

$k_{12} = k_{23} = \dots = k_{67} = -100$

La matriz de rigidez estructural global es:

$$\begin{bmatrix} k_{11} & k_{12} & & & & & \\ k_{12} & k_{22} & k_{23} & & & & 0 \\ & k_{23} & k_{33} & k_{34} & & & \\ & & k_{34} & k_{44} & k_{45} & & \\ & & & k_{45} & k_{55} & k_{56} & \\ & 0 & & & k_{56} & k_{66} & k_{67} \\ & & & & & k_{67} & k_{77} \end{bmatrix} * \begin{bmatrix} u1 \\ u2 \\ u3 \\ u4 \\ u5 \\ u6 \\ u7 \end{bmatrix} = \begin{bmatrix} f1 \\ f2 \\ f3 \\ f4 \\ f5 \\ f6 \\ f7 \end{bmatrix}$$

Utilizando el **método de Penalidad**

$$\begin{bmatrix} k_{11} + \alpha & k_{12} & 0 & 0 & 0 & 0 & 0 \\ k_{12} & k_{22} + \alpha & k_{23} & 0 & 0 & -\alpha & 0 \\ 0 & k_{23} & k_{33} & k_{34} & 0 & 0 & 0 \\ 0 & 0 & k_{34} & k_{44} & k_{45} & 0 & 0 \\ 0 & 0 & 0 & k_{45} & k_{55} & k_{56} & 0 \\ 0 & -\alpha & 0 & 0 & k_{56} & k_{66} + \alpha & k_{67} \\ 0 & 0 & 0 & 0 & 0 & k_{67} & k_{77} \end{bmatrix} * \begin{bmatrix} u1 \\ u2 \\ u3 \\ u4 \\ u5 \\ u6 \\ u7 \end{bmatrix} = \begin{bmatrix} f1 + 0 * \alpha \\ f2 + 0.20 * \alpha \\ f3 \\ f4 \\ f5 \\ f6 - 0.20 * \alpha \\ f7 \end{bmatrix}$$

$\alpha = [ 200 ] \times 10^4$

$$\begin{bmatrix} 2 \cdot 10^6 & -100 & 0 & 0 & 0 & 0 & 0 \\ -100 & 2 \cdot 10^6 & -100 & 0 & 0 & -2 \cdot 10^6 & 0 \\ 0 & -100 & 200 & -100 & 0 & 0 & 0 \\ 0 & 0 & -100 & 200 & -100 & 0 & 0 \\ 0 & 0 & 0 & -100 & 200 & -100 & 0 \\ 0 & -2 \cdot 10^6 & 0 & 0 & -100 & 2 \cdot 10^6 & -100 \\ 0 & 0 & 0 & 0 & 0 & -100 & 100 \end{bmatrix} \begin{bmatrix} u1 \\ u2 \\ u3 \\ u4 \\ u5 \\ u6 \\ u7 \end{bmatrix} = \begin{bmatrix} 1 \\ 400002 \\ 3 \\ 4 \\ 5 \\ -399994 \\ 7 \end{bmatrix}$$

$$\begin{bmatrix} u1 \\ u2 \\ u3 \\ u4 \\ u5 \\ u6 \\ u7 \end{bmatrix} = \begin{bmatrix} 1.4 \cdot 10^{-5} \\ 0.27001 \\ 0.27502 \\ 0.25002 \\ 0.18502 \\ 0.07003 \\ 0.14003 \end{bmatrix}$$

Utilizando el método de **Multiplicador de Lagrange**

$$\begin{bmatrix} 100 & -100 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ -100 & 200 & -100 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & -100 & 200 & -100 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -100 & 200 & -100 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -100 & 200 & -100 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -100 & 200 & -100 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & -100 & 100 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u1 \\ u2 \\ u3 \\ u4 \\ u5 \\ u6 \\ u7 \\ \lambda1 \\ \lambda2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 0 \\ 0.20 \end{bmatrix}$$

$$\begin{bmatrix} u1 \\ u2 \\ u3 \\ u4 \\ u5 \\ u6 \\ u7 \\ \lambda1 \\ \lambda2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.27 \\ 0.275 \\ 0.25 \\ 0.185 \\ 0.07 \\ 0.14 \\ 28 \\ -24.5 \end{bmatrix}$$

### 3.3.- ALGORITMOS DE INTERACCION

#### 3.3.1.- MASTER-SLAVE

Define que la interacción se desarrolla entre dos cuerpos: cuerpo de contacto y cuerpo objetivo.

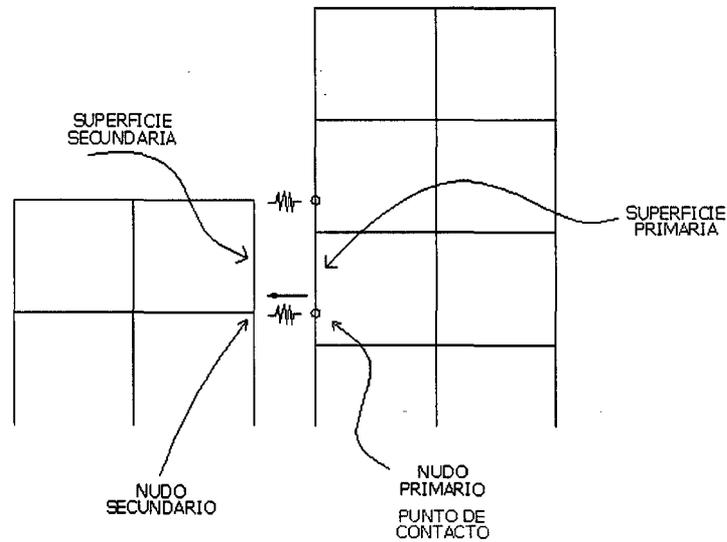


Figura 3.6: Representación del Cuerpo de Contacto y Cuerpo Objetivo

El algoritmo garantizará las ecuaciones de compatibilidad entre las superficies primaria y secundaria, evaluando la posible inclusión de los nodos secundarios en la superficie primaria pero no evalúa la posible inclusión de los nodos primarios en superficie secundaria.

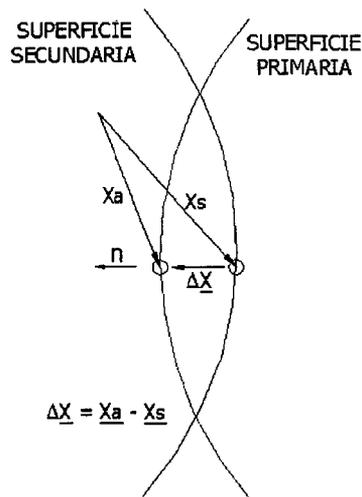


Figura 3.7: Representación de la Inclusión de un cuerpo sobre otro

Inclusión:

$$l = \Delta x \cdot n$$

$l > 0 \rightarrow$  hay contacto entre nudos, aplicar restricción.

$l < 0 \rightarrow$  no hay contacto y no se aplicará restricción.

$$\Delta x = \bar{H} \cdot \bar{X}$$

$\bar{X} \rightarrow$  vector posición de todos los nudos, primarios y secundarios

$\bar{H} \rightarrow$  matriz de interpolación para todos los nudos.

**Ecuación de Compatibilidad:**

$$l = (H X) \cdot n > 0$$

La condición de contacto se garantizará si la inclusión es igual a cero.

$$(H X) \cdot n = 0$$

Nota: Para que la condición se cumpla, se debe de multiplicar el valor a restringir ( $l$ ) por una rigidez muy alta (método de penalidad,  $k \cdot l = 0$ )

**Ecuación de Constitutiva:**

Cuando la inclusión se detecta, se aplica la condición de contacto y se aplica en el nudo secundario una fuerza en la dirección de la normal.

$$f_s^n = F_s \cdot n$$

$f_s^n$  : es la fuerza a aplicar en el nudo secundario en la dirección normal.

$F_s$  : es la magnitud de la fuerza normal en la interface.

Además:

$$F_s = F_s(l) = K_s \cdot l$$

$K_s$  = rigidez alta a aplicar para garantizar la condición de contacto.

**Ecuación de Equilibrio:**

Para mantener el equilibrio, se aplica una fuerza opuesta en el punto de contacto, la cual se distribuye entre los nudos del elemento primario. La distribución se hace en función de las funciones de interpolación.

$$f_e^n = -N^T f_s^n ; = -F_s N^T n$$

$f_e^n$  es el vector de fuerzas aplicadas en los nudos del elemento primario.

**Trabajo Virtual:**

La ecuación de trabajo virtual es:

$$VW = \Delta U_e \cdot fe^n + \Delta U_s \cdot fs^n = ( \Delta U_e \quad \Delta U_s ) \cdot ( fe^n \quad fs^n )$$

$\Delta U_e$  vector de desplazamiento virtual del nudo primario

$\Delta U_s$  vector de desplazamiento virtual del nudo secundario

Reemplazando:

$$VW = ( \Delta U_e \quad \Delta U_s ) \cdot ( -F_s N^T n \quad F_s n )$$

$$VW = - \Delta U \cdot ( F_s H^T n )$$

$$\Delta VW = - \Delta U \cdot ( H^T n ) \Delta F_s$$

$$\Delta F_s = \Delta F_s (l) = K_s \Delta l$$

$$\Delta l = (H \Delta X) \cdot n = n^T (H \Delta U)$$

$$\Delta VW = - \Delta U \cdot ( H^T \cdot n ) \cdot K_s \cdot n^T \cdot (H \Delta u)$$

La contribución a la matriz de rigidez:  $K_s \cdot H^T \cdot n \cdot n^T \cdot H$

La contribución a la matriz de fuerza:  $F_s \cdot H^T \cdot n$

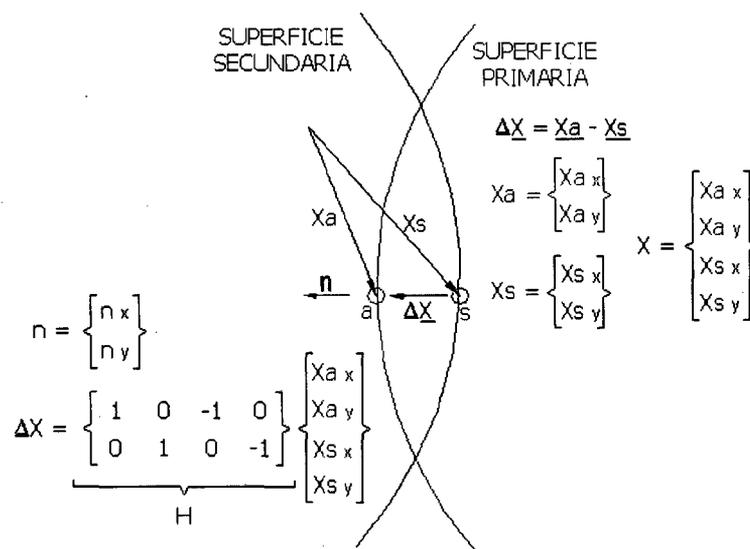


Figura 3.8: Vector Desplazamiento en la Inclusión

$$n^T \cdot H = [n_x \quad n_y \quad -n_x \quad -n_y]$$

$$k^e = K_s \cdot H^T \cdot n \cdot n^T \cdot H$$

$$k^e = K_s \cdot \begin{bmatrix} (n_x.n_x) & (n_x.n_y) & (-n_x.n_x) & (-n_x.n_y) \\ & (n_y.n_y) & (-n_x.n_y) & (-n_y.n_y) \\ \text{Simetrico} & & (n_x.n_x) & (n_x.n_y) \\ & & & (n_y.n_y) \end{bmatrix}$$

NOTA: En el caso de un elemento horizontal, el vector  $n = (n_x \quad n_y) = (1 \quad 0)$

### 3.3.2.- MORTAR ELEMENT

Este método es algo similar al método de Master-Slave, la variación está en que utiliza los multiplicadores de Lagrange para restringir las condiciones de penetrabilidad en vez de utilizar un elemento tipo gap.

La idea es proyectar variacionalmente los desplazamientos de un contacto continuo con los operadores directamente por transferencia.

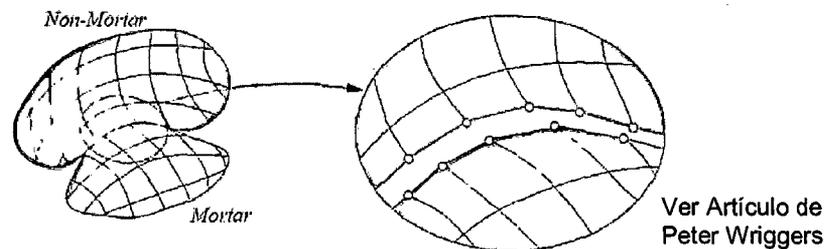


Figura 3.9: Representación del Algoritmo Mortar Element

En esta descripción, los Multiplicadores de Lagrange se asocian a la llamada superficie non-mortar; otra superficie de contacto es llamada superficie mortar. Desde la superficie non-mortar se define el vector normal y tangencial

### 3.3.3.- SPLITTING PINBALLS

Llamado también Algoritmo Pinball (Neal y Belytschko), representa una buena alternativa basada en el empleo de partículas esféricas discretas para la representación de un cuerpo. La idea principal es asociar las esferas rígidas a los elementos de contacto y utilizar las características geométricas y físicas de las mismas en el proceso de búsqueda de las fuerzas de contacto simplificando

de esta manera las operaciones ya que solo está en función a los parámetros geométricos de las esferas.

Las esferas localizadas en la superficie de cada cuerpo son efectivamente candidatos a participar en el contacto y a estos elementos es necesario asociarle una esfera tal como se describe en la figura.

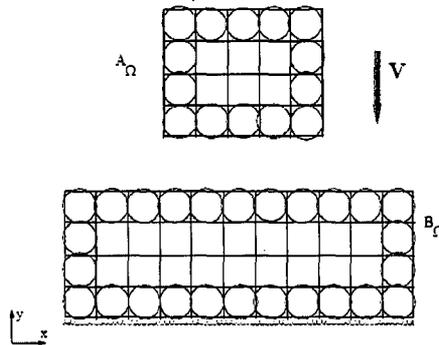


Figura 3.10: Representación del Algoritmo Splitting Pinballs

Para el caso de las fuerzas de fricción, este método aun tiene una deficiencia para el cálculo del mismo.

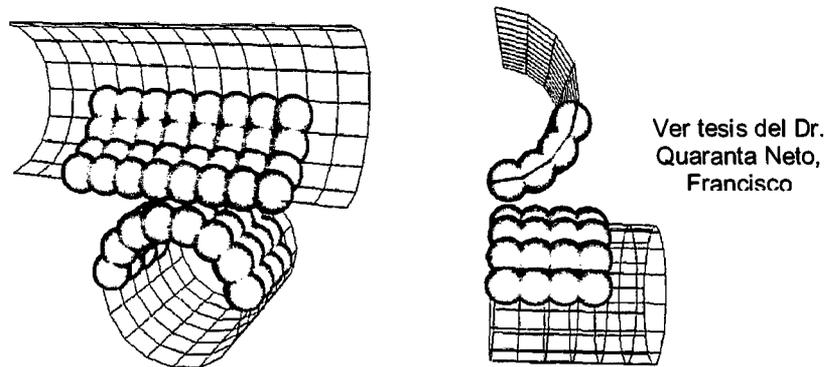


Figura 3.11: Representación del Algoritmo Splitting Pinballs

### 3.4.- OPERADORES DE CONTACTO

Uno de los problemas de contacto es generalmente no saber cuál es la región exacta donde va a ocurrir el problema. Por ello la geometría de contacto es un aspecto importante a ser considerado en los problemas de contacto.

De modo general la actualización de la geometría de contacto, que es función del las cargas y de las posiciones relativas entre cuerpos, se da en dos pasos:

La búsqueda entre los candidatos al contacto, determinando la región de contacto; y el establecimiento de las relaciones cinemáticas locales.

La búsqueda de la región a contactar no es trivial en el caso general ya que un punto de la superficie de un cuerpo puede contactar en cualquier porción de la superficie de otro cuerpo, como también tal punto puede entrar en contacto con una parte de superficie de su propio cuerpo. Así mismo, la búsqueda para la localización correcta del contacto demanda, dependiendo del problema, de un esfuerzo computacional considerable.

Para esta tesis, se va a obviar este procedimiento de localización de la zona y se considerará los puntos a contactar ya que solo se toma los efectos de translación y rotación pequeños descritos en el capítulo de Mecanismo de Contacto.

Es por eso que existen tres tipos de elementos de contacto u operadores de contacto: nudo-nudo, nudo-superficie y superficie-superficie. Dichos operadores serán utilizados según el problema de contacto que se tenga, teniendo en cuenta el costo operacional que ocasionaría, ya que es distinto modelar un elemento usando el operador nudo-nudo que el operador superficie-superficie.

Para el caso de problemas de contacto, los operadores de contacto deberán de tener en cuenta la condición de impenetrabilidad.

A continuación se describe cada tipo de operador.

#### **Operador: Nudo-Nudo**

Es un operador de uso típico en la cual se necesita conocer anticipadamente la localización de contacto, es decir se conoce la posición y dirección de las fuerzas de contacto y ésta no cambia durante la aplicación de la carga externa. Se puede usar este tipo de operador para resolver problemas de tipo superficie-superficie, siempre y cuando las deflexiones de ambas superficies fuesen pequeñas.

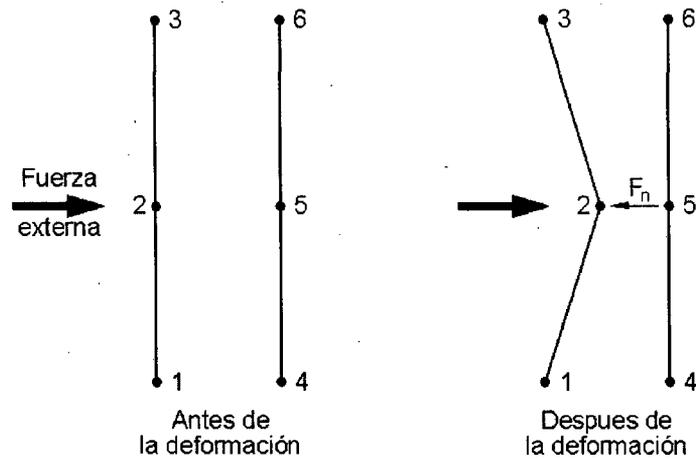


Figura 3.12: Representación del Operador Nudo-Nudo

### Operador Nudo-Superficie

El uso de este operador es de mejor utilidad que el anterior ya que para este tipo de operador no se necesita saber anticipadamente la localización del punto de contacto.

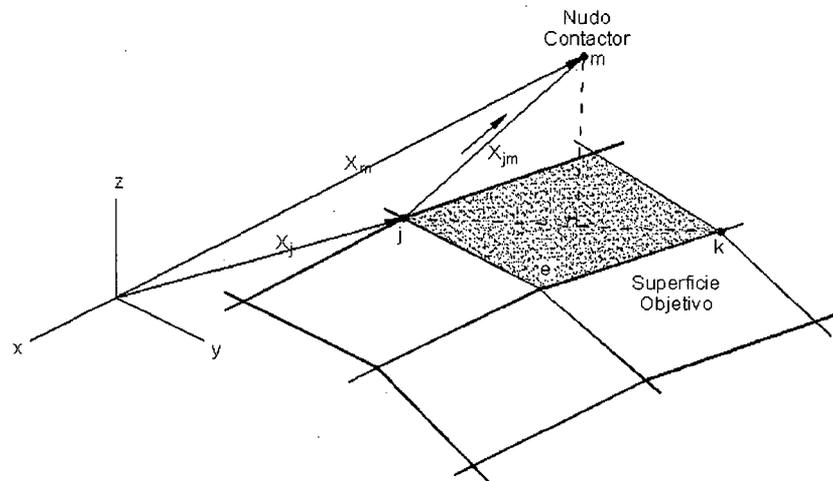


Figura 3.13: Representación del Operador Nudo-Superficie

### Operador Superficie-Superficie

Este operador es utilizado cuando se desconocen ambas regiones de contacto, se utiliza una superficie de contacto (master) y una superficie objetivo (slave) para formar el contacto.

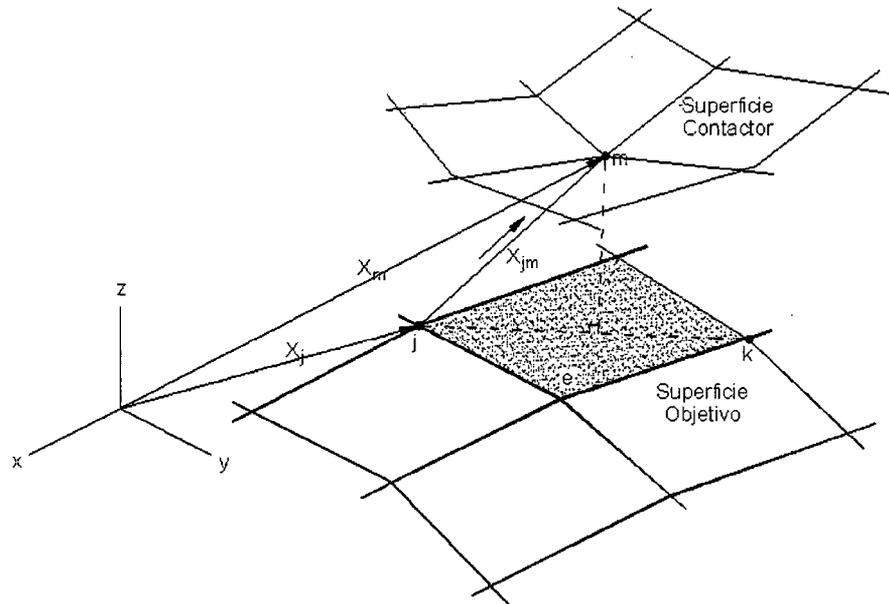


Figura 3.14: Representación del Operador Superficie- Superficie

## CAPITULO IV EJEMPLOS APLICATIVOS

Se presentan tres ejemplos aplicativos las cuales se analizan los desplazamientos y los esfuerzos en las zonas de contacto.

El primer ejemplo está referido al modelo realizado experimentalmente por Hertz, llamado Contacto de Hertz. Para este caso, se presentan los esfuerzos teóricos y los esfuerzos calculados en la presente tesis.

El segundo ejemplo está basado en los efectos que produciría el impacto de un pórtico de un nivel a una placa. Siendo este un ejemplo más didáctico y la cual se puede observar en la realidad en las juntas de cada edificación.

El último ejemplo es la aplicación de dos vigas en voladizo, las cuales está separado por cierta distancia y es aplicada una fuerza externa de manera que una viga impacta sobre otra y se produce el contacto.

Para el cálculo de los desplazamientos y los esfuerzos de estos ejemplos se han utilizado los programas elaborados para la presente tesis y dos programas de cómputo comercial llamados SAP2000 y ANSYS, las cuales de alguna manera han corroborados los resultados.

#### 4.1.- CONTACTO DE HERTZ

Se analiza un cilindro apoyado sobre una superficie rígida con los datos siguientes:

##### Elemento 01 (Cilindro)

Radio	=	500 mm
Módulo Elasticidad (E1)	=	210 GPa
Módulo de Poisson ( $\nu$ 1)	=	0.30
Carga (P)	=	1 kN/mm

##### Elemento 02 (Superficie)

Longitud (L)	=	500 mm
Altura (h)	=	500 mm
Módulo Elasticidad (E2)	=	210 GPa
Módulo de Poisson ( $\nu$ 2)	=	0.30
Separación entre cuerpos (S)	=	0.00 mm

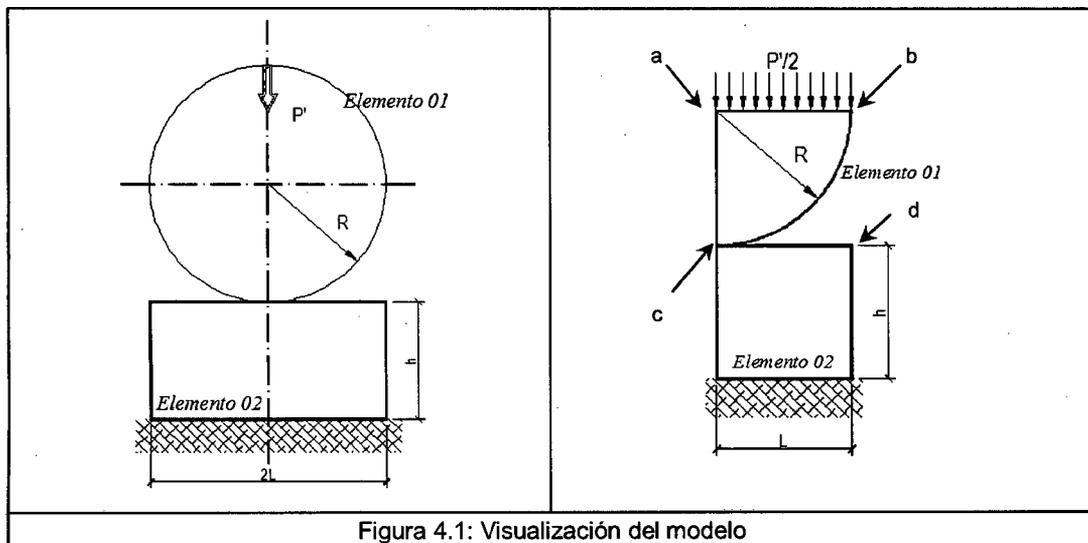


Figura 4.1: Visualización del modelo

Siendo ambos cuerpos simétricos, se ha simplificado de una manera adecuada su geometría, siendo discretizado ambos cuerpos en 2172 elementos triangulares, 1187 nudos y 2 elementos de contacto.

Por las proporciones que guarda la geometría se ha usado el Elemento Finito triangular con Deformación Plana.

Los resultados de los desplazamientos se muestran a continuación:

ID Nudo	$\delta_y$ 3pto Lineal	$\delta_y$ 6pto Lineal	$\delta_y$ 6pto ANSYS	$\delta_y$ 8pto ANSYS
a	-0.0336	-0.0407	-0.0404	-0.0337
b	-0.0387	-0.0459	-0.0455	-0.0386
c	-0.0173	-0.0208	-0.0207	-0.0172
d	+0.0003	+0.0003	+0.0003	+0.0003
Nº Nudos	1187	4542	4679	14762

Los gráficos que se muestran corresponden a los estados de procesamiento con el programa elaborado.

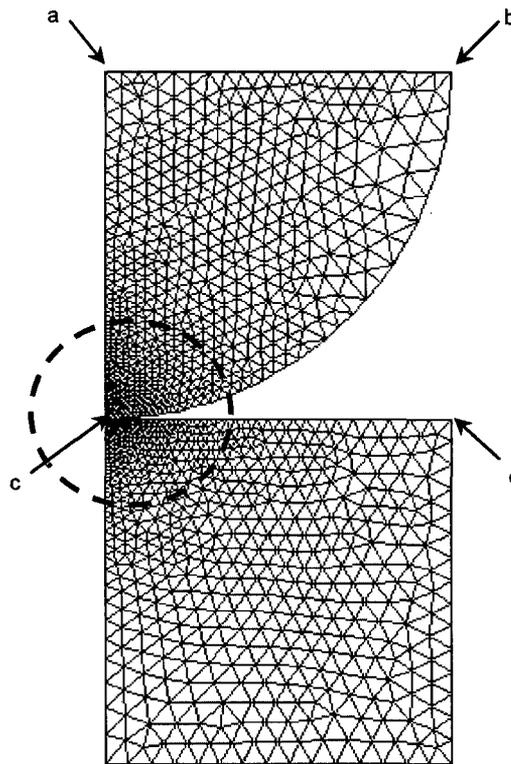
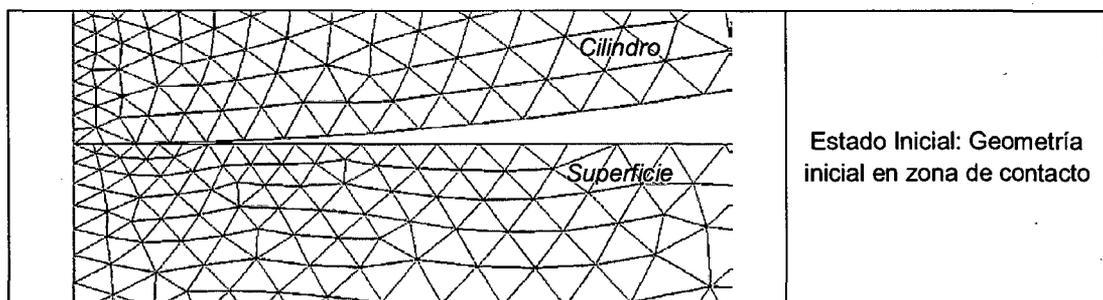
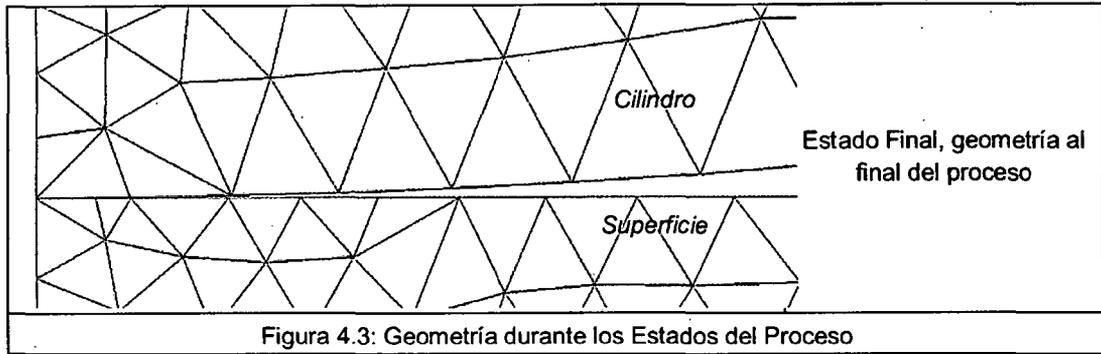


Figura 4.2: Estructura discretizada en 2172 elementos triangulares





Los resultados de los esfuerzos se muestran a continuación:

ID Nudo	Tipo	3pto Lineal	6pto Lineal	6pto ANSYS	8pto ANSYS
a	$\sigma_y$	-1.103	-1.919	-1.565	-0.9958
b	$\sigma_y$	-0.972	-1.783	-1.529	-1.002
c	$\sigma_y$	-159.331	-349.976	-359.56	-312.537
d	$\sigma_y$	-2.525	-0.0030	-0.0034	-0.00001

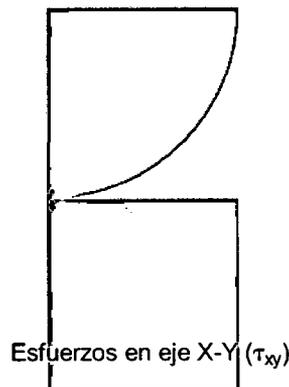
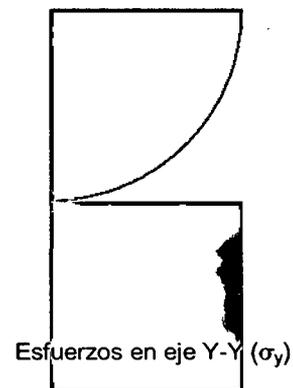
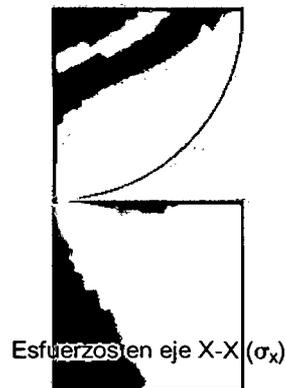


Figura 4.4: Esfuerzos Alisados en cada sentido

Se presentan los esfuerzos en cada dirección en la zona de contacto.

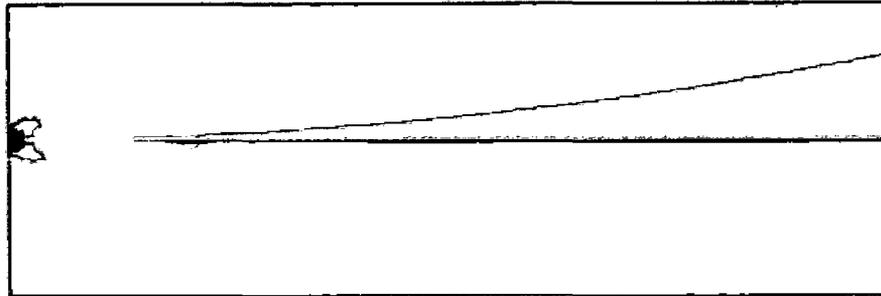


Figura 4.5: Esfuerzos en el sentido X-X en zona de Contacto ( $\sigma_x$ )

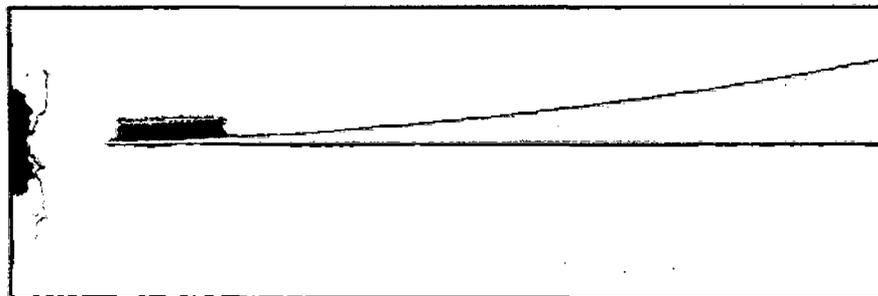


Figura 4.6: Esfuerzos en el sentido Y-Y en zona de Contacto ( $\sigma_y$ )

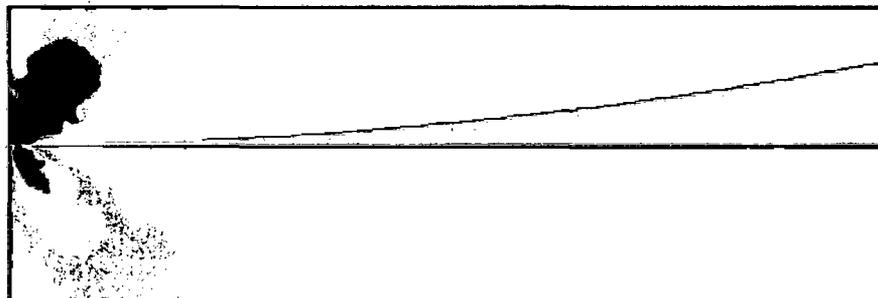


Figura 4.7: Esfuerzos en el sentido X-Y en zona de Contacto ( $\tau_{xy}$ )

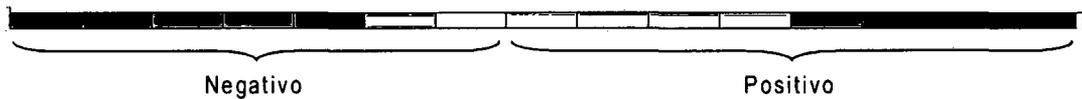


Figura 4.8: Escala de Colores para los Esfuerzos

Según la Teoría del Contacto de Hertz, para el caso de un cilindro sobre un plano se tiene la siguiente expresión en la que describe el esfuerzo en compresión en dicha zona:

$$p(x) = k \cdot \sqrt{b^2 - x^2}$$

$$k = \frac{2 \cdot P'}{\pi \cdot b^2}$$

$$b^2 = \frac{4 \cdot P' \cdot R}{\pi \cdot E}$$

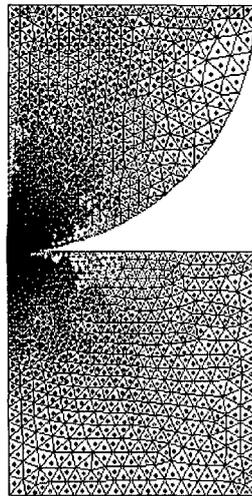
De esta manera el valor del máximo esfuerzo en compresión sería:

$$p(x = 0) = k \cdot b = 365.6 \text{ MPa}$$

Con la finalidad de mejorar los resultados a nivel de esfuerzos, se procederá a realizar dicha mejora usando la metodología de mallas adaptivas. Dicho procedimiento es evaluado con el programa computacional en la versión de Elemento Finito de tres nudos.

Del ejemplo, presentamos los resultados referentes al error, considerando un Error Global Relativo igual a 0.05 (5%):

Norma de la Energética Total	=	2.5067
Norma de la Deformación Total	=	6.8794
Parámetro de Error Total	=	7.2875



Color Rojo = Parámetro local > 1.  
Color Cyan = Parámetro local ≤ 1

Parámetro local mínimo = 0.11  
Parámetro local máximo = 161.62

Siendo 1 el valor óptimo del Parámetro

Figura 4.9: Valor de Parámetros Locales

Para este caso, según los parámetros locales existe mucha diferencia en los esfuerzos calculados en los puntos gaussianos y su respectivo alisado (promedio aritmético), así podemos citar como ejemplo el nudo localizado en la zona de contacto.

Los Esfuerzos ( $\sigma_y$ ) de los elementos que contienen al nudo de contacto correspondiente al cilindro y al plano rígido son:

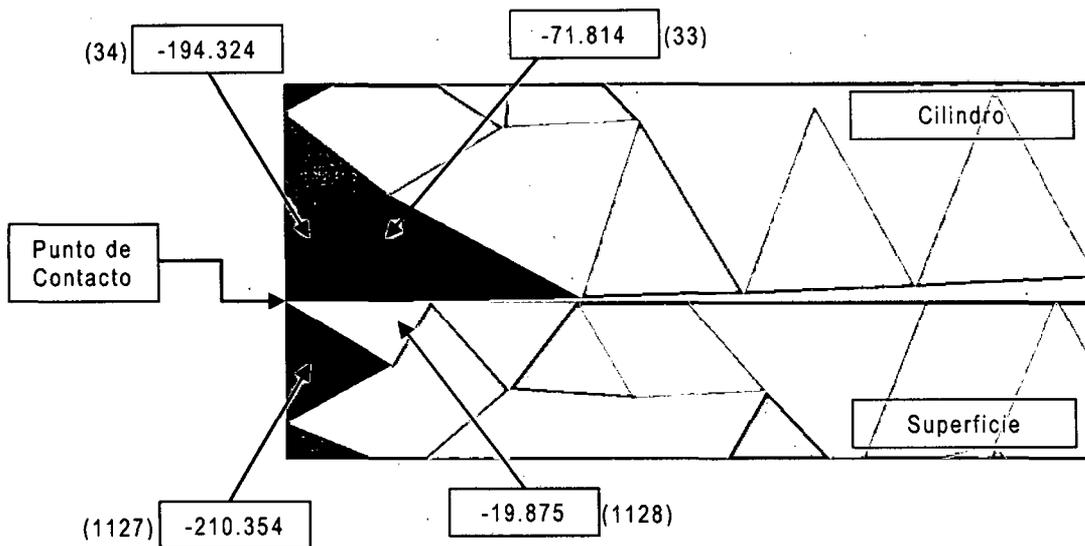


Figura 4.10: Valores de los Esfuerzos en cada elemento

Los valores de los esfuerzos en cada elemento triangular están entre el rango de -210.354 a + 9.284



Figura 4.11: Valores de los Esfuerzos Alisados

Los valores de los esfuerzos en cada nudo están entre el rango de -159.331 a + 0.583.

De la figura 4.10, podemos apreciar la diferencia que existe entre los valores de los esfuerzos en cada elemento triangular adjunto.

Los valores de los parámetros locales para dichos elementos son:

ID Elemento	Parámetro Local
33	94.0698
34	93.6306
1127	161.615
1128	72.0729

De esta manera y conociendo a priori el valor del esfuerzo máximo de compresión, se ha tomado la decisión de discretizar o reducir los tamaños de los elementos triangulares de solo aquellos que estén involucrados dentro de la zona de contacto y que tengan como parámetro local mayor a 1

La nueva malla de los cuerpos en estudio está compuesta por:

1809 nudos, 3353 elementos triangulares y 09 elementos de contacto

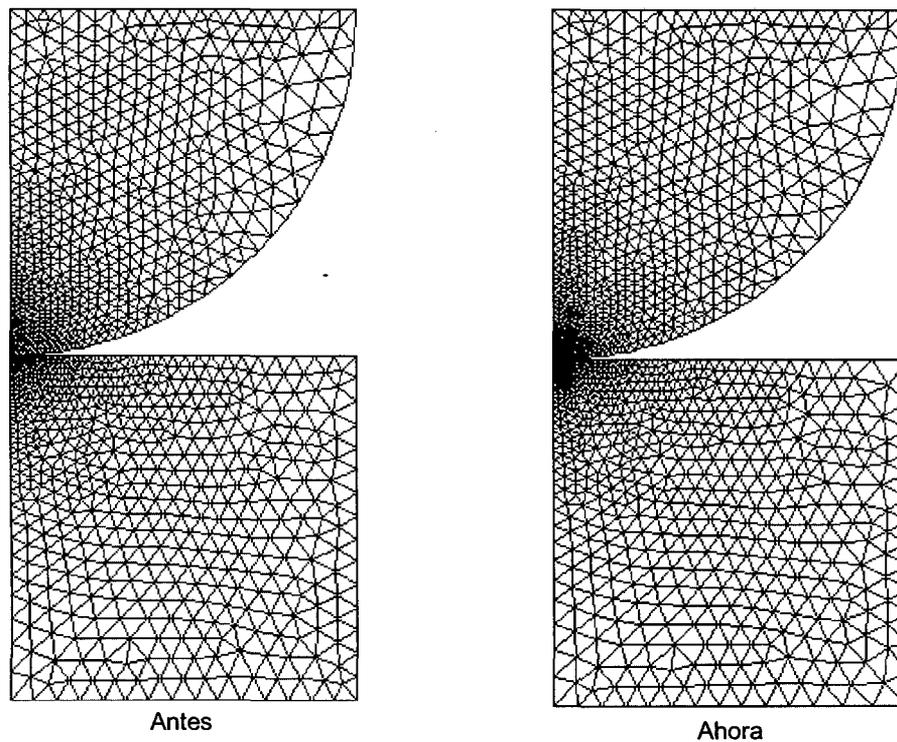


Figura 4.12: Comparativa entre mallas

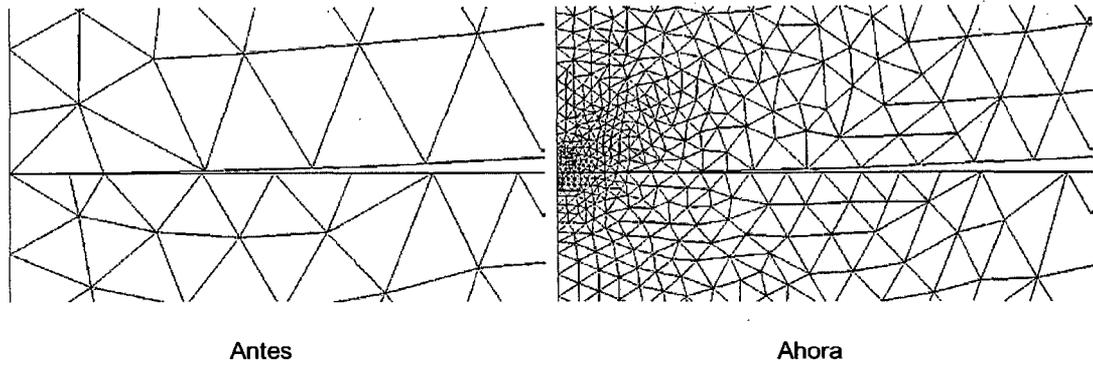


Figura 4.13: Comparativa entre mallas, área localizada

A continuación se presentan los resultados obtenidos:

- Para el caso de los Desplazamientos ( $\delta_y$ ):

ID Nudo	3pto Lineal	3pto Lineal (H-Adap)
a	-0.0336	-0.0340
b	-0.0387	-0.0391
c	-0.0173	-0.0174
d	+0.0003	+0.0003
Nº Nudos	1187	1809

- Para el caso de los Esfuerzos ( $\sigma_y$ ):

ID Nudo	Tipo	3pto Lineal	3pto Lineal (H Adap)
a	$\sigma_y$	-1.103	-1.39
b	$\sigma_y$	-0.972	-1.33
c	$\sigma_y$	-159.331	-367.948
d	$\sigma_y$	-2.525	+0.018

- Para el caso de la Norma:

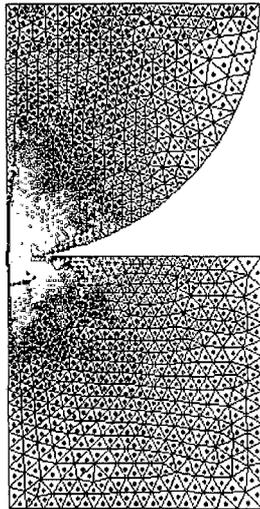


Figura 4.14: Valor de Parámetros Locales

Para un Error Global del 0.05 (5%):

Normas:

De la Energética Total = 0.8856

De la Deformación Total = 7.042

Parámetro de Error Total = 2.5153

Parámetro local mínimo = 0.22

Parámetro local máximo = 8.82

Verificamos ahora los valores de los esfuerzos en la zona de contacto.

- Para el caso de Esfuerzos ( $\sigma_y$ ) calculados en el Punto de Gauss (Un punto de Gauss por cada elemento triangular) son:

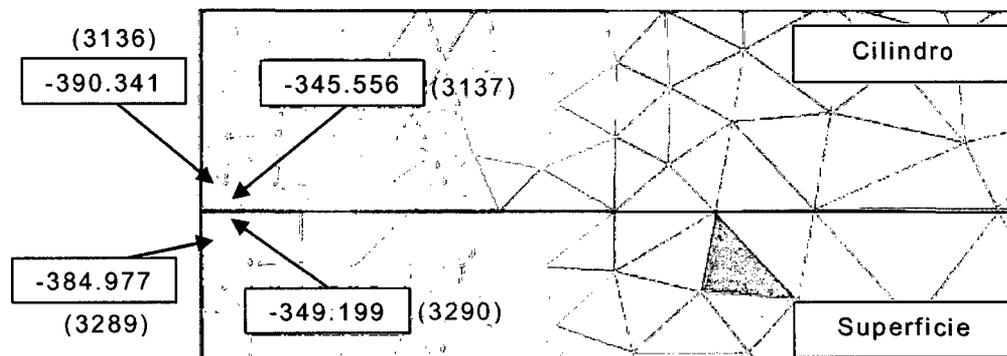


Figura 4.15: Esfuerzos en cada elemento triangular

Los valores de los esfuerzos en cada elemento triangular están entre el rango de -390.341 a +3.509

- Para el caso de Esfuerzos ( $\sigma_y$ ) calculados en los Nudos través de su respectiva extrapolación son:

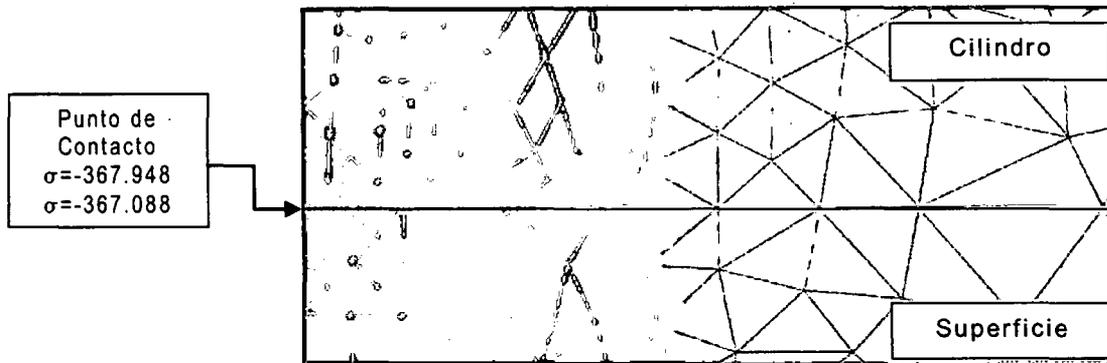


Figura 4.16: Esfuerzos Alisados

Los valores de los esfuerzos en cada nudo están entre el rango de -367.948 a +0.228

Los valores de los parámetros locales para dichos elementos son:

ID Elemento	Parámetro Local
3136	4.1963
3137	5.0862
3289	3.7669
3290	4.8722

Se podría tomar como aceptable estos parámetros de error local, debido que al seguir reduciendo el tamaño del elemento triangular podríamos llegar a tener un error de redondeo computacional. El área del elemento más pequeño en la zona de contacto es 0.022mm<sup>2</sup> y el tamaño promedio en dicha zona es 0.50mm<sup>2</sup>.

Existe otra variable muy importante para el desarrollo de estos problemas a través del algoritmo usado en la presente tesis, se trata del valor de la rigidez que actúa cuando cumple la condición de la inclusión.

Para este ejemplo en particular el procedimiento para obtener el valor de dicha rigidez fue de manera tentativa, pues se fue evaluando progresivamente hasta cumplir la condición de impenetrabilidad y la variación de los esfuerzos con la ayuda de la metodología de la estimación del error.

Se presenta el cuadro indicando los valores de la rigidez de los elementos de contacto y sus resultados en dicha zona.

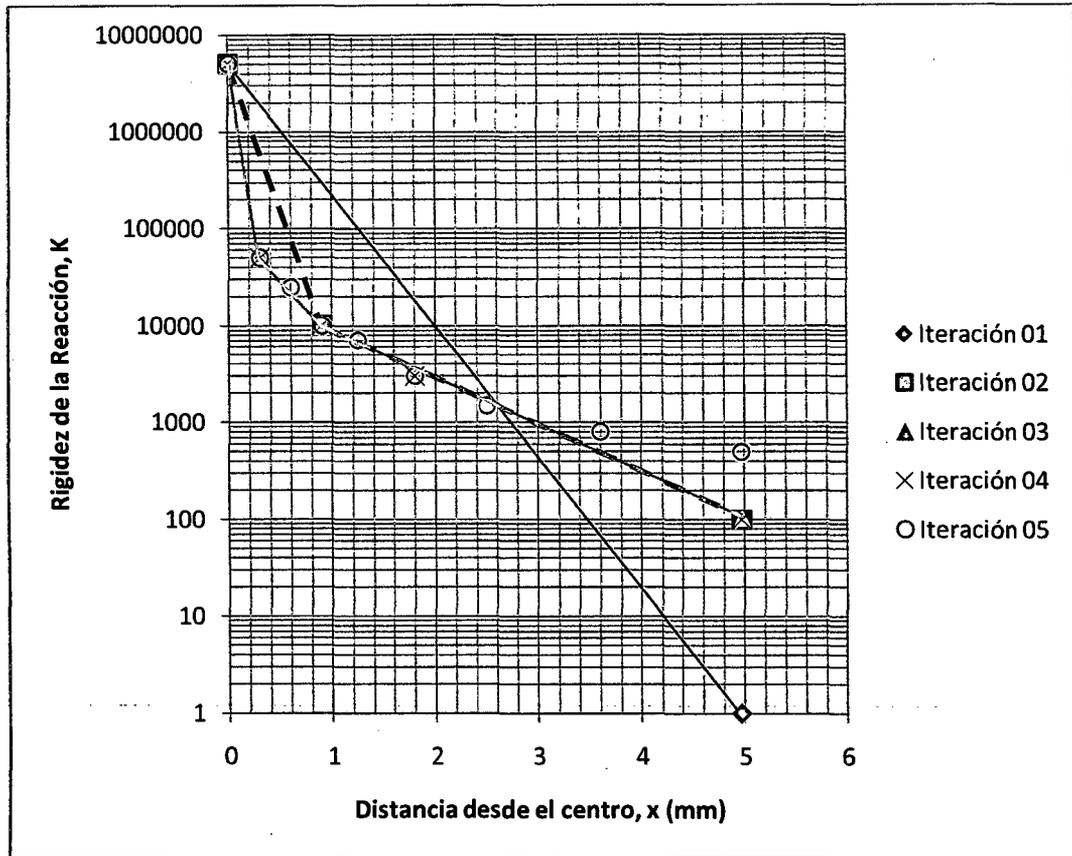
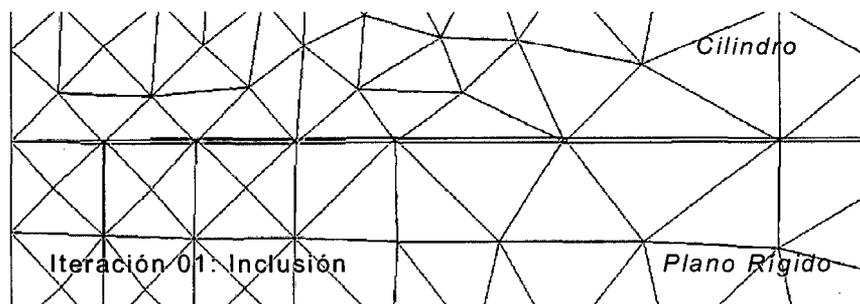


Figura 4.17: Variación de la Rigidez de los Elementos de Contacto

ID Iteración	Parámetro Local Máximo del Error	Parámetro Total del Error	Desplazamiento ( $\delta_y$ )	Esfuerzo Alisado ( $\sigma_y$ )
Iter 01	108.6	5.83	-0.0250	-2439.2
Iter 02	81.0	4.83	-0.0223	-1694.6
Iter 03	37.25	3.42	-0.0205	-1175.0
Iter 04	34.11	3.24	-0.0198	-1012.0
Iter 05	8.8	2.51	-0.0174	-367.95

Mostramos además, la gráfica de la inclusión del cilindro al plano rígido.



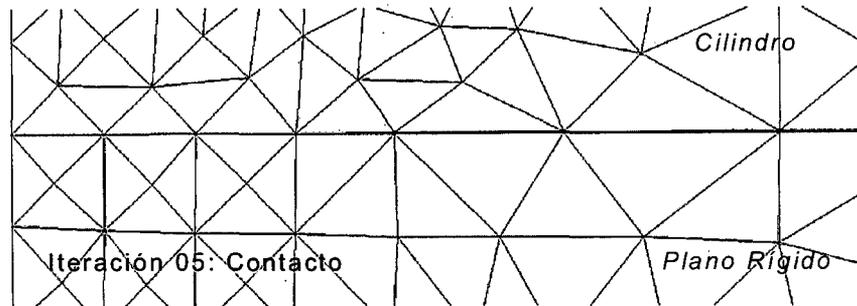


Figura 4.18: Desplazamiento en los Nudos

Finalmente se presenta gráfica de los valores de los esfuerzos alisados (Promedio Directo) para los cuatro casos analizados.

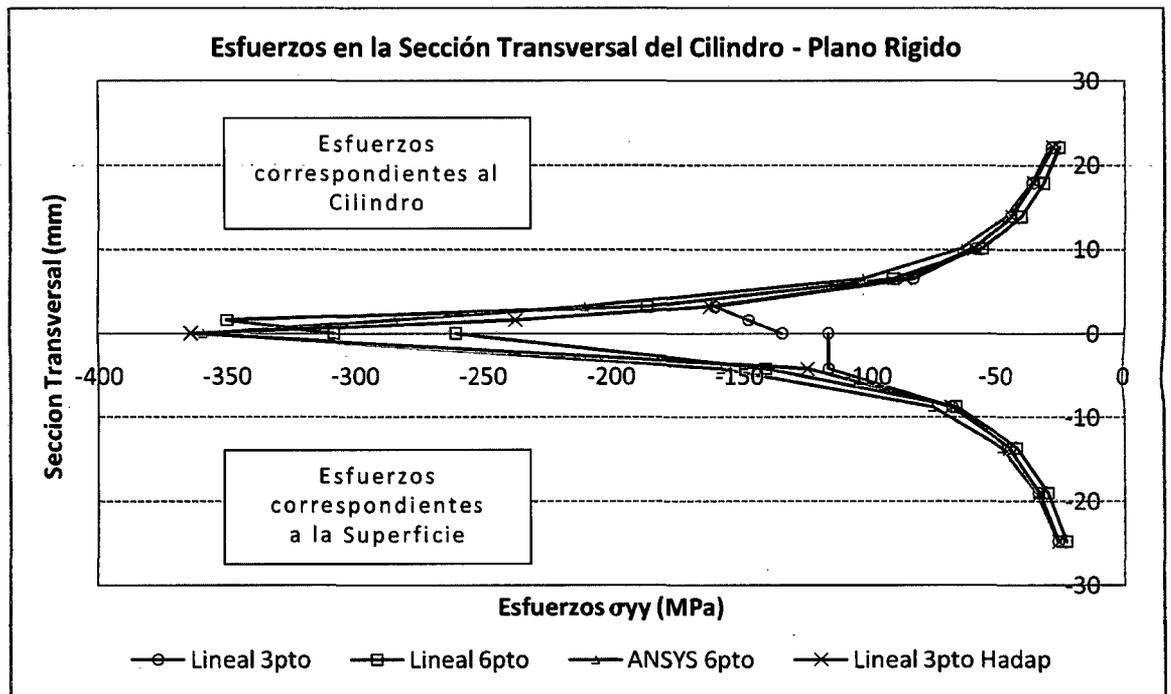


Figura 4.19: Esfuerzos Alisados en Zona de Contacto

## 4.2.- SISTEMA APORTICADO

Se analiza un sistema de pórticos de un nivel con los datos siguientes:

### Elemento 01

Longitud ( <i>Columna, Viga</i> )	=	5.0m
Ancho ( <i>Columna, Viga</i> )	=	0.50m
Espesor ( <i>Columna, Viga</i> )	=	0.30m
Módulo Elasticidad (E)	=	2100000 Ton/m <sup>2</sup>
Módulo de Poisson ( $\nu$ )	=	0.20
Carga (P)	=	30.0 Ton

### Elemento 02

Longitud ( <i>Placa</i> )	=	5.0m
Ancho ( <i>Placa</i> )	=	1.50m
Espesor ( <i>Placa</i> )	=	0.30m
Separación entre cuerpos (S)	=	0.1 cm

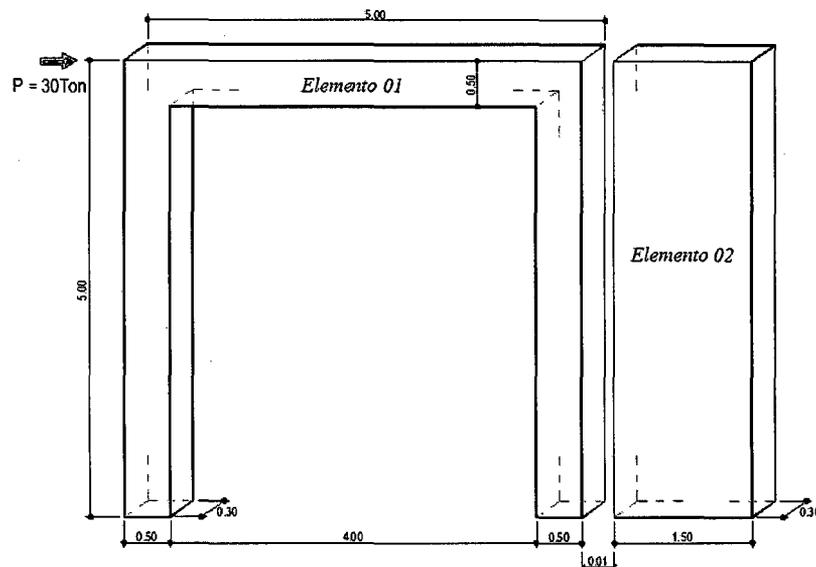


Figura 4.20: Visualización del modelo

Se han discretizado las vigas en 2171 elementos triangulares, 1236 nudos y 01 elemento de contacto.

Por las proporciones que guarda la geometría se ha usado el Elemento Finito triangular con Tensión Plana.

Los resultados de los desplazamientos se muestran a continuación:

ID Nudo	$\delta_x$ 3pto Lineal	$\delta_x$ 6pto Lineal	$\delta_x$ 3pto NoLineal	$\delta_x$ SAP2000
a	0.0139	0.0146	0.0139	0.0144
b	0.0134	0.0139	0.0134	0.0138
c	0.0034	0.0039	0.0034	0.0038
d	0.0033	0.0030	0.0033	0.0036

Los gráficos que se muestran corresponden a los estados de procesamiento con el programa elaborado.

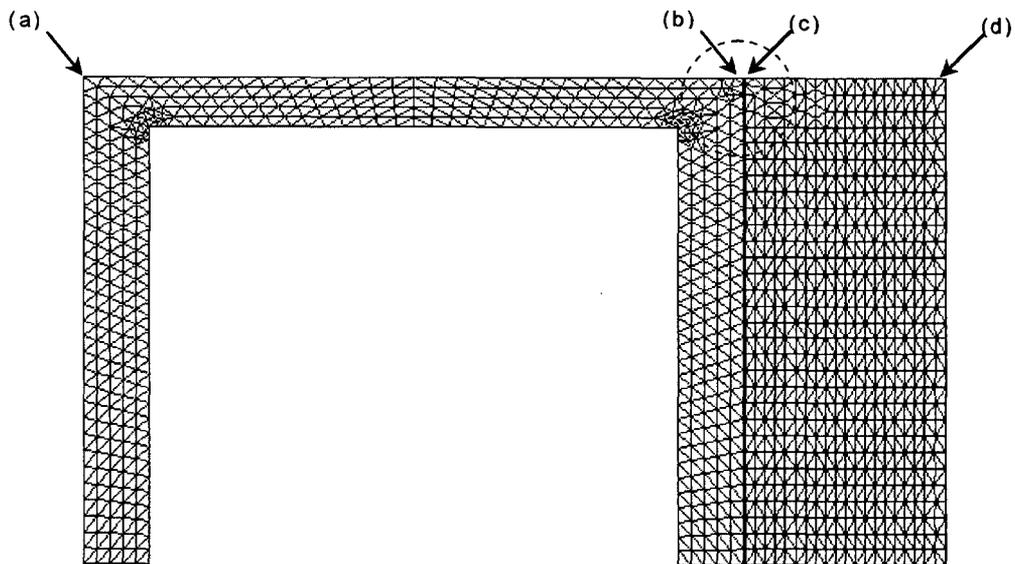
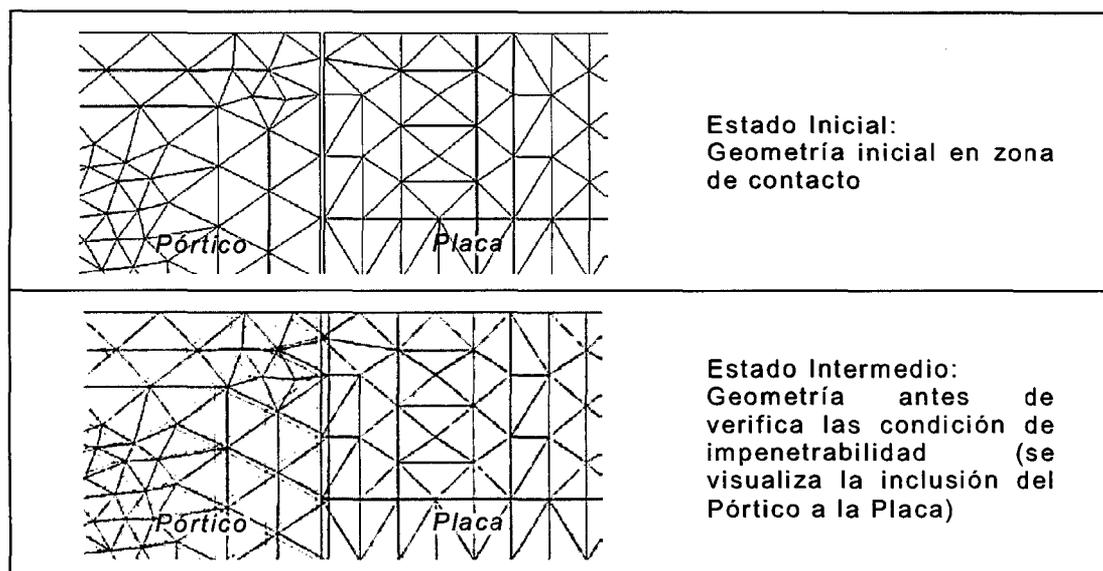


Figura 4.21: Estructura Discretizada



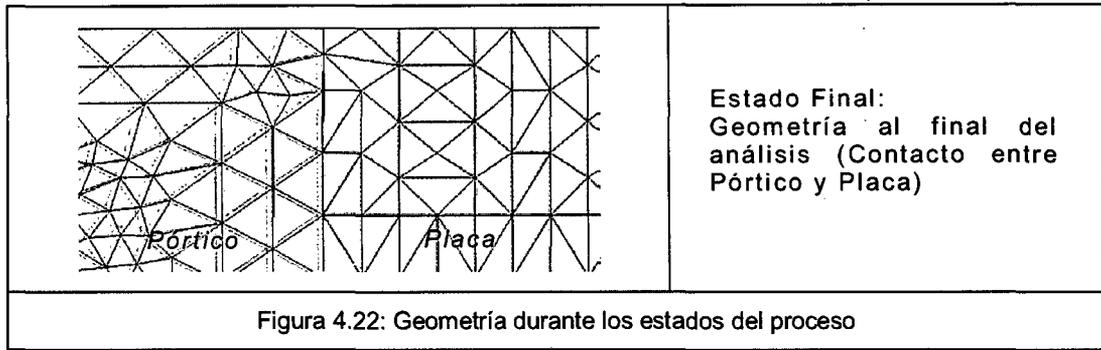


Figura 4.22: Geometría durante los estados del proceso

Los resultados de los esfuerzos se muestran a continuación:

ID Nudo	Tipo	3pto Lineal	6pto Lineal	3pto NoLineal	SAP2000
1x	$\sigma_x$	-979.13	-1173.23	-980.26	-1132.05
2x	$\sigma_x$	1105.04	1388.52	1106.33	1092.59
3x	$\sigma_x$	599.79	762.71	602.56	716.77
4x	$\sigma_x$	-1464.33	-1748.00	-1467.44	-1315.12
5x	$\sigma_x$	-994.24	-3320.31	-988.92	-1240.52
6x	$\sigma_x$	-961.31	-3221.82	-968.76	-1093.96
1y	$\sigma_y$	1227.96	1686.60	1228.34	1628.89
2y	$\sigma_y$	-584.08	-797.02	-586.31	-698.92

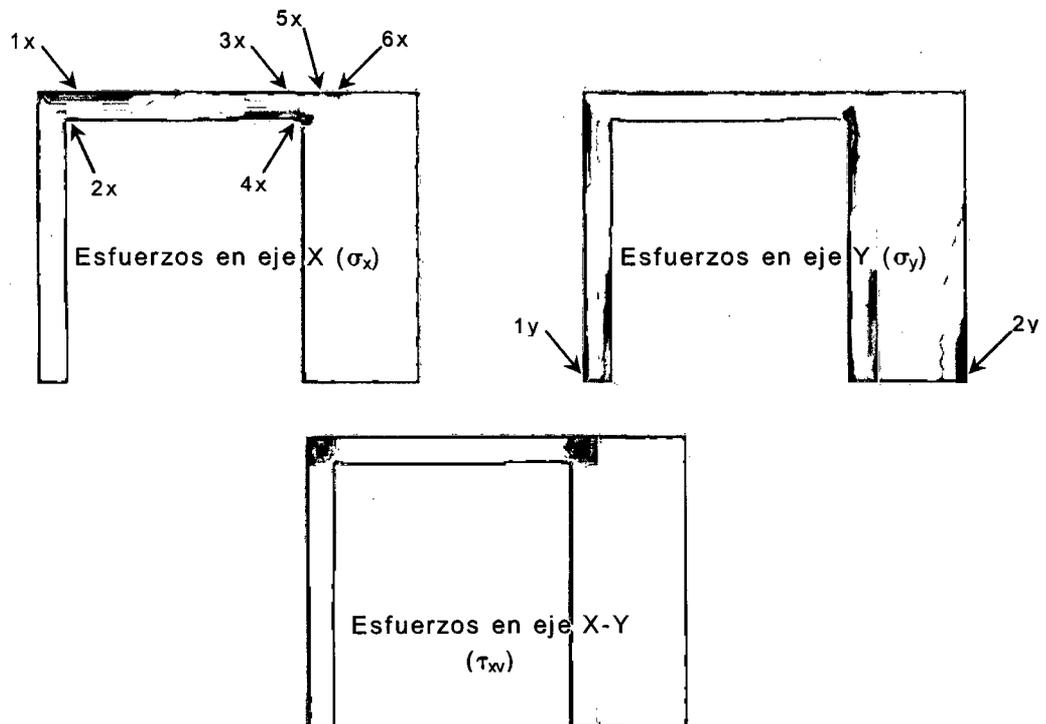
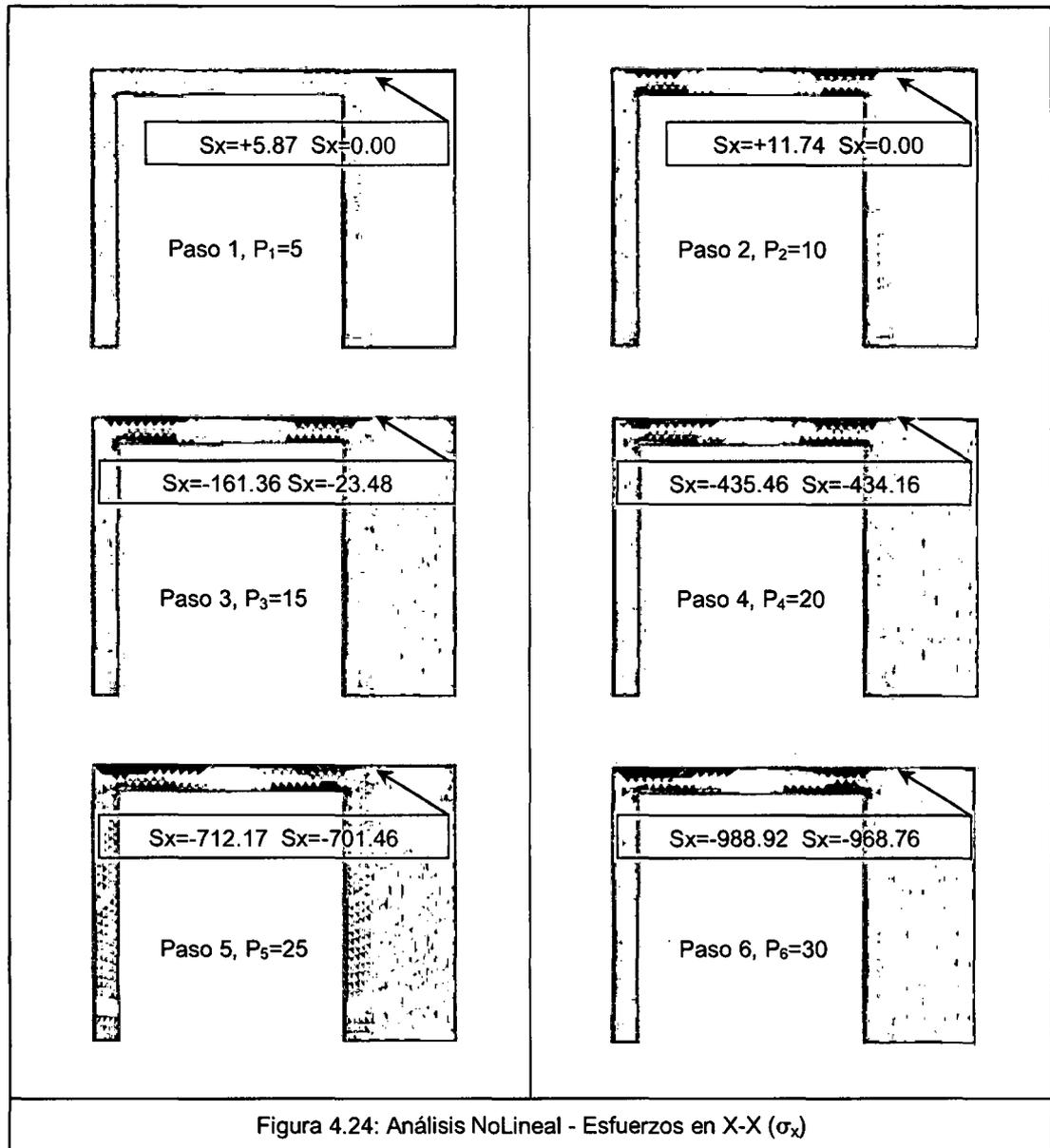
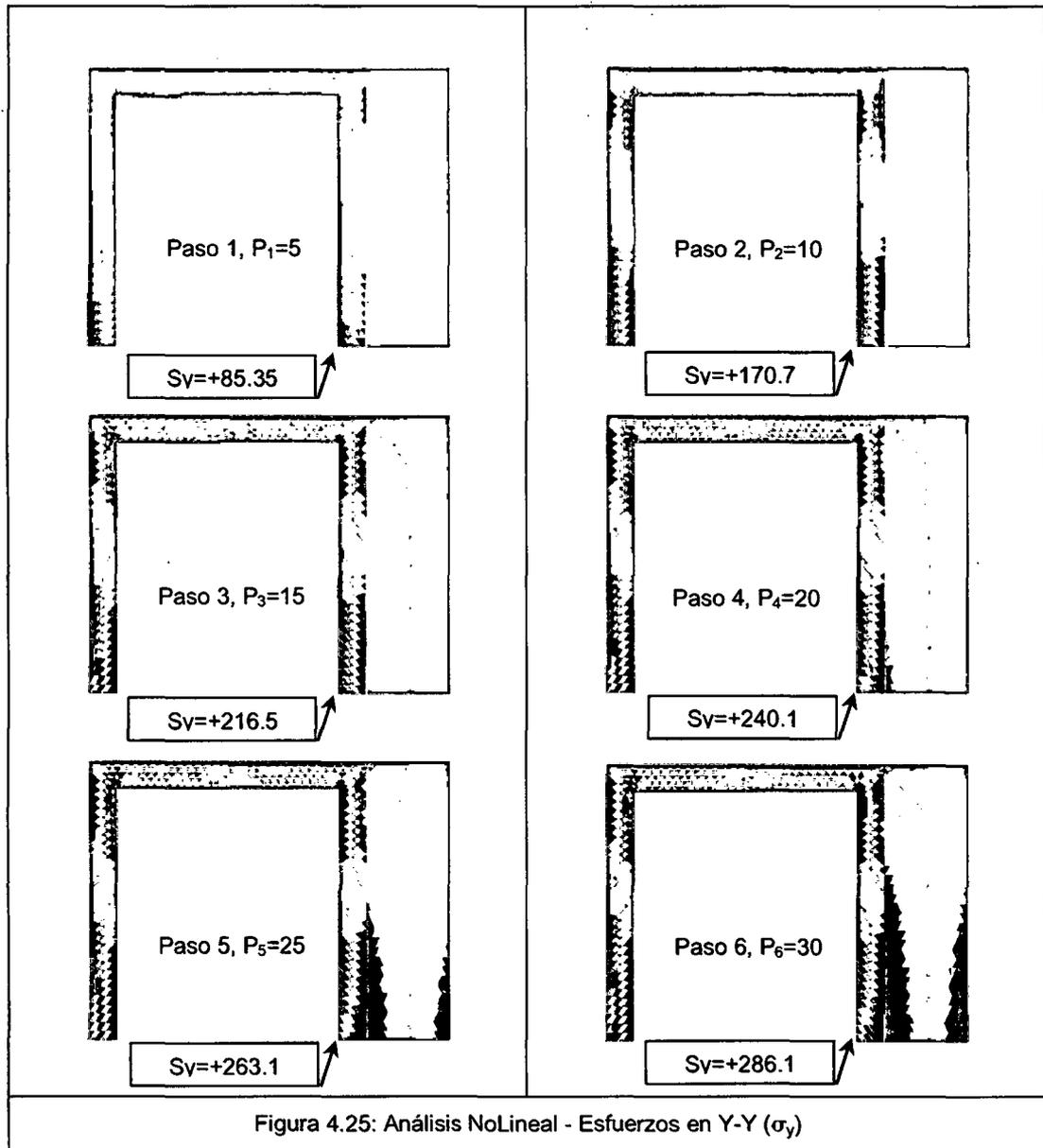


Figura 4.23: Esfuerzos Alisados en cada sentido

Se realizó además, el análisis no lineal geométrico a través de proceso iterativo incremental "paso a paso". Las cuales se muestran los Esfuerzos a medida que se va incrementando las cargas. Para este ejemplo se dividió la carga en 06 partes hasta llegar con la carga total.





Del cuadro de resultados mostrados anteriormente, la comparación entre los esfuerzos calculados mediante un análisis lineal y los calculados mediante un análisis no lineal geométrico no varían mucho, debido a que no existe un excesivo cambio geométrico. Algo importante de este método es que nos visualiza el comportamiento de los cuerpos a medida que se incrementa las cargas.

A continuación se presenta los Esfuerzos ( $\sigma_x$ ) en la zona de contacto, calculados en el Punto de Gauss (Un punto de Gauss por cada elemento triangular) y los Esfuerzos Alisados mediante el método de Promedio Directo:

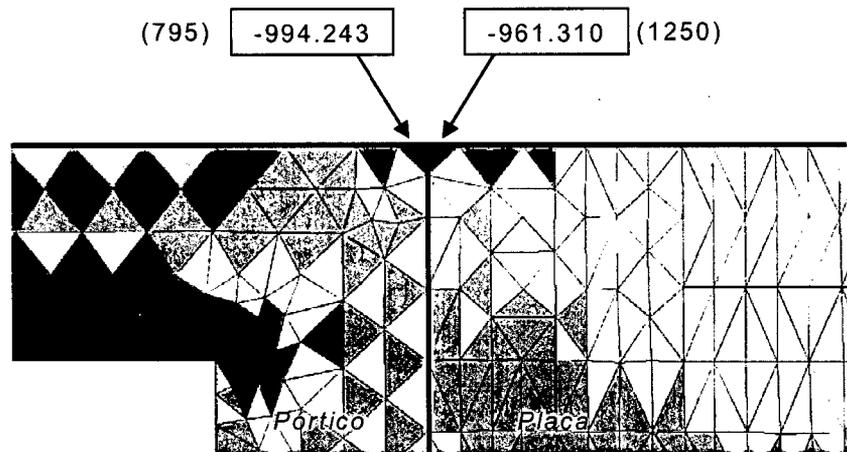


Figura 4.26: Esfuerzos en cada elemento triangular

Los valores de los esfuerzos en cada elemento triangular están entre el rango de -1962.18 a +1557.73

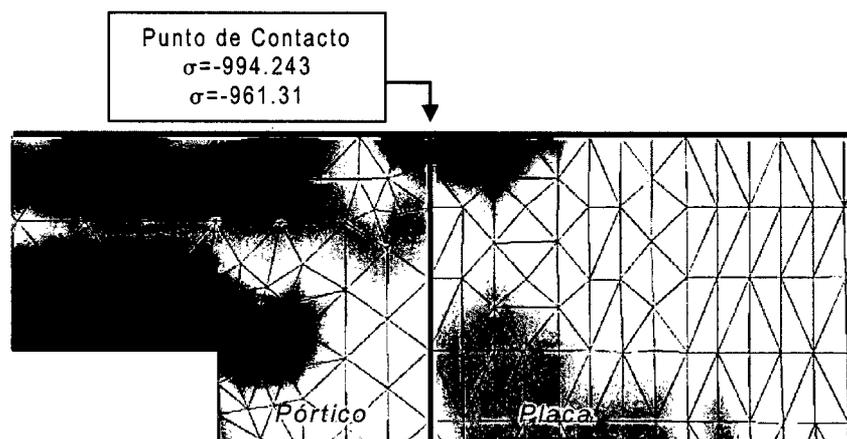


Figura 4.27: Esfuerzos Alisados

Los valores de los esfuerzos en cada nudo están entre el rango de -1464.33 a +1105.04

Los valores de los parámetros locales para dichos elementos son:

ID Elemento	Parámetro Local
795	18.211
1250	19.933

Con respecto a la Normas:

Para un Error Global Relativo igual a 0.05 (5%):

Norma de la Energética Total	=	0.35105
Norma de la Deformación Total	=	0.83134
Parámetro de Error Total	=	8.44535

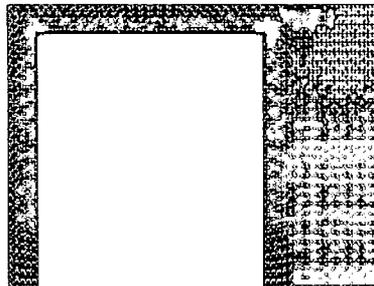


Figura 4.28: Valor de Parámetros Locales

Parámetro Local:

Color Cyan  $\leq 1$

Color Rojo [1, 17]

Color Negro  $>17$

Parámetro local mínimo = 0.14

Parámetro local máximo = 33.12

Siendo 1 el valor óptimo del Parámetro

En este ejemplo existen dos detalles importantes. La primera es que solo se ha usado un elemento de contacto (01 gap) asumiendo inicialmente que el contacto ocurre en un solo punto, lo cual puede ser falso y como consecuencia se tendría que redistribuir el esfuerzo puntual calculado a lo largo de una longitud de contacto que se calculará cuando se actualice la nueva malla.

El segundo detalle importante ocurre en la geometría del elemento triangular en el supuesto punto de contacto. El cambio de orientación de dos triángulos adyacentes no debe de influenciar mucho en los resultados, si esto ocurre es porque no se tiene una buena distribución de los elementos o los nuevos triángulos cambian sustancialmente en su geometría (área y ángulos internos).

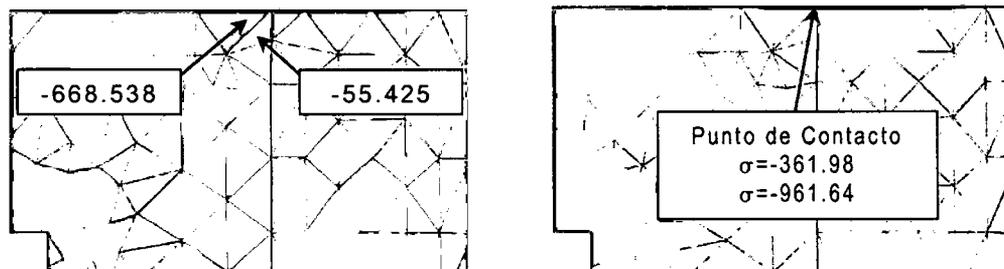


Figura 4.29: Cambio de Orientación, Esfuerzos

De los resultados presentados anteriormente en referencia a la norma, se procederá a discretizar o reducir los tamaños de los elementos triangulares que están involucrados en la zona de contacto con la finalidad de mejorar los resultados de los esfuerzos.

La nueva malla de los cuerpos en estudio está compuesta por:  
2216 nudos, 3964 elementos triangulares y 07 elementos de contacto

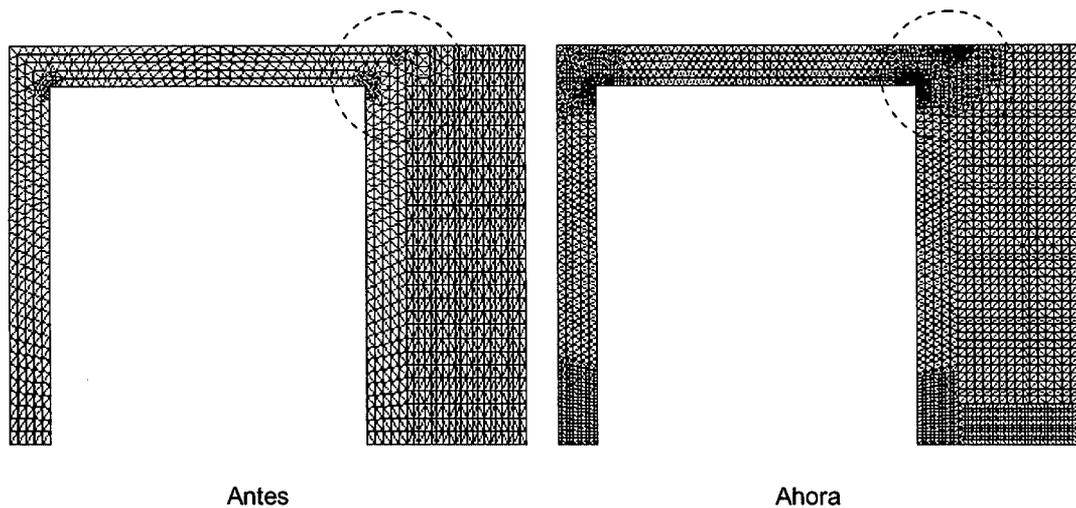


Figura 4.30: Comparativa entre mallas

Amplificando la zona de contacto para tener una mejor visualización de la geometría.

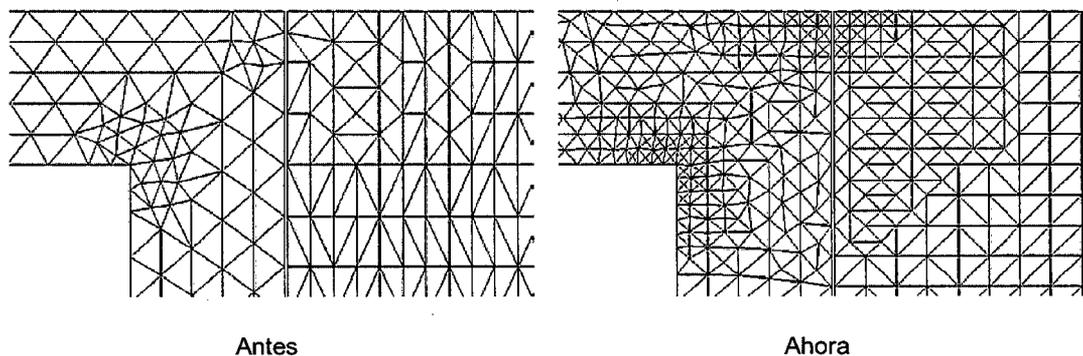


Figura 4.31: Comparativa entre mallas, área localizada

A continuación se presentan los resultados obtenidos:

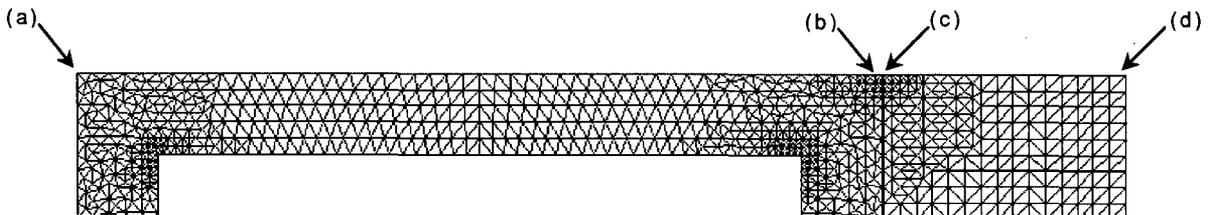
- Para el caso de la Longitud de Contacto ( $l_c$ ):

Para obtener este valor, se procedió usando solo un elemento de contacto y cambiando de manera iterativa el valor de la rigidez del elemento de contacto hasta llegar a cumplir la condición de impenetrabilidad.

N° ElemCont	Longitud de Contacto (m)										
	0	0.05	0.1	0.15	0.2	0.3	0.4	0.5	0.6	0.7	0.8
1	0	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0002	0.0001	0	-0.0001
7	0	0	0	0	0	0	0	-0.0001	-0.0002	-0.0003	-0.0004

De la tabla adjunta, se puede visualizar que usando un (01) elemento de contacto, la longitud de contacto llega medir hasta 0.70m, a partir de allí la longitud de penetración se hace nula ya que ambos cuerpos comienzan a separarse. A medida que se va activando los elementos de contacto con su respectiva rigidez, se va reduciendo la longitud de contacto así como también se va reduciendo la longitud de penetración. La longitud de contacto final estaría en el rango de 0.40m

- Para el caso de los Desplazamientos ( $\delta_x$ ):



ID Nudo	$\delta_x$ 3pto Lineal	$\delta_x$ 3pto Lineal (H Adap)
a	0.0139	0.0140
b	0.0134	0.0136
c	0.0034	0.0036
d	0.0033	0.0035

- Para el caso de los Esfuerzos ( $\sigma_x$ ):

ID Nudo	Tipo	3pto Lineal	3pto Lineal (H Adap)
1x	$\sigma_x$	-979.13	-1,073.20
2x	$\sigma_x$	1,105.04	1,353.00
3x	$\sigma_x$	599.79	709.26
4x	$\sigma_x$	-1,464.33	-1,626.56

5x	$\sigma_x$	-994.24	-448.27
6x	$\sigma_x$	-961.31	-435.91
1y	$\sigma_y$	1,227.96	1,431.16
2y	$\sigma_y$	-584.08	-669.525

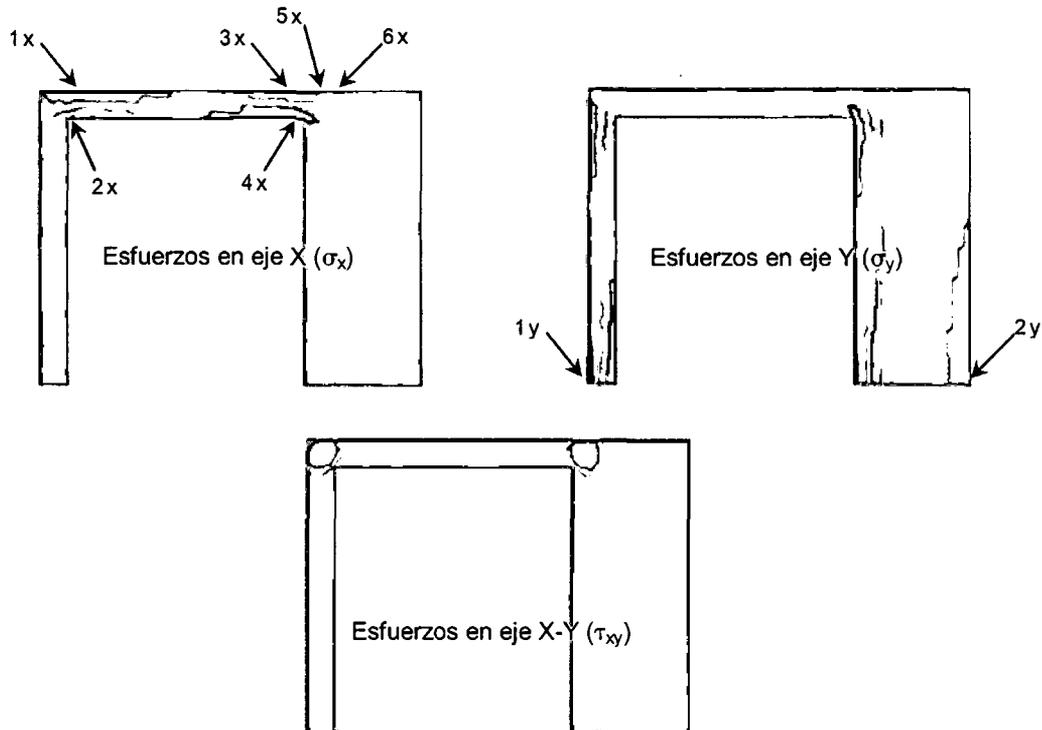


Figura 4.32: Esfuerzos Alisados en cada sentido

- Para el caso de la Norma:

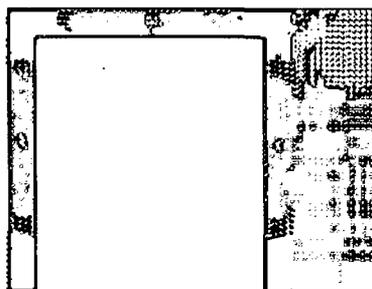


Figura 4.33:  
Valor de Parámetros Locales

Para un Error Global del 0.05 (5%):

Normas:

De la Energética Total = 0.2354

De la Deformación Total = 0.8568

Parámetro de Error Total = 5.4945

Parámetro local mínimo = 0.235

Parámetro local máximo = 22.207

Verificamos ahora los valores de los esfuerzos en la zona de contacto:

- Para el caso de Esfuerzos ( $\sigma_y$ ) calculados en el Punto de Gauss (Un punto de Gauss por cada elemento triangular) son:

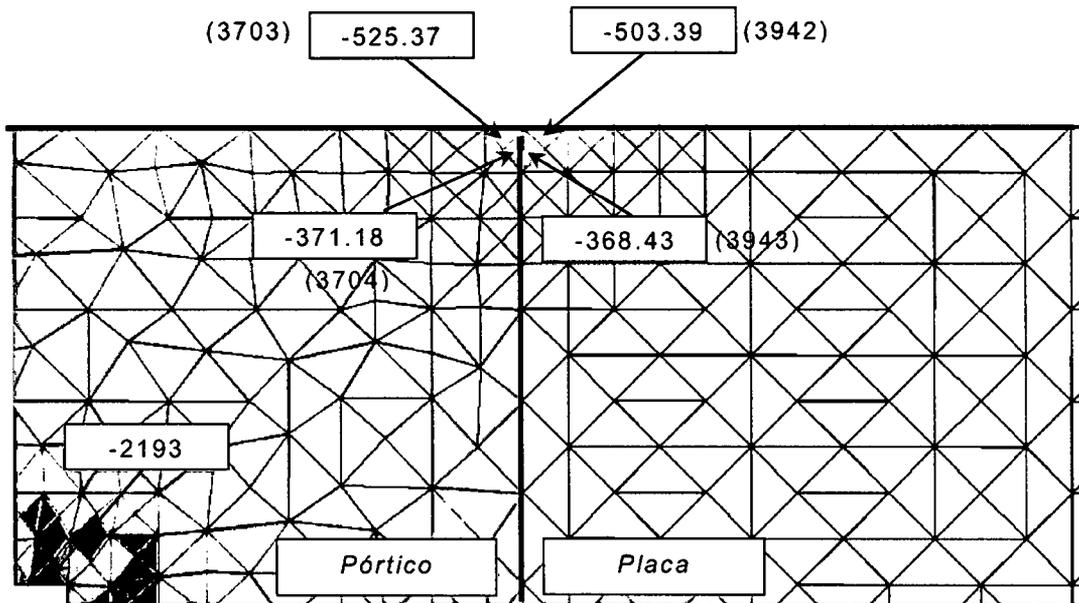


Figura 4.34: Esfuerzos en cada elemento triangular

Los valores de los esfuerzos en cada elemento triangular están entre el rango de -2193 a +1946.5

- Para el caso de Esfuerzos ( $\sigma_y$ ) calculados en los Nudos través de su respectiva extrapolación son:

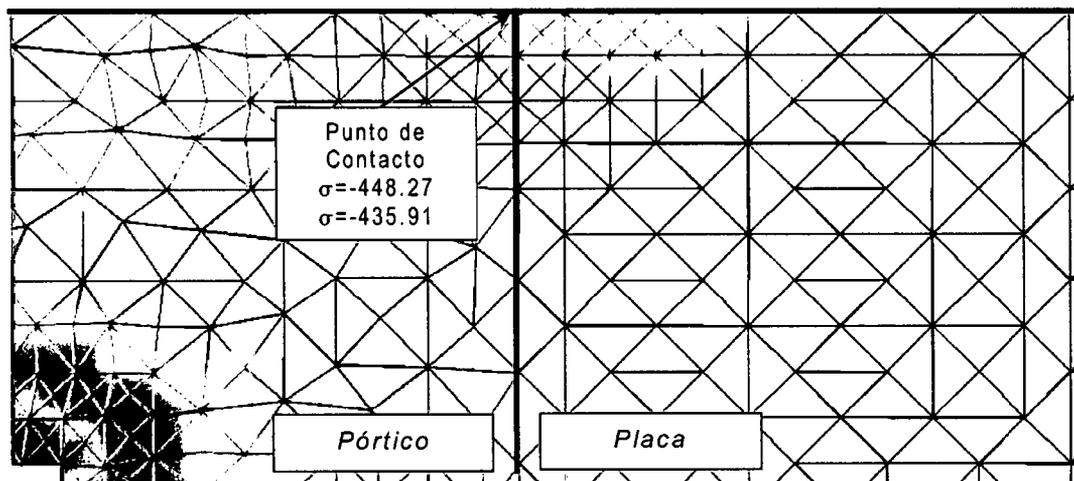


Figura 4.35: Esfuerzos Alisados

Los valores de los esfuerzos en cada nudo están entre el rango de -1626.6 a +1353

Los valores de los parámetros locales para dichos elementos son:

ID Elemento	Parámetro Local
3703	3.08271
3704	2.24475
3942	2.93276
3943	2.44018

Se presenta el cuadro indicando los valores de la rigidez de los elementos de contacto y sus resultados en dicha zona.

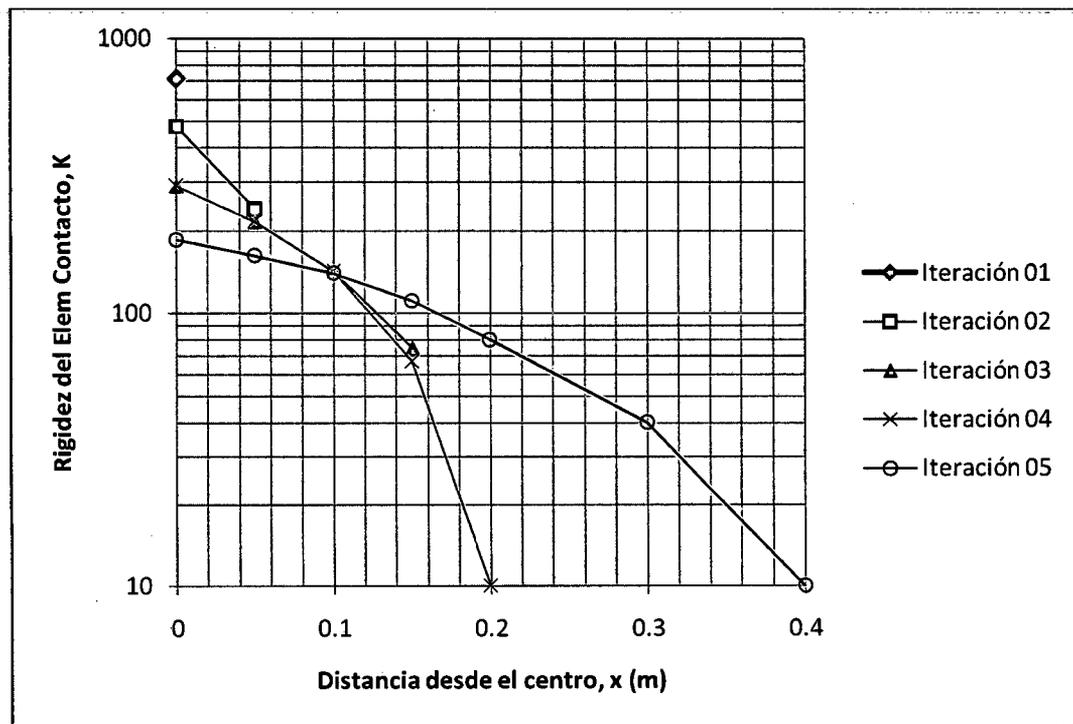


Figura 4.36: Variación de la Rigidez de los Elementos de Contacto

Finalmente se presenta gráfica de los valores de los esfuerzos alisados (Promedio Aritmético) para los tres casos analizados en la sección transversal del pórtico-placa ubicada sobre los puntos (5x) y (6x)

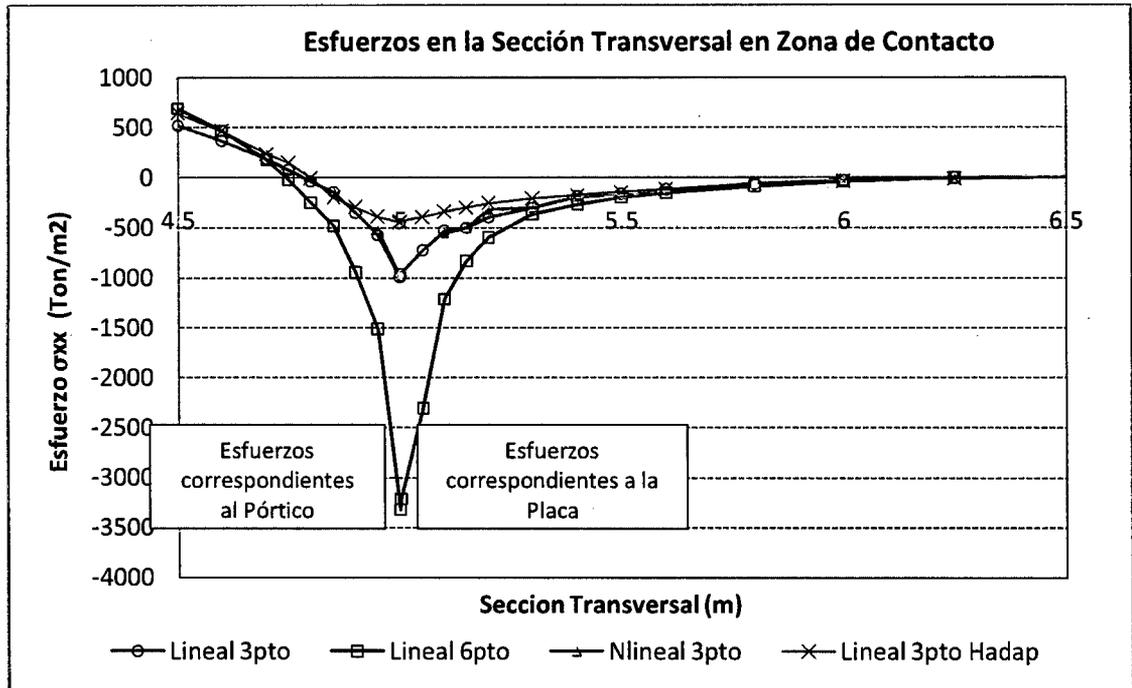


Figura 4.37: Esfuerzos en Zona de Contacto

### 4.3.- VIGAS EN VOLADIZO

Se analiza dos vigas en voladizo con los datos siguientes:

Longitud ( $L_1, L_2$ )	=	3.0m
Ancho de viga ( $b_1, b_2$ )	=	0.20m
Altura de viga ( $h_1, h_2$ )	=	0.40m
Módulo Elasticidad (E)	=	2100000 Ton/m <sup>2</sup>
Módulo de Poisson ( $\nu$ )	=	0.20
Carga (P)	=	3.0 Ton
Separación entre cuerpos (S)	=	0.5 cm

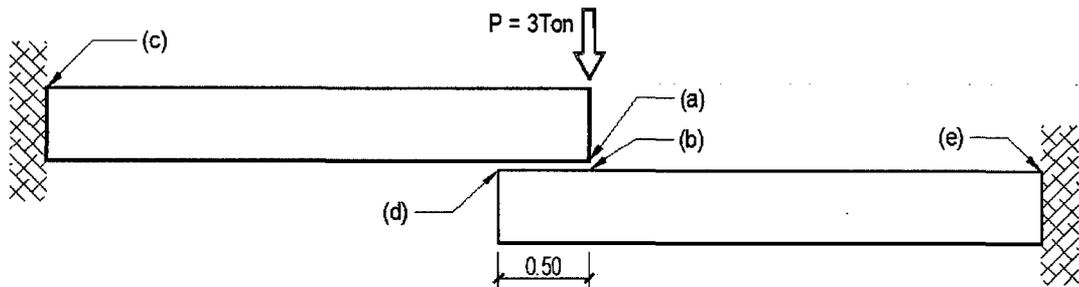


Figura 4.38: Visualización del modelo

Se han discretizado las vigas en 2156 elementos triangulares, 1216 nudos y 01 elemento de contacto.

Los resultados de los desplazamientos se muestran a continuación:

ID Nudo	$\delta_y$ 3pto Lineal	$\delta_y$ 6pto Lineal	$\delta_y$ 3pto NoLineal	$\delta_y$ SAP2000
a	-0.0076	-0.0077	-0.0075	-0.0077
b	-0.0025	-0.0026	-0.0026	-0.0026
d	-0.0032	-0.0034	-0.0032	-0.0034

Los gráficos que se muestran corresponden a los estados de procesamiento de los resultados con el programa elaborado.

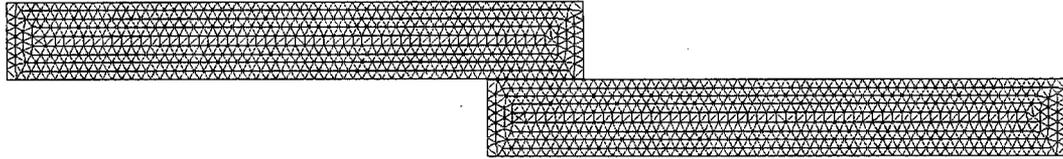


Figura 4.39: Estructura discretizada en 2156 elementos triangulares

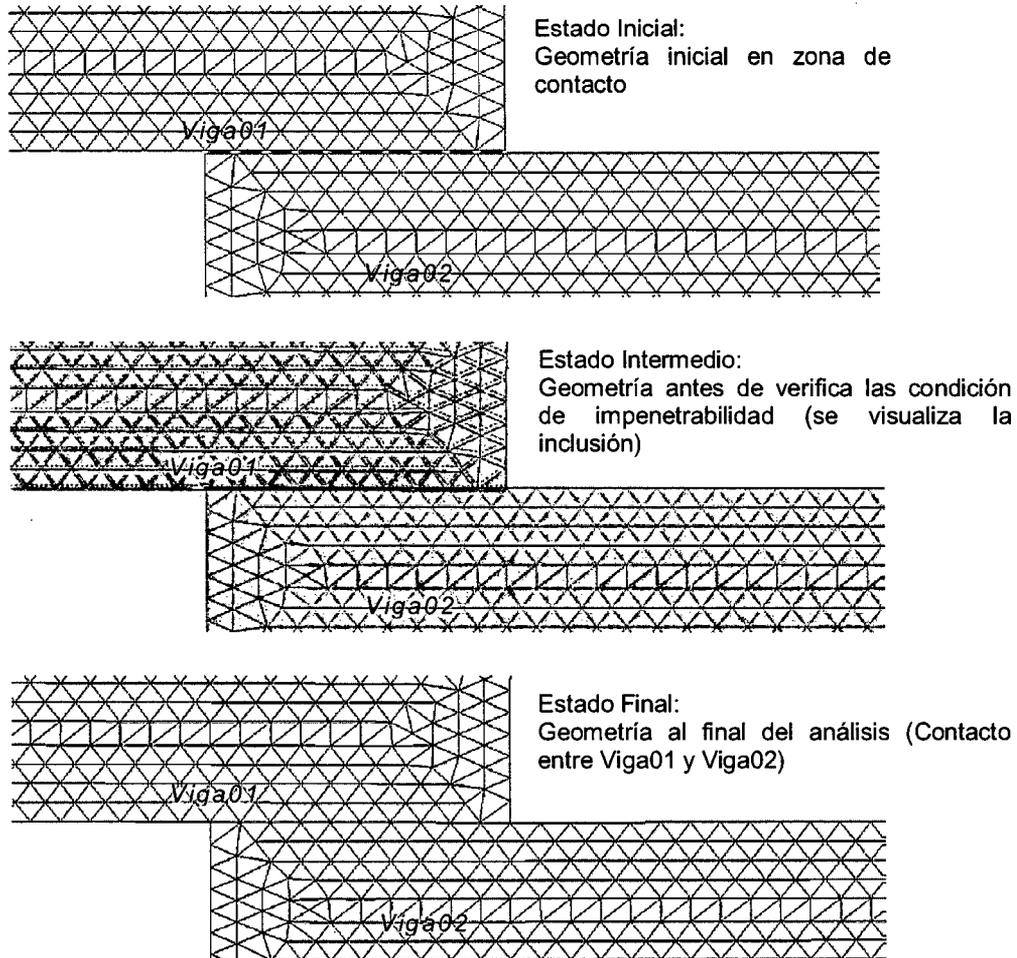


Figura 4.40: Geometría durante los estados del proceso

Los resultados de los esfuerzos se muestran a continuación:

ID Nudo	Tipo	3pto Lineal	6pto Lineal	3pto NoLineal	SAP2000
a	$\sigma_y$	-128.25	-366.324	-129.08	-189.97
b	$\sigma_y$	-64.13	-195.436	-65.91	-111.88
c	$\sigma_x$	1068.70	1228.13	1054.33	1083.45
e	$\sigma_x$	498.87	606.05	507.81	527.45

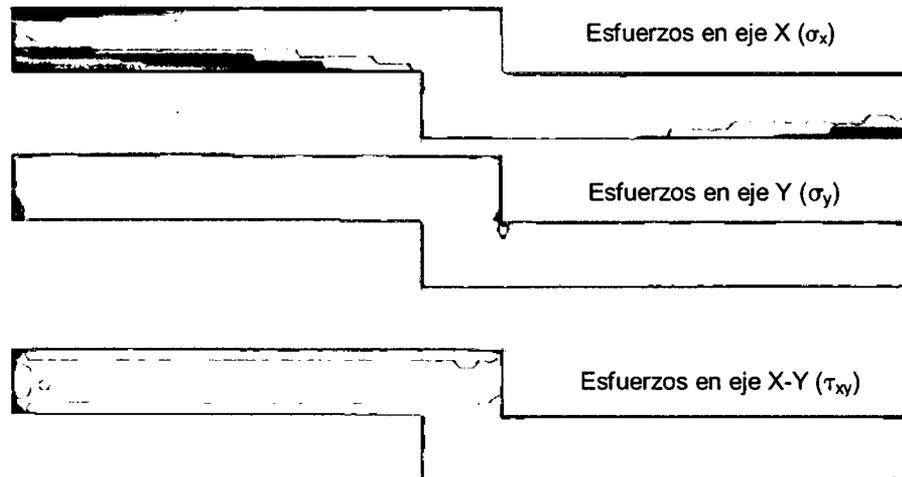
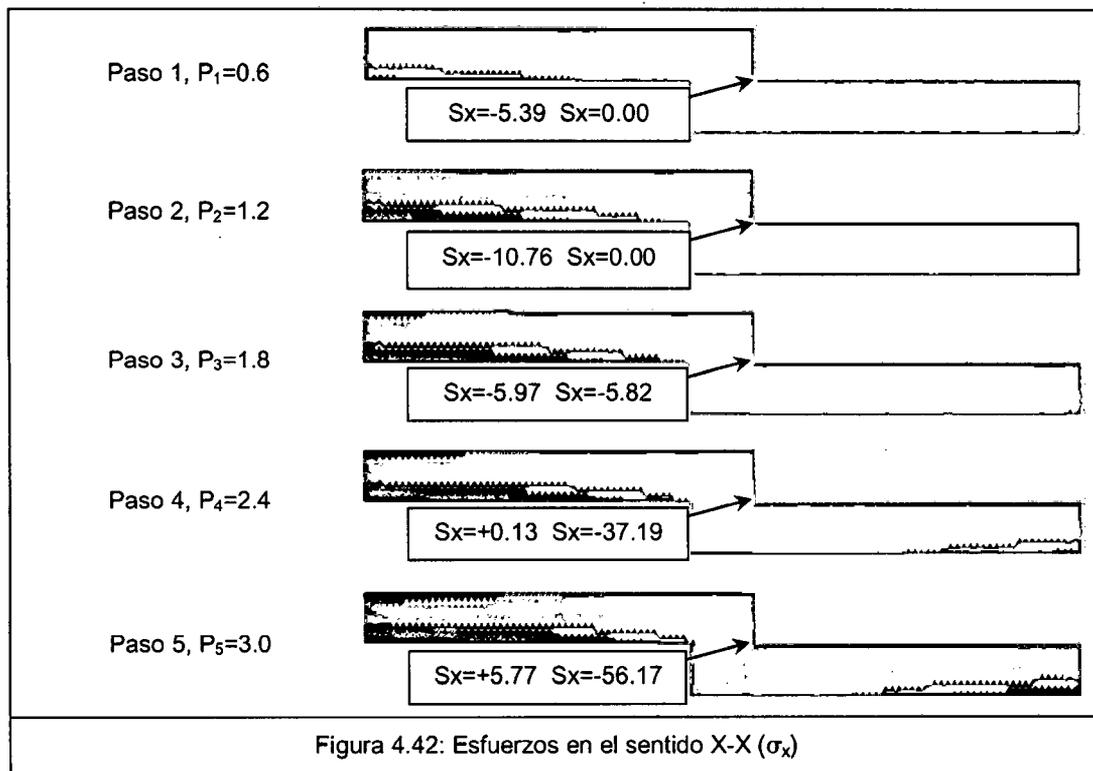
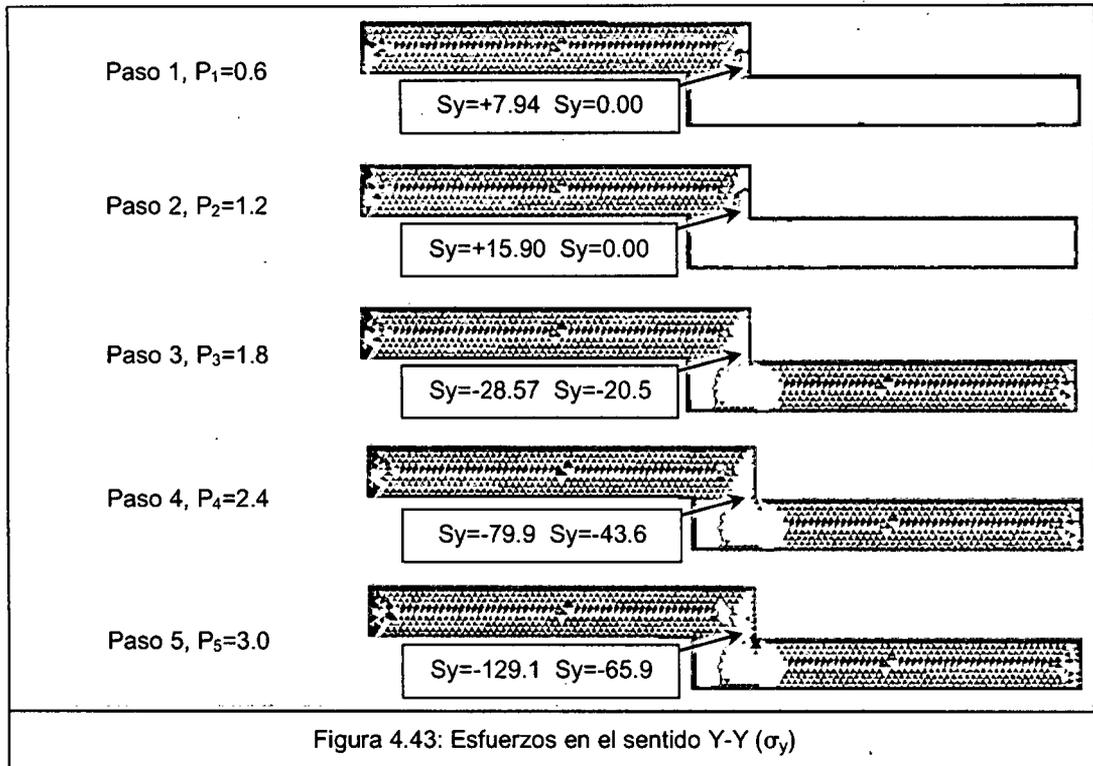


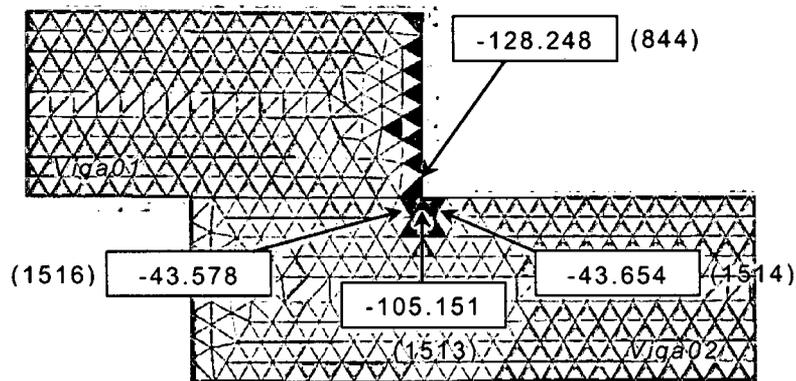
Figura 4.41: Esfuerzos Alisados en cada sentido

Se realizó además, el análisis no lineal geométrico a través de proceso iterativo incremental “paso a paso”. Las cuales se muestran los Esfuerzos a medida que se va incrementando las cargas. Para este ejemplo se dividió la carga en 05 partes hasta llegar con la carga total.





A continuación se presenta los Esfuerzos ( $\sigma_y$ ) en la zona de contacto, calculados en el Punto de Gauss y los Esfuerzos Alisados mediante el método de Promedio Directo:



Los valores de los esfuerzos en cada elemento triangular están entre el rango de -213.31 a +213.74

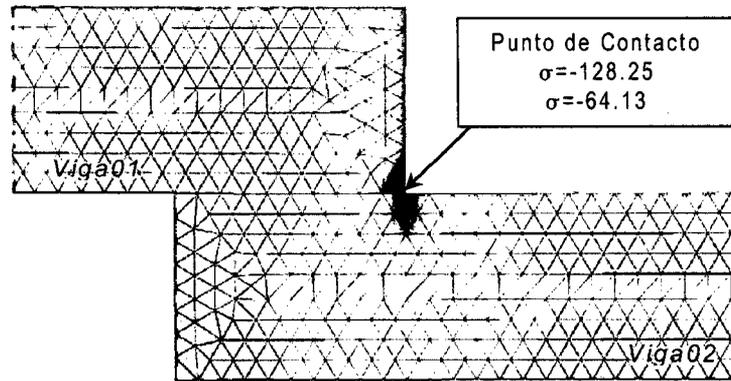


Figura 4.45: Esfuerzos Alisados

Los valores de los esfuerzos en cada nudo están entre el rango de -213.31 a +213.74

Los valores de los parámetros locales para dichos elementos son:

ID Elemento	Parámetro Local
844	5.35747
1516	3.93183
1513	4.55104
1514	3.83542

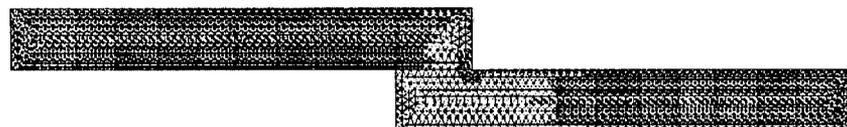
Con respecto a la Normas:

Para un Error Global Relativo igual a 0.05 (5%):

Norma de la Energética Total = 0.0509

Norma de la Deformación Total = 0.2220

Parámetro de Error Total = 4.5851



Parámetro Local:

Color Cyan  $\leq 1$

Color Rojo  $[1, 8.4]$

Color Negro  $>8.4$

Parámetro local mínimo = 0.004

Parámetro local máximo = 15.73

Figura 4.46: Valor de Parámetros Locales

Se procederá a reducir los tamaños de los elementos triangulares que están involucrados en la zona de contacto y en la zona de empotramiento de la viga 01 con la finalidad de mejorar los resultados de los esfuerzos.

La nueva malla de los cuerpos en estudio está compuesta por:  
1668 nudos, 3022 elementos triangulares y 16 elementos de contacto

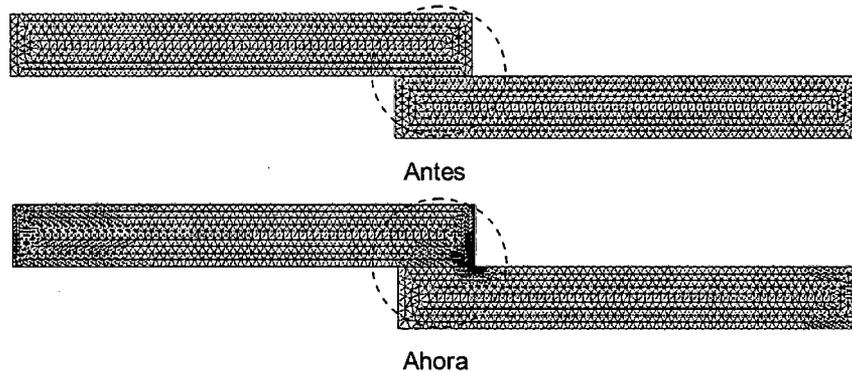


Figura 4.47: Comparativa entre mallas

Amplificando la zona de contacto para tener una mejor visualización de la geometría:

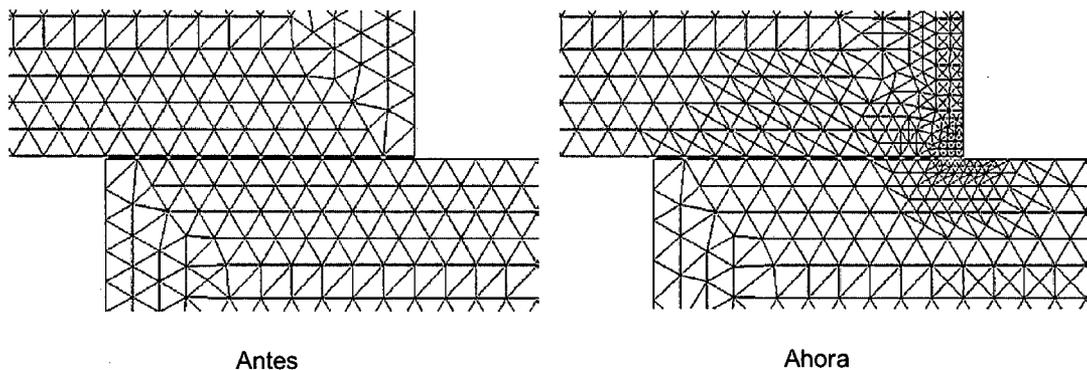


Figura 4.48: Comparativa entre mallas, área localizada

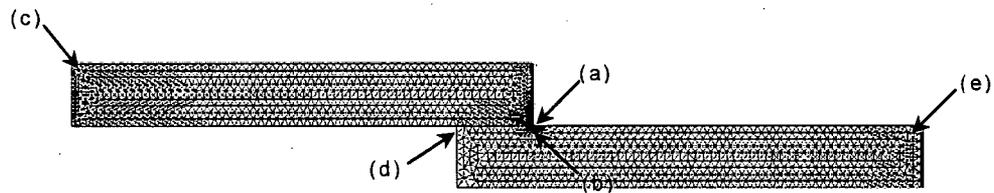
A continuación se presentan los resultados obtenidos:

- Para el caso de la Longitud de Contacto ( $l_c$ ):

En este caso, no se encontró más que un solo punto de contacto a pesar que se fue incrementando de manera iterativa el valor de la rigidez del elemento de contacto hasta llegar a cumplir la condición de impenetrabilidad.

Nº ElemCont	Longitud de Contacto (m)										
	0	0.012	0.025	0.0365	0.05	0.065	0.10	0.125	0.15	0.20	0.25
1	0	0.0001	0.0001	0.0002	0.0002	0.0004	0.0005	0.0006	0.0008	0.0010	0.0013

- Para el caso de los Desplazamientos ( $\delta_y$ ):



ID Nudo	$\delta_y$ 3pto Lineal	$\delta_y$ 3pto Lineal (H Adap)
a	-0.0076	-0.0076
b	-0.0025	-0.0026
d	-0.0032	-0.0033

- Para el caso de los Esfuerzos:

ID Nudo	Tipo	3pto Lineal	3pto Lineal (H Adap)
a	$\sigma_y$	-128.25	-507.62
b	$\sigma_y$	-64.13	-264.92
c	$\sigma_x$	1,068.70	1,115.40
e	$\sigma_x$	498.87	493.26

- Para el caso de la Norma:



Error Global Relativo = 0.05 (5%)

Parámetro de Error Total = 3.75

De la Energética Total = 0.0419

Parámetro local mínimo = 0.005

De la Deformación Total = 0.2235

Parámetro local máximo = 8.55

Figura 4.49: Valor de Parámetros Locales

A continuación se presenta los Esfuerzos ( $\sigma_y$ ) en la zona de contacto, calculados en el Punto de Gauss y los Esfuerzos Alisados mediante el método de Promedio Directo:

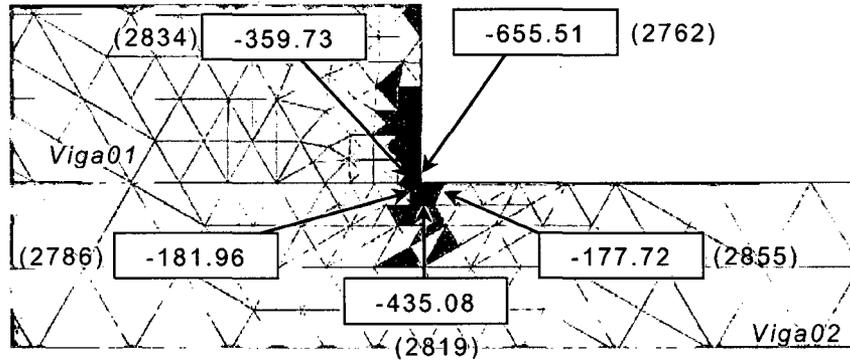


Figura 4.50: Esfuerzos en cada elemento triangular

Los valores de los esfuerzos en cada elemento triangular están entre el rango de -655.51 a +206.32

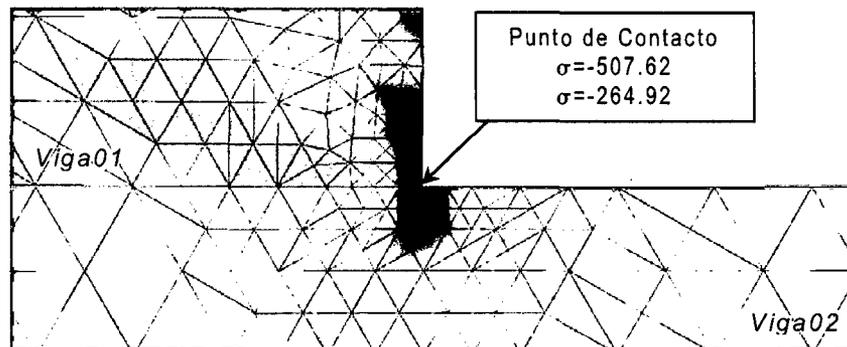


Figura 4.51: Esfuerzos Alisados

Los valores de los esfuerzos alisados en cada nudo están entre el rango de -507.62 a +160.20

Los valores de los parámetros locales para dichos elementos son:

ID Elemento	Parámetro Local
2762	4.42361
2834	3.67977
2786	4.73616
2819	5.52792
2855	4.72957

Para la zona de contacto, se grafica los valores de los esfuerzos en la dirección Y-Y para los casos analizados en la sección transversal de la viga ubicada sobre los puntos (a) y (b).

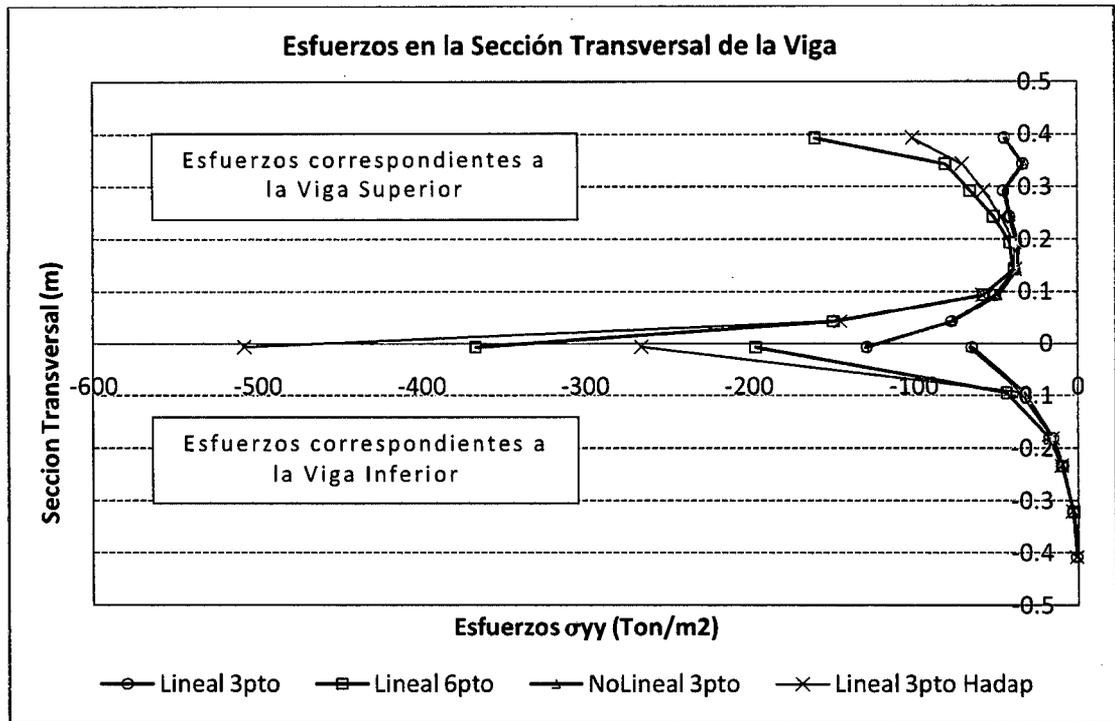


Figura 4.52: Esfuerzos en Zona de Contacto

Los valores de los Esfuerzos mostrados son obtenidos con Esfuerzos Alisados (Método Promedio Directo)

## CONCLUSIONES

A continuación se presentan las conclusiones que se pueden obtener a partir de los ejemplos realizados en la presente tesis.

- El uso de mallas no estructuradas tiene mayor importancia en comparación a las mallas estructuradas debido al tiempo de proceso y la zona donde se desea afinar más los resultados. Por consecuencia, se debe de tener un mejor algoritmo de generador de malla.
- El uso de los Multiplicadores de Lagrange da mejores resultados en comparación a los de Penalidad.
- Los desplazamientos obtenidos mediante los distintos modelos matemáticos son similares así mismo los resultados con el programa comercial SAP2000 y ANSYS.
- Los esfuerzos normales tiene una tendencia de inestabilidad, debido a la cantidad de grados de libertad y la forma de discretizar el elemento en estudio.
- Los valores de la rigidez del resorte en la zona de contacto, tienen una mayor importancia, llegando a involucrar en los resultados.
- El criterio de aplicar la carga puntual sobre un cuerpo tiene relevancia en la activación de los elementos de contacto, las cuales se pueden apreciar en los dos últimos ejemplos.
- La implementación del análisis no lineal geométrico, ha hecho visualizar mejor el problema de manera que se puede ver cómo se van activando los elementos de contacto a medida que se incrementa la carga.
- Para el cálculo de los desplazamientos, el uso de elementos triangulares con seis nudos tiene mayor relevancia en comparación con elementos triangulares de tres nudos.
- El uso de elementos de contacto tiene una importancia cuando se desea analizar cuerpos que interactúan entre ellos mismos.
- La Teoría de Estimación de Error es una herramienta importante en la verificación de los esfuerzos.

## RECOMENDACIONES

Después de haber realizado este trabajo, se puede recomendar lo siguiente:

- Existe otros operadores de contacto, las cuales la información es muy limitada o su implementación computacional es muy elevada. Queda la posibilidad de investigar más sobre estos operadores y verificar su calidad de resultados.
- El obtener un valor de esfuerzo más próximo al teórico involucra un gran esfuerzo computacional e incluso no pudiendo llegar a dicho valor. Una alternativa sería haciendo uso de otros funcionales matemáticos que tengan como variables los desplazamientos y los esfuerzos al mismo tiempo.
- Realizar ensayos en laboratorio para poder calibrar los resultados obtenidos con los operadores de contacto Master-Slave.
- Realizar estudios e implementar el programa para el caso que los cuerpos estén en un estado dinámico, así como también ampliar el tema cuando se aplique cargas que generen deformaciones significantes llegando ocasionar el autocontacto del mismo cuerpo.
- Implementar mejores algoritmos computacionales con el fin de tener menos tiempo de proceso.

## BIBLIOGRAFIA

- Bathe, Klaus-Jurgen; *FINITE ELEMENT PROCEDURES IN ENGINEERING ANALYSIS*; Prentice Hall; Estados Unidos; 1982
- Belytschko, Ted; *NONLINEAR FINITE ELEMENTS FOR CONTINUA AND STRUCTURES*; Northwestern University; 1998
- Chandrupatla, Tirupathi R.; *Introducción al estudio del elemento finito en ingeniería*; Prentice Hall; Mexico; 1999
- Dari, Enzo Alberto; *CONTRIBUCIONES A LA TRIANGULACIÓN AUTOMÁTICA DE DOMINIOS TRIANGULARES*; Tesis para optar el grado de Doctor en Ingeniería Nuclear; Universidad Nacional de Cuyo; Argentina; 1994
- Echegoyen Martín, Tomás José; *DESARROLLO DE UN LENGUAJE ORIGINAL DE PROCESO DE MALLAS Y APLICACIONES A LA INGENIERÍA CIVIL*; Tesis para optar el grado de Doctor; Universidad Politécnica de Madrid; España; 2002
- Felippa, Carlos A.; *INTRODUCTION TO FINITE ELEMENT METHODS*; University of Colorado at Boulder; Estados Unidos; 2006
- Fischer-Cripps, Anthony; *INTRODUCTION TO CONTACT MECHANICS*; Springer; Estados Unidos; 2007
- Hoffman, Joe D.; *NUMERICAL METHODS FOR ENGINEERS AND SCIENTISTS*; Marcel Dekker Inc., Estados Unidos; 2001
- Johnson K.L.; *CONTACT MECHANICS*; Cambridge University Press; Reino Unido; 1985
- Kiusalaas, Jaan; *NUMERICAL METHODS IN ENGINEERING WITH MATLAB*; Cambridge University Press; 2005

- Laursen, Tod A.; *A MORTAR-FINITE ELEMENT FORMULATION FOR FRICTIONAL CONTACT PROBLEMS*; Research Report, Duke University; 1999
- Nakasone, Yoshimoto and Stolarski; *ENGINEERING ANALYSIS WITH ANSYS SOFTWARE*; Elsevier; Reino Unido; 2006
- Oñate Ibáñez de Navarra, Eugenio; *CALCULO DE ESTRUCTURAS POR EL MÉTODO DE ELEMENTOS FINITOS*; CIMNE, 2º Edición; España; 1995
- Petkevicius, Kulak, Marchertas; *A PINBALL METHOD BY DIRECT LOCALIZATION OF THE IMPACT AREA*; Conference on Structural Mechanics in Reactor Technology; Prague; 2003
- Quaranta Neto, Francisco; *MODELAGEM DE PROBLEMAS DE CONTATO-IMPACTO EMPREGANDO FORMULACOES PENALIZADAS DO METODO DOS ELEMENTOS FINITOS*; Tesis para optar el grado de Doctor en Ciencias en Ingeniería Civil; Universidade Federal do Rio de Janeiro, UFRJ; Brasil; 2002
- Scaletti Farina, Hugo; *Apuntes de Clase, ELEMENTOS FINITOS*; Postgrado en Estructuras, Universidad Nacional de Ingeniería; Perú; 2007
- Silva, Joao M. S. – Sousa, Leonel Augusto; *IMPLEMENTACAO PARALELA DE UM ALGORITMO DE TRIANGULACAO DE DELAUNAY*; Artículo Técnico, INESC ID Lisboa; Portugal; 2001
- Tomas Celigueta, Juan; *Apuntes de Clase, ALISADO DE TENSIONES*; Universidad de Navarra; España; 2007
- Tomas Celigueta, Juan; *ANALISIS DE ESTRUCTURAS CON NO LINEALIDAD GEOMETRICA*; Universidad de Navarra; España; 2010
- Vásquez Chicata, Luis; *Apuntes de Clase, ANÁLISIS AVANZADO DE ESTRUCTURAS*; Postgrado en Estructuras, Universidad Nacional de Ingeniería; Perú; 2003

- Wriggers, Peter and Fischer, Katherine; *RESEARCH PROJECT, MORTAR METHOD FOR NONLINEAR CONTACT PROBLEMS*; University Hannover; 2003
- Xavier Oliver Olivella; *MECANICA DE MEDIOS CONTINUOS PARA INGENIEROS*; Ediciones UPC; España; 2000
- Zienkiewicz, O. C. / Taylor, R. L.; *EL MÉTODO DE LOS ELEMENTOS FINITOS, Vol.1 Vol.2*; CIMNE, 2º Edición; España; 1995

## **ANEXOS**

- 1.- CALCULO DE TENSIONES**
- 2.- NO LINEALIDAD GEOMETRICA**
- 3.- ESTIMACION DEL ERROR**
- 4.- EJEMPLOS DEMOSTRATIVOS**

## CÁLCULO DE TENSIONES

Debido que el funcional utilizado para el desarrollo del elemento finito está en función a los desplazamientos, las tensiones en los nudos están en función de los desplazamientos de la manera siguiente:

$$\sigma_i = D.B.u^{(e)}$$

Una manera de conseguir este valor en el nudo, primero se debe de calcular la tensión en los puntos de Gauss ya que este punto los valores de las tensiones son más precisas, luego por un proceso de extrapolación se podrá obtener la tensión en el nudo, no obstante existe una discontinuidad en un nudo determinado por los elementos que contiene dicho nudo.

Para eliminar dicha discontinuidad de tensión en dicho nudo, se procede a realizar unos cálculos denominados "Alisado de Tensiones".

### Calculo de Tensiones en Elementos Triangulares de 3 Nudos

Existe una particularidad para este tipo de elemento, las tensiones tienen el mismo valor en todo el elemento triangular de manera constante (CST)

La matriz  $D$  llamada matriz de *constante elástica* o matriz *constitutiva*, está formada por:

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

$$d_{11} = d_{22} = \frac{E}{1-\nu^2}$$

$$d_{11} = d_{22} = \frac{E.(1-\nu)}{(1+\nu).(1-2\nu)}$$

$$d_{12} = d_{21} = \nu.d_{11}$$

$$d_{12} = d_{21} = \left(\frac{\nu}{1-\nu}\right).d_{11}$$

$$d_{33} = \frac{E}{2.(1+\nu)} = G$$

$$d_{33} = \frac{E}{2.(1+\nu)} = G$$

Tensión Plana

Deformación Plana

La matriz  $D$  mostrada corresponde a un elemento Isotrópico y esta a la vez dependerá al estado plano a elegir

La matriz  $B$  llamada matriz de *deformación de un elemento* esta dado por:

$$B = [B_1, B_2, \dots, B_n]$$

n=3: elementos triangulares de 3 nudos.  
n=6: elementos triangulares de 6 nudos

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

Matriz de Deformación del Elemento

Matriz de Deformación del Nodo i

$$B_i = \frac{1}{2.A^{(e)}} \begin{bmatrix} b_i & 0 \\ 0 & c_i \\ c_i & b_i \end{bmatrix}$$

$$a_i = x_j y_k - x_k y_j ; \quad b_i = y_j - y_k ; \quad c_i = x_k - x_j ; \quad i, j, k = 1, 2, 3$$

Por lo tanto:

De manera directa se puede calcular los valores de las Tensiones ( $\sigma_x, \sigma_y, \tau_{xy}$ ) en cada nudo de cada elemento.

Por ejemplo:

Para el nudo 1 del elemento 5

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix} = \begin{bmatrix} d_{11} & d_{12} & 0 \\ d_{21} & d_{22} & 0 \\ 0 & 0 & d_{33} \end{bmatrix} \cdot \frac{1}{2.A^{(5)}} \begin{bmatrix} y_2 - y_3 & 0 \\ 0 & x_3 - x_2 \\ x_3 - x_2 & y_2 - y_3 \end{bmatrix} \begin{bmatrix} u1_x \\ u1_y \end{bmatrix}$$

### Alisado de Tensiones para elementos triangulares de 3 Nudos

Como existe una discontinuidad en las tensiones de cada nudo, se procederá a realizar el alisado del mismo, las cuales existen algunos métodos que nos ayudaran a mejorar los resultados.

**Alisado Promedio Directo.**- Consiste en calcular un promedio de los valores de las tensiones que actúan en dicho punto. Es un procedimiento simple y rápido.

$$\bar{\sigma}_i = \frac{\sum_e \sigma_i^e}{n_i^e}$$

**Alisado Global.**- Consiste en calcular el valor de la tensión con respecto a todos los nudos involucrados en el problema.

$$\int N_f^T \cdot N_f \cdot dv \cdot \bar{\sigma}^e = \int N_f^T \cdot \sigma \cdot dv$$

$$M^e \bar{\sigma}^e = R^e$$

$$N_1 = L_1 ; N_2 = L_2 ; N_3 = L_3$$

$$L_1 = 1 - \alpha - \beta ; L_2 = \alpha ; L_3 = \beta$$

Otro dato que se debe de tener en cuenta para los desarrollos de las integrales en coordenadas de Áreas

$$\iint L_1^p \cdot L_2^q \cdot L_3^r = \frac{p! \cdot q! \cdot r!}{(p + q + r + 2)!} \cdot 2A$$

Ordenando la ecuación:

$$\iint \begin{bmatrix} N_1 \\ N_2 \\ N_3 \end{bmatrix} \cdot [N_1 \quad N_2 \quad N_3] \cdot dA \cdot \begin{bmatrix} \sigma_{im} \\ \sigma_{jm} \\ \sigma_{km} \end{bmatrix}_{\text{alisadas}} = \iint \begin{bmatrix} N_1 \\ N_2 \\ N_3 \end{bmatrix} \cdot dA \cdot [\sigma_{im} \quad \sigma_{jm} \quad \sigma_{km}]_{\text{calculadas}}$$

$$i, j, k = \text{nudos} ; m = x, y, xy$$

Desarrollando cada integral:

$$(*) \iint \begin{bmatrix} N_1 \\ N_2 \\ N_3 \end{bmatrix} \cdot [N_1 \quad N_2 \quad N_3] \cdot dA = \iint \begin{bmatrix} L_1^2 & L_1 L_2 & L_1 L_3 \\ L_2 L_1 & L_2^2 & L_2 L_3 \\ L_3 L_1 & L_3 L_2 & L_3^2 \end{bmatrix} \cdot dA = \frac{A}{12} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

$$(**) \iint \begin{bmatrix} N_1 \\ N_2 \\ N_3 \end{bmatrix} \cdot dA = \iint \begin{bmatrix} L_1 \\ L_2 \\ L_3 \end{bmatrix} \cdot dA = \frac{A}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

De esta manera se elabora una matriz de 6x6 para cada elemento y para cada componente analizado.

### Calculo de Tensiones en Elementos Triangulares de 6 Nudos

En este caso, primero se calcularán las tensiones en los puntos gaussianos correspondientes, debido que existe una mejor aproximación al valor verdadero.

Se consideraran como puntos gaussianos las coordenadas como se indican a continuación en el sistema de coordenadas naturales:

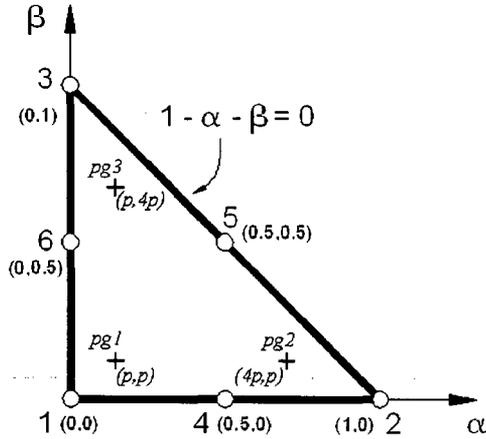
Usaremos 3 puntos gaussianos para desarrollar las integrales de las Tensiones:

$$pg1 = (p, p) \Rightarrow L_1 = 1 - p - p ; L_2 = p ; L_3 = p ; W_1 = 1/6$$

$$pg2 = (4p, p) \Rightarrow L_1 = 1 - p - 4p ; L_2 = 4p ; L_3 = p ; W_2 = 1/6$$

$$pg3 = (p, 4p) \Rightarrow L_1 = 1 - 4p - p ; L_2 = p ; L_3 = 4p ; W_3 = 1/6$$

siendo :  $p = 1/6$



El valor de la tensión está dada por:

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix}_p = \sum_{i=1}^6 [D.B_i]_p \cdot u_i^{(e)} ; p = 1..3 \text{ (pto.gauss)}$$

$$B_i = \begin{bmatrix} \frac{\partial N_i}{\partial x} & 0 \\ 0 & \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} \end{bmatrix} ; \begin{bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{bmatrix} = [J^{(e)}]^{-1} \cdot \begin{bmatrix} \frac{\partial N_i}{\partial \alpha} \\ \frac{\partial N_i}{\partial \beta} \end{bmatrix} ; J^{(e)} = \sum_{i=1}^n \begin{bmatrix} \frac{\partial N_i}{\partial \alpha} \cdot x_i & \frac{\partial N_i}{\partial \alpha} \cdot y_i \\ \frac{\partial N_i}{\partial \beta} \cdot x_i & \frac{\partial N_i}{\partial \beta} \cdot y_i \end{bmatrix}$$

Las funciones de Forma en coordenadas naturales son:

$$L_1 = 1 - \alpha - \beta ; L_2 = \alpha ; L_3 = \beta$$

$$N_1 = (1 - 2\alpha - 2\beta) \cdot (1 - \alpha - \beta) ; N_2 = (2\alpha - 1)\alpha ; N_3 = (2\beta - 1)\beta$$

$$N_4 = 4(1 - \alpha - \beta)\alpha ; N_5 = 4\alpha\beta ; N_6 = 4(1 - \alpha - \beta)\beta$$

Después de calcular las tensiones en los puntos gaussianos, se procederá a extrapolarlos a los nudos que conforman el elemento triangular de 6 nudos.

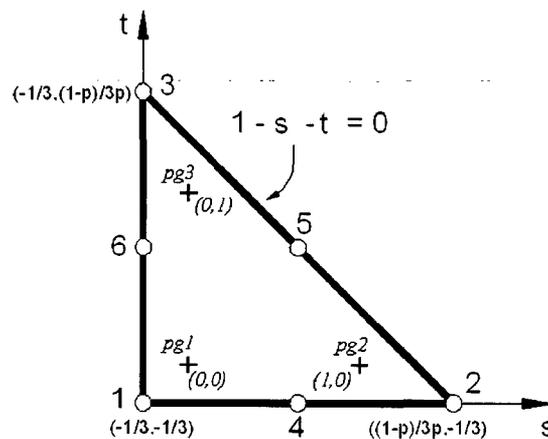
### Extrapolación de Tensiones en elementos triangulares de 6 Nudos

La extrapolación desde los puntos de gauss hacia los nudos del elementos triangular se realizará a través de las funciones de forma, y con cierta modificación en su enunciado.

$$\sigma = \sum_{i=1}^3 N_i(s,t) \cdot \sigma_i$$

La variación del sistema de coordenadas naturales es:

$$s = \frac{(\alpha - p)}{3p} ; t = \frac{(\beta - p)}{3p}$$



El valor de la tensión en el nudo del elemento triangular de 6 nudos, estará calculada de acuerdo a la formula siguiente, teniendo en cuenta la posición, el orden y la dirección tensional.

$$\sigma = (1 - s - t) \cdot \sigma_{pg1} + s \cdot \sigma_{pg2} + t \cdot \sigma_{pg3}$$

### Alisado de Tensiones para elementos triangulares de 6 Nudos

Para este caso, se podrá utilizar el método de Alisado Promedio Directo descrito anteriormente, debido que simplicidad computacional.

$$\bar{\sigma}_i = \frac{\sum_e \sigma_i^e}{n_i^e}$$

## NO LINEALIDAD GEOMETRICA

Inicialmente, para el estudio de una estructura, asumimos varias hipótesis las cuales nos ayuden a entender el comportamiento e incluso nos ayuden a eliminar variables complejas de las que se presentan. Para el caso del estudio y/o comportamiento de una estructura a un nivel superior, las cargas aplicadas a un cuerpo en algunos casos producen grandes deformaciones a este, de manera que la hipótesis de que la posición deformada final coincide con la posición inicial no puede aceptarse, concluyendo que no se puede plantear las ecuaciones de equilibrio en la posición inicial.

Debido a estas consideraciones, se hace que el problema sea un problema no lineal del tipo geométrico, haciendo que no se pueda calcular en general la situación deformada final en un solo paso, aplicando la totalidad de la carga de una vez. Teniendo una alternativa para este caso, la cual implicaría realizar un proceso de carga incremental y determinado de esta manera la respuesta en cada incremento.

Siguiendo las teorías de las medidas de deformaciones, las ecuaciones constitutivas y las ecuaciones de equilibrio, se llegan a las formulaciones planteadas por Lagrange conocidas como: Formulación Lagrangiana Total y Formulación Lagrangiana Actualizada.

Sea un sólido en equilibrio en un instante  $t$ . La condición de equilibrio viene dada por el Principio de Trabajo Virtual.

$$\delta W_{Int} = \int_v \delta \epsilon \cdot \sigma \cdot dv = \delta W_{Ext}$$

Las relaciones de la Deformación Unitaria Infinitesimal con el gradiente de Desplazamientos está dada por:

$$\delta \bar{\epsilon} = A_o \cdot \delta \bar{H}_t$$

$$\epsilon_{ij} = \frac{1}{2} \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) = \frac{1}{2} (H_t + H_t^T)$$

$$\bar{H}_t = \begin{bmatrix} \frac{\partial}{\partial x_1} & 0 \\ \frac{\partial}{\partial x_2} & 0 \\ 0 & \frac{\partial}{\partial x_1} \\ 0 & \frac{\partial}{\partial x_2} \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \partial_t \cdot u \quad \bar{\varepsilon} = \begin{bmatrix} \frac{\partial u_1}{\partial x_1} \\ \frac{\partial u_2}{\partial x_2} \\ \frac{\partial u_1}{\partial x_2} + \frac{\partial u_2}{\partial x_1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial u_1}{\partial x_1} \\ \frac{\partial u_1}{\partial x_2} \\ \frac{\partial u_2}{\partial x_1} \\ \frac{\partial u_2}{\partial x_2} \end{bmatrix}$$

$$u = N \cdot U$$

$$\bar{\varepsilon} = A_o \cdot \bar{H}_t$$

$$\bar{H}_t = \partial_t \cdot u = \partial_t \cdot N \cdot U = G_t \cdot U$$

$$\delta \bar{\varepsilon} = A_o \cdot G_t \cdot \delta U = B_t \cdot \delta U$$

$$\delta W_{Int} = \int_v \delta U^T \cdot B_t^T \cdot \bar{\sigma} \cdot dv = \delta U^T \cdot \int_v B_t^T \cdot \bar{\sigma} \cdot dv = \delta U^T \cdot Q$$

Siendo  $Q$  el vector de fuerzas nodales equivalentes a los esfuerzos interiores en el instante  $t$ .

Supongamos conocida la configuración de equilibrio en el instante  $t$  del proceso de carga y buscamos la configuración de equilibrio en  $(t + \Delta t)$

$$\delta W_I^{t+\Delta t} \approx \delta W_I + \Delta_{\hat{u}} \cdot (\delta W_I) = \delta W_I + \delta U^T \cdot \hat{R} \cdot \hat{U}$$

El vector  $\hat{U}$  contiene los incrementos en las deformaciones nodales del elemento asociada al incremento  $\hat{u}$

$$\Delta_{\hat{u}} \cdot (\delta W_I) = \int_v \delta \bar{\varepsilon}^T \cdot \Delta \bar{\sigma} \cdot dv + \int_v \Delta(\delta \varepsilon) : \sigma \cdot dv$$

$$\Delta(\delta W_I) = \Delta(\delta W_I)_{mat} + \Delta(\delta W_I)_{geo}$$

El primer sumando corresponde al incremento de las tensiones manteniendo fijas las deformaciones unitarias (*componente del material*). El segundo sumando corresponde al incremento de las deformaciones unitarias manteniendo fijas las tensiones (*componente geométrico*).

Para el Componente del Material, sabiendo que la relación tensión-deformación está dada por:

$$\Delta \bar{\sigma} = D \cdot \Delta \bar{\varepsilon}$$

$$\Delta \bar{\varepsilon} = B \cdot \hat{U}$$

Sustituyendo las expresiones:

$$\Delta(\delta W_I)_{mat} = \int_v \delta \bar{\varepsilon}^T \cdot \Delta \bar{\sigma} \cdot dv = \delta U^T \cdot \int_v B^T \cdot D \cdot B \cdot dv \cdot \hat{U}$$

Para el Componente Geométrico, el incremento de la variación de la deformación unitaria es:

$$\Delta(\delta \varepsilon) = \frac{1}{2} \cdot (\delta H_t^T \cdot \Delta H_t + \Delta H_t^T \cdot \delta H_t)$$

Sustituyendo las expresiones:

$$\Delta(\delta W_I)_{geo} = \int (\delta H_t^T \quad \Delta H_t) : \sigma \cdot dv$$

$$\Delta(\delta W_I)_{geo} = \int \begin{bmatrix} \frac{\partial \delta u_1}{\partial x_1} \\ \frac{\partial \delta u_1}{\partial x_2} \\ \frac{\partial \delta u_2}{\partial x_1} \\ \frac{\partial \delta u_2}{\partial x_2} \end{bmatrix}^T \cdot \begin{bmatrix} \sigma_{11} & \sigma_{12} & 0 & 0 \\ \sigma_{12} & \sigma_{22} & 0 & 0 \\ 0 & 0 & \sigma_{11} & \sigma_{12} \\ 0 & 0 & \sigma_{12} & \sigma_{22} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial \hat{u}_1}{\partial x_1} \\ \frac{\partial \hat{u}_1}{\partial x_2} \\ \frac{\partial \hat{u}_2}{\partial x_1} \\ \frac{\partial \hat{u}_2}{\partial x_2} \end{bmatrix} \cdot dv$$

Agrupando

$$\Delta(\delta W_I)_{geo} = \int \delta \hat{H}_t^T \cdot \sigma \cdot \hat{H}_t \cdot dv$$

$$\hat{H}_t = \partial_t \hat{u} = \partial_t N \hat{U} = G_t \hat{U}$$

$$\Delta(\delta W_I)_{geo} = \int_v \Delta(\delta \varepsilon) : \sigma \cdot dv = \delta U^T \int_v G_t^T \cdot \sigma \cdot G_t \cdot dv \cdot \hat{U}$$

Al final, el incremento del Trabajo Virtual queda definida por:

$$\Delta(\delta W_I) = \delta U^T \cdot \int_v B^T \cdot D \cdot B \cdot dv \cdot \hat{U} + \delta U^T \int_v G_t^T \cdot \sigma \cdot G_t \cdot dv \cdot \hat{U}$$

$$\Delta(\delta W_I) = \delta U^T \cdot (\hat{K}_D + \hat{K}_\sigma) \cdot \hat{U} = \delta U^T \cdot \hat{K} \cdot \hat{U}$$

Siendo  $\hat{K}$  la matriz de Rigidez Tangente, las cuales constan de dos sumandos. La primera corresponde a la rigidez asociada al incremento de Tensiones sobre un material dado, es similar con la matriz de rigidez en el análisis lineal, aunque ahora la matriz  $B$  es dependiente de las deformaciones existentes. El segundo

sumando, corresponde a la rigidez asociada al incremento de las deformaciones unitarias actuando sobre el estado de tensiones ya existentes.

El estado de equilibrio del sistema sería:

$$\hat{K} \cdot \hat{U} = P^{t+\Delta t} - \int_v B_t^T \cdot \bar{\sigma} \cdot dv$$

## ESTIMACION DEL ERROR

La norma energética del error correspondiente a las tensiones está dada por la ecuación:

$$\|e_\sigma\|^2 = \int_{\Omega} e_\sigma^T \cdot D^{-1} \cdot e_\sigma \cdot d\Omega \quad ; \quad e_\sigma = \sigma_a - \sigma$$

Siendo  $\sigma_a$  el esfuerzo alisado nodal y  $\sigma$  el esfuerzo en el punto de Gauss

- Para el caso de un elemento triangular de 3 nudos sería:

Sea  $i$  el nudo perteneciente al elemento triangular

$$\begin{bmatrix} \sigma_{i-x}^a \\ \sigma_{i-y}^a \\ \sigma_{i-xy}^a \end{bmatrix} - \begin{bmatrix} \sigma_{i-x} \\ \sigma_{i-y} \\ \sigma_{i-xy} \end{bmatrix} = \begin{bmatrix} \Delta\sigma_{i-x} \\ \Delta\sigma_{i-y} \\ \Delta\sigma_{i-xy} \end{bmatrix}$$

La Matriz Constitutiva y la Matriz Constitutiva Inversa, está dada por:

$$D = \begin{bmatrix} d_{11} & d_{12} & 0 \\ d_{21} & d_{22} & 0 \\ 0 & 0 & d_{33} \end{bmatrix} \Rightarrow D^{-1} = \begin{bmatrix} dd_{11} & dd_{12} & 0 \\ dd_{21} & dd_{22} & 0 \\ 0 & 0 & dd_{33} \end{bmatrix}$$

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

Matriz Constitutiva, Estado de Tensión Plana

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

Matriz Constitutiva, Estado de Deformación Plana

Sea:

$$\begin{bmatrix} \Delta\sigma_{i-x} & \Delta\sigma_{i-y} & \Delta\sigma_{i-xy} \end{bmatrix} \begin{bmatrix} dd_{11} & dd_{12} & 0 \\ dd_{21} & dd_{22} & 0 \\ 0 & 0 & dd_{33} \end{bmatrix} \begin{bmatrix} \Delta\sigma_{i-x} \\ \Delta\sigma_{i-y} \\ \Delta\sigma_{i-xy} \end{bmatrix} = \gamma_i$$

Donde  $\gamma_i$  es el producto de las matrices indicadas anteriormente correspondientes al nudo  $i$  del elemento triangular y es un valor numérico (puede ser positivo o negativo).

De acuerdo al desarrollo de estas matrices, la norma energética de un elemento triangular de 3 nudos estaría dada por:

$$\|e_\sigma\|^2 = \int_{\Omega} \gamma_i \cdot d\Omega = \sum_{i=1}^{n=3} (\gamma_i \cdot t \cdot A) = t \cdot A \cdot (\gamma_1 + \gamma_2 + \gamma_3)$$

Para la presente tesis, una herramienta importante para saber que tan aceptable son los valores de los esfuerzos en los elementos estaría dada por un parámetro en la cual contenga el error global y parámetro de refinamiento del elemento triangular. Dentro de la teoría de errores estas herramientas engloban a la llamada "estrategia de refinamiento de la malla" las cuales deben de cumplir la condición de error global y la condición de malla optima.

Dado que la norma de energética de error global debe ser menor que la energía de deformación total. Entonces, el parámetro de error global estaría dada por:

Energía de Deformación:

$$\xi_g = \frac{\|e_\sigma\|}{\eta \cdot \|U\|} \qquad \|U\|^2 = \int_{\Omega} \sigma_a^T \cdot D^{-1} \cdot \sigma_a \cdot d\Omega$$

Porcentaje de error global:

$$\eta$$

El valor adecuado del parámetro de error global es uno ( $\xi_g = 1$ ) lo que indica que se cumple la condición de error global para un determinado porcentaje de error. Para un valor ( $\xi_g > 1$ ) indica que el tamaño del elemento debe de

reducirse, mientras que para un valor ( $\xi_g < 1$ ) indica que el tamaño del elemento debe de ampliarse.

Siguiendo el criterio de la malla optima basado en la equi-distribución del error global, el parámetro de refinamiento del elemento es el producto del parámetro de error global con parámetro de error local

$$\xi^{(e)} = \xi_g \cdot \bar{\xi}^{(e)} = \frac{\|e_\sigma\|}{\eta \cdot \|U\|} \cdot \frac{\|e_\sigma\|^{(e)}}{\|e_\sigma\| \cdot (\sqrt{n})^{-1}} = \frac{\|e_\sigma\|^{(e)} \cdot \sqrt{n}}{\eta \cdot \|U\|}$$

## EJEMPLOS DEMOSTRATIVOS

A continuación se presentará una serie de ejemplos simples que servirá para verificar y comparar los resultados con otros programas comerciales que tienen ciertas características entre ellas, así como también se detallará la metodología de cálculo.

### 1.- Viga empotrada en un extremo (V1)

$$L=1.00\text{m}$$

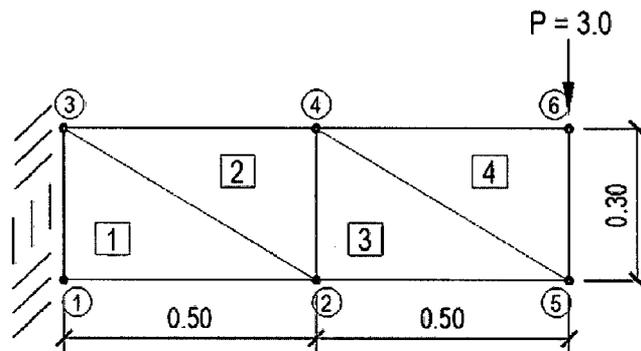
$$h=0.30\text{m}$$

$$e=0.20\text{m}$$

$$E=2.1\text{e}06 \text{ N/m}^2$$

$$\nu=0.25$$

$$P=-3\text{N}$$



Malla Triangular de 3 nudos

A continuación se presentan unas tablas comparativas entre el programa educativo ED-Elas2D v2.00.2 y el programa MEC-3pto implementado para esta tesis:

- Desplazamientos:

Nudo	ED-Elas2D		MEC-3pto	
	x	y	x	y
1	0.0000	0.0000	0.0000	0.0000
2	0.0000	-0.0001	0.0000	-0.0001
3	0.0000	0.0000	0.0000	0.0000
4	0.0000	-0.0001	0.0000	-0.0001
5	0.0000	-0.0002	0.0000	-0.0002
6	0.0000	-0.0002	0.0000	-0.0002

- Esfuerzos calculados en el Punto de Gauss:

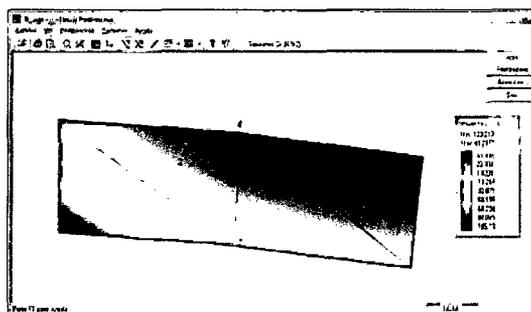
Elemento	ED-Elas2D			MEC - 3pto		
	Sx	Sy	Sxy	Sx	Sy	Sxy
1	-123.2130	-30.8032	-126.0720	-123.213	-30.803	-126.072
2	123.2130	5.2858	26.0723	123.213	5.286	26.072
3	-41.7111	-35.9452	-74.9733	-41.711	-35.945	-74.973
4	41.7111	-44.9840	-25.0267	41.711	-44.984	-25.027

- Esfuerzos Alisados (Promedio Directo)

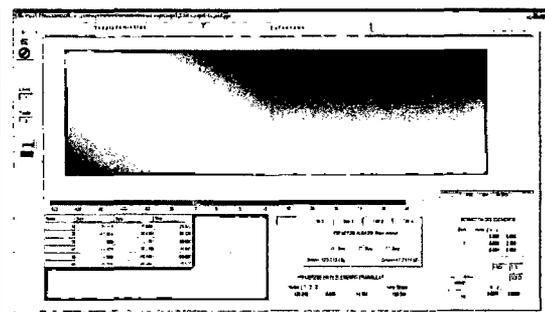
Nudo	ED-Elas2D			MEC - 3pto		
	Sx	Sy	Sxy	Sx	Sy	Sxy
1	-123.213	-30.803	-126.072	-123.213	-30.803	-126.072
2	-13.904	-20.488	-58.324	-13.904	-20.488	-58.324
3	0.000	-12.759	-50.000	0	-12.759	-50
4	41.071	-25.215	-24.643	41.071	-25.215	-24.643
5	0.000	-40.465	-50.000	0	-40.465	-50
6	41.711	-44.984	-25.027	41.711	-44.984	-25.027

- Esfuerzos Alisados (Global)

Nudo	ED-Elas2D			MEC - 3pto		
	Sx	Sy	Sxy	Sx	Sy	Sxy
1	NO PRESENTA ESTE ALGORITMO			-262.726	-53.36	-212.364
2				-22.374	-17.477	-63.242
3				54.975	0.985	-16.318
4				87.575	-17.695	4.121
5				-54.975	-54.208	-83.682
6				67.122	-54.016	-10.273



Esfuerzos Alisados Sx (EDElas2D)



Esfuerzos Alisados Sx (MEC-3pto)

Comparativa de Esfuerzos

A continuación se presenta la metodología de cálculo del alisado global para el caso del esfuerzo  $S_x$ .

$$\frac{A^{(e)}}{12} * \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} * \begin{bmatrix} Sax_i \\ Sax_j \\ Sax_k \end{bmatrix} = \frac{A^{(e)}}{3} * \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} * \begin{bmatrix} Sx_j \\ Sx_j \\ Sx_j \end{bmatrix}$$

Para el elemento 01:

(1)	(2)	(3)					
0.0125	0.00625	0.00625	Sax1	=	0.025	-123.213	(1)
0.00625	0.0125	0.00625	Sax2		0.025	-123.213	(2)
0.00625	0.00625	0.0125	Sax3		0.025	-123.213	(3)

Para el elemento 02:

(2)	(4)	(3)					
0.0125	0.00625	0.00625	Sax2	=	0.025	123.213	(2)
0.00625	0.0125	0.00625	Sax4		0.025	123.213	(4)
0.00625	0.00625	0.0125	Sax3		0.025	123.213	(3)

Para el elemento 03:

(2)	(5)	(4)					
0.0125	0.00625	0.00625	Sax2	=	0.025	-41.7111	(2)
0.00625	0.0125	0.00625	Sax5		0.025	-41.7111	(5)
0.00625	0.00625	0.0125	Sax4		0.025	-41.7111	(4)

Para el elemento 04:

(5)	(6)	(4)					
0.0125	0.00625	0.00625	Sax5	=	0.025	41.7111	(5)
0.00625	0.0125	0.00625	Sax6		0.025	41.7111	(6)
0.00625	0.00625	0.0125	Sax4		0.025	41.7111	(4)

Ensamblando estas cuatro matrices en la matriz general:

0.0125	0.00625	0.00625	0	0	0	Sax1	=	-3.08033
0.00625	0.0375	0.0125	0.0125	0.00625	0	Sax2		-1.04278
0.00625	0.0125	0.025	0.00625	0	0	Sax3		0
0	0.0125	0.00625	0.0375	0.0125	0.00625	Sax4		3.080325
0	0.00625	0	0.0125	0.025	0.00625	Sax5		0
0	0	0	0.00625	0.00625	0.0125	Sax6		1.042778

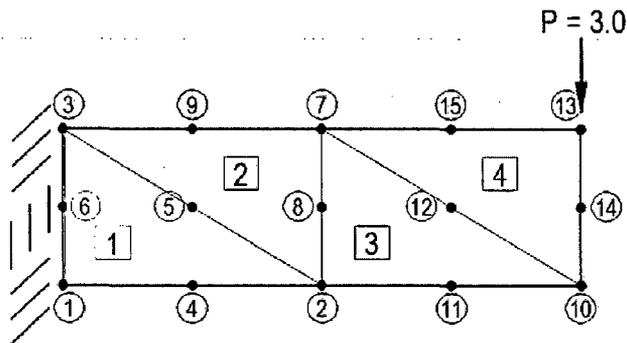
La solución del sistema de ecuaciones es:

$$\begin{array}{l|l|l} \text{Sax1} & & -262.726 \\ \text{Sax2} & & -22.3739 \\ \text{Sax3} & & 54.9747 \\ \text{Sax4} & = & 87.57546 \\ \text{Sax5} & & -54.9747 \\ \text{Sax6} & & 67.12182 \end{array}$$

De esta manera se obtiene los esfuerzos alisados ( $\sigma_x$ ) por el método global.

## 2.- Viga empotrada en un extremo (V2)

$L=1.00\text{m}$                        $e=0.20\text{m}$                        $\nu=0.25$   
 $h=0.30\text{m}$                        $E=2.1 \times 10^6 \text{ N/m}^2$                        $P=-3\text{N}$



Malla Triangular de 6 nudos

A continuación se presentan unas tablas comparativas entre el programa educativo ED-Elas2D v2.00.2 y el programa MEC-6pto implementado para esta tesis:

- Desplazamientos:

Nudo	ED-Elas2D		MEC - 6pto	
	x	y	x	y
1	0	0	0	0
2	-0.0002	-0.0003	-0.0002	-0.0003
3	0	0	0	0
4	-0.0001	-0.0001	-0.0001	-0.0001
5	0.0000	-0.0001	0	-0.0001
6	0.0000	0.0000	0	0

Nudo	ED-Elas2D		MEC - 6pto	
	x	y	x	y
7	0.0002	-0.0003	0.0002	-0.0003
8	0.0000	-0.0003	0	-0.0003
9	0.0001	-0.0001	0.0001	-0.0001
10	-0.0002	-0.0011	-0.0002	-0.0011
11	-0.0002	-0.0007	-0.0002	-0.0007
12	0.0000	-0.0007	0	-0.0007
13	0.0002	-0.0011	0.0002	-0.0011
14	0.0000	-0.0011	0	-0.0011
15	0.0002	-0.0007	0.0002	-0.0007

- Esfuerzos calculados en el Punto de Gauss (3ptos/elemento):

Pto G	Coor Pto Gauss		ED-Elas2D			MEC - 6pto		
	x	y	Sx	Sy	Sxy	Sx	Sy	Sxy
1	0.33333	0.05	-433.3160	-6.9281	-29.3170	-433.316	-6.92806	-29.317
2	0.08333	0.2	252.4020	19.9012	-62.1348	252.402	19.9012	-62.1348
3	0.08333	0.05	-566.4050	-47.7015	-60.1573	-566.405	-47.7015	-60.1573
4	0.41667	0.25	435.3830	31.4254	-50.6872	435.383	31.4254	-50.6872
5	0.16667	0.25	563.1090	-14.3427	-27.1715	563.109	-14.3427	-27.1715
6	0.41667	0.1	-251.1740	-16.3827	-70.5321	-251.174	-16.3827	-70.5321
7	0.83333	0.05	-104.2820	-16.7444	-24.0277	-104.282	-16.7444	-24.0277
8	0.58333	0.2	78.3679	-16.7170	-70.9666	78.3679	-16.717	-70.9666
9	0.58333	0.05	-226.7680	30.8304	-53.3966	-226.768	30.8304	-53.3966
10	0.91667	0.25	104.7780	-82.2800	-62.8667	104.778	-82.28	-62.8667
11	0.66667	0.25	232.6270	7.6621	-26.1732	232.627	7.6621	-26.1732
12	0.91667	0.1	-84.7229	-14.4166	-62.5693	-84.7229	-14.4166	-62.5693

- Extrapolación desde los puntos de Gauss hacia los nudos de cada elemento

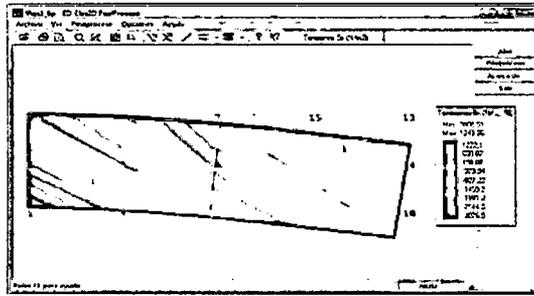
Nudo	MEC - 6pto Elem(1)			MEC - 6pto Elem(2)		
	Sx	Sy	Sxy	Sx	Sy	Sxy
1	-883.703	-83.8268	-69.7782			
2	-617.525	-2.28003	-8.09768	-751.454	-32.9987	-91.6006
3	753.91	51.3786	-73.7332	877.111	-28.9188	-4.87944
4	-750.614	-43.0534	-38.938			
5	68.1924	24.5493	-40.9154	62.8287	-30.9587	-48.24
6	-64.8966	-16.2241	-71.7557			
7				621.661	62.6174	-51.9108
8				-64.897	14.809	-71.756
9				749.386	16.849	-28.395
10						
11						

12		
13		
14		
15		

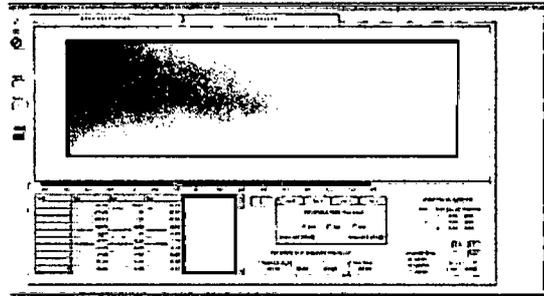
Nudo	MEC - 6pto Elem(3)			MEC - 6pto Elem(4)		
	Sx	Sy	Sxy	Sx	Sy	Sxy
1						
2	-369.308	62.5378	-57.3295			
3						
4						
5						
6						
7	240.963	-32.557	-92.4696	381.027	45.0024	-1.81
8	-64.1724	14.9904	-74.8996			
9						
10	-124.337	-32.612	1.408	-253.673	0.845	-74.602
11	-246.823	14.963	-27.9606			
12	58.3131	-32.5844	-45.5307	63.677	22.924	-38.206
13				125.328	-134.882	-75.197
14				-64.172	-67.018	-74.900
15				253.177	-44.940	-38.504

• Esfuerzos Alisados (Promedio Directo)

Nudo	ED-Elas2D			MEC - 6pto		
	Sx	Sy	Sxy	Sx	Sy	Sxy
1	-3608.51	-4800.81	-4786.2	-883.703	-83.8268	-69.7782
2	-2282.92	-3241.76	-2852.13	-579.429	9.08636	-52.3426
3	-956.349	-2603.34	-2620.62	815.511	11.2299	-39.3063
4	-427.384	-914.339	-1076.35	-750.614	-43.0534	-38.938
5	-623.128	-452.581	-588.637	65.5105	-3.20474	-44.5777
6	303.194	584.873	731.328	-64.8966	-16.2241	-71.7557
7	-2089.74	-2996.82	-2946.04	414.55	25.021	-48.7301
8	127.087	-304.97	-579.637	-64.5345	14.8999	-73.3276
9	1243.36	1505.57	1767.9	749.386	16.849	-28.395
10	-1959.61	-3610.74	-3613.88	-189.005	-15.883	-36.597
11	-550.996	-239.081	-319.014	-246.823	14.963	-27.9606
12	171.227	288.738	494.947	60.995	-4.83035	-41.8684
13	-1801.7	-3096.73	-2743.27	125.328	-134.882	-75.197
14	-500.364	-264.727	-471.273	-64.172	-67.018	-74.900
15	581.818	139.636	-11.6364	253.177	-44.940	-38.504



Esfuerzos Alisados Sx (EDElas2D)

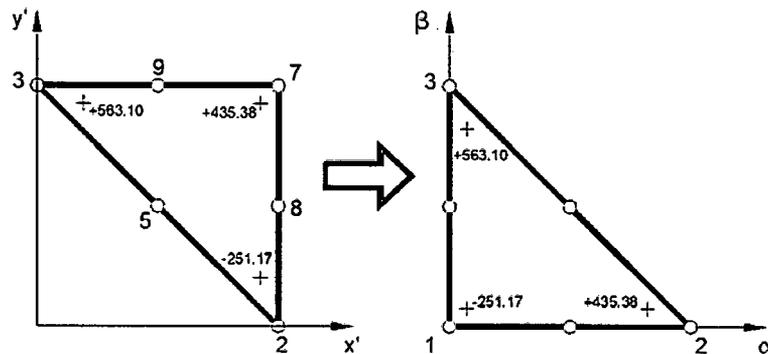


Esfuerzos Alisados Sx (MEC-6pto)

Comparativa de Esfuerzos

Se presenta la metodología de la extrapolación del esfuerzo desde el punto de Gauss a los seis nudos:

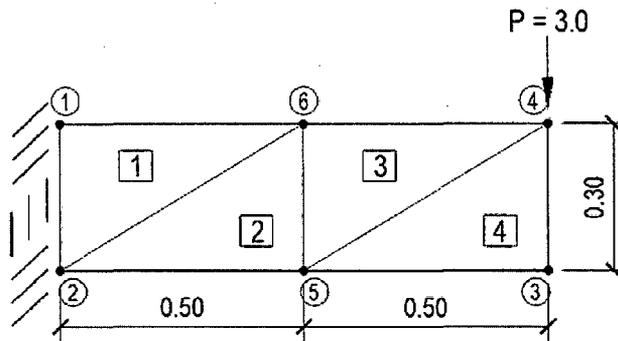
Para el elemento (2):



PG1	-251.174
PG2	435.383
PG3	563.109

Nudo	2	7	3	8	9	5
$\alpha$	0	1	0	0.5	0.5	0
$\beta$	0	0	1	0	0.5	0.5
s	-0.33333	1.66667	-0.33333	0.66667	0.66667	-0.33333
t	-0.33333	-0.33333	1.66667	-0.33333	0.66667	0.66667
1-s-t	1.66667	-0.33333	-0.33333	0.66667	-0.33333	0.66667
Sx	-751.454	621.66	877.112	-64.897	749.386	62.829

### 3.- Viga empotrada en un extremo (V3)



A continuación se presentan unas tablas comparativas entre el programa comercial ANSYS Workbench v12.1.0 y el programa MEC-3pto implementado para esta tesis:

- Desplazamientos:

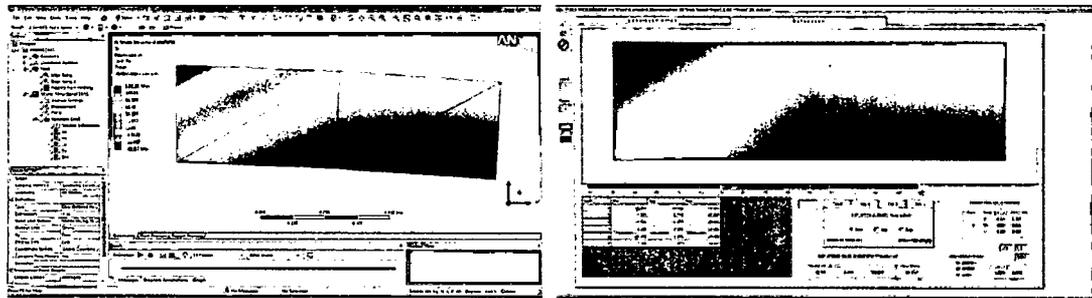
Nudo	ANSYS		MEC-3pto	
	x	y	x	y
1	0.0000	0.0000	0.0000	0.0000
2	0.0000	0.0000	0.0000	0.0000
3	0.0000	-0.0002	0.0000	-0.0002
4	0.0000	-0.0002	0.0000	-0.0002
5	0.0000	-0.0001	0.0000	-0.0001
6	0.0000	-0.0001	0.0000	-0.0001

- Esfuerzos calculados en el Punto de Gauss:

Elemento	ANSYS			MEC - 3pto		
	Sx	Sy	Sxy	Sx	Sy	Sxy
1				123.21	30.802	-126.074
2	NO PRESENTA ESTE RESULTADO			-123.21	-5.383	26.074
3				39.924	35.4	-76.046
4				-39.924	-14.373	-23.954

- Esfuerzos Alisados (Promedio Directo)

Nudo	ANSYS			MEC - 3pto		
	Sx	Sy	Sxy	Sx	Sy	Sxy
1	123.210	30.802	-126.070	123.21	30.802	-126.074
2	0.000	12.710	-50.000	0	12.71	-50
3	-39.924	-14.373	-23.954	-39.924	-14.373	-23.954
4	0.000	10.514	-50.000	0	10.514	-50
5	-41.070	5.215	-24.642	-41.07	5.215	-24.642
6	13.308	20.273	-58.682	13.308	20.273	-58.682

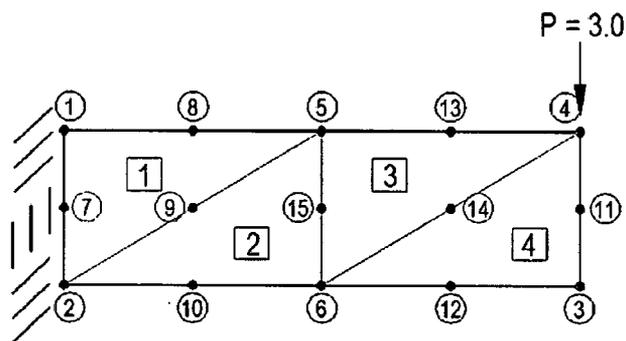


Esfuerzos Alisados Sx (ANSYS)

Esfuerzos Alisados Sx (MEC-3pto)

Comparativa de Esfuerzos

#### 4.- Viga empotrada en un extremo (V4)



A continuación se presentan unas tablas comparativas entre el programa comercial ANSYS Workbench v12.1.0 y el programa MEC-6pto.

- Desplazamientos:

Nudo	ANSYS		MEC-6pto	
	x	y	x	y
1	0.0000	0.0000	0.0000	0.0000
2	0.0000	0.0000	0.0000	0.0000
3	-0.0002	-0.0011	-0.0002	-0.0011
4	0.0002	-0.0011	0.0002	-0.0011
5	0.0002	-0.0003	0.0002	-0.0003
6	-0.0002	-0.0003	-0.0002	-0.0003
7	0.0000	0.0000	0.0000	0.0000
8	0.0001	-0.0001	0.0001	-0.0001
9	0.0000	-0.0001	0.0000	-0.0001
10	-0.0001	-0.0001	-0.0001	-0.0001
11	0.0000	-0.0011	0.0000	-0.0011
12	-0.0002	-0.0007	-0.0002	-0.0007
13	0.0002	-0.0007	0.0002	-0.0007
14	0.0000	-0.0007	0.0000	-0.0007
15	0.0000	-0.0003	0.0000	-0.0003

- Esfuerzos calculados en los Nudos:

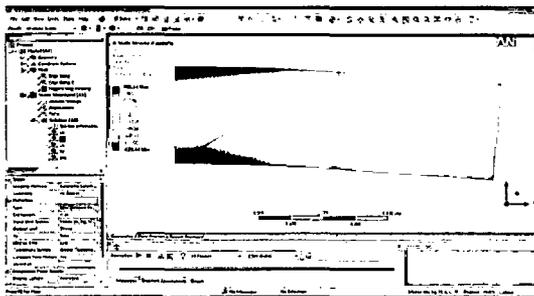
Nudo	ANSYS Elem(1)			ANSYS Elem(2)		
	Sx	Sy	Sxy	Sx	Sy	Sxy
1	901.24	225.31	-86.57			
2	-776.89	-194.22	-50.69	-869.99	22.78	-10.122
3						
4						
5	621.28	5.8191	-15.359	731.84	31.691	-84.328
6				-607.49	-51.564	-52.932
7	62.177	15.544	-68.63			
8	761.260	115.560	-50.964			
9	-77.804	-94.202	-33.024	-69.075	27.235	-47.225
10				-738.740	-14.392	-31.527
11						
12						
13						
14						
15				62.177	-9.937	-68.630

Nudo	ANSYS Elem(3)			ANSYS Elem(4)		
	Sx	Sy	Sxy	Sx	Sy	Sxy
1						
2						
3				-95.946	-34.541	-57.568
4	119.64	17.125	-20.169	227.1	-167.9	-71.787
5	402.46	-50.653	-46.961			
6	-271.32	32.478	-82.393	-381.94	112.15	-21.122
7						
8						
9						
10						
11				65.575	-101.220	-64.677
12				-238.950	38.806	-39.345
13	261.050	-16.764	-33.565			
14	-75.835	24.802	-51.281	-77.424	-27.873	-46.454
15	65.575	-9.087	-64.677			

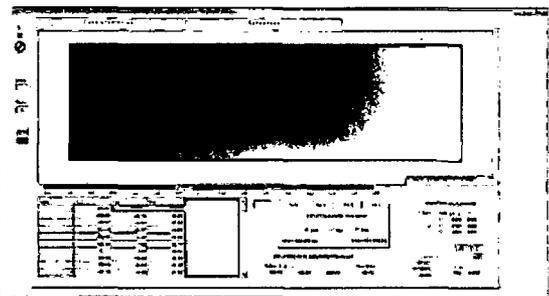
Los mismos valores de los esfuerzos en los nudos presenta el programa MEC-6pto

• Esfuerzos Alisados (Promedio Directo)

Nudo	ANSYS			MEC - 6pto		
	Sx	Sy	Sxy	Sx	Sy	Sxy
1	901.24	225.31	-86.57	901.243	225.311	-86.57
2	-823.44	-85.721	-30.406	-823.439	-85.721	-30.406
3	-95.946	-34.541	-57.568	-95.947	-34.541	-57.568
4	173.37	-75.387	-45.978	173.37	-75.387	-45.978
5	585.2	-4.3809	-48.883	585.195	-4.381	-48.883
6	-420.25	31.023	-52.149	-420.248	31.023	-52.149
7	38.902	69.795	-58.488	62.177	15.544	-68.63
8	743.22	110.46	-67.726	761.262	115.565	-50.964
9	-119.12	-45.051	-39.644	-73.439	-33.483	-40.125
10	-621.84	-27.349	-41.277	-738.738	-14.392	-31.527
11	38.712	-54.964	-51.773	65.575	-101.22	-64.677
12	-258.1	-1.7591	-54.858	-238.945	38.807	-39.345
13	379.28	-39.884	-47.43	261.055	-16.764	-33.565
14	-123.44	-22.182	-49.063	-76.630	-1.535	-48.868
15	82.474	13.321	-50.516	63.876	-9.512	-66.654



Esfuerzos Alisados Sx (ANSYS)

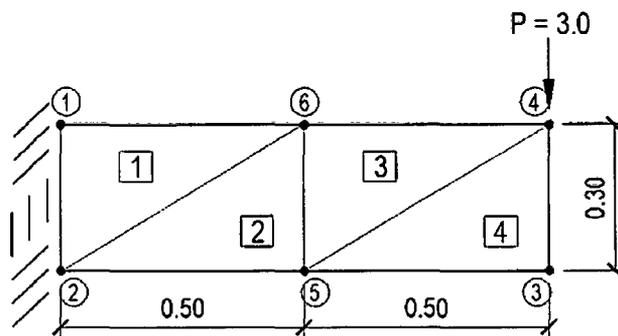


Esfuerzos Alisados Sx (MEC-6pto)

Comparativa de Esfuerzos

5.- Viga empotrada en un extremo (V5)

Se presenta una comparativa usando el método de Penalidad y el método de Multiplicadores de Lagrange para las condiciones de restricción, así como también la sensibilidad del valor  $K_{penalidad}$ .



A continuación se presentan unas tablas comparativas entre el programa MEC-3pto y MEC-3ptoLagrange implementados para esta tesis:

- Desplazamientos:

Nudo	MEC-3pto (Penalidad)				MEC-3pto-Lagrange	
	k=1.0+e05		k=1.0+e12		Multipl Lagrange	
	x	y	x	y	x	y
1	0.001	-0.0002	0.0000	0.0000	0	0
2	-0.001	-0.0001	0.0000	0.0000	0	0
3	-0.001	-0.0070	0.0000	-0.0002	0	-0.0002
4	0.001	-0.0070	0.0000	-0.0002	0	-0.0002
5	-0.001	-0.0036	0.0000	-0.0001	0	-0.0001
6	0.001	-0.0036	0.0000	-0.0001	0	-0.0001

- Esfuerzos Alisados (Promedio Directo)

Nudo	MEC-3pto (Penalidad)				MEC-3pto-Lagrange	
	k=1.0+e05		k=1.0+e12		Multipl Lagrange	
	Sx	Sy	Sx	Sy	Sx	Sy
1	125.455	-43.763	123.21	30.802	123.21	30.802
2	0	-24.915	0	12.71	0	12.71
3	-39.928	-14.374	-39.924	-14.373	-39.924	-14.373
4	0	10.452	0	10.514	0	10.514
5	-41.818	4.945	-41.07	5.215	-41.07	5.215
6	13.309	-4.851	13.308	20.273	13.308	20.273

```
// Método de Elemento Finito, Malla Triangular de 3 Nudos
// Método de Elementos de Contacto , Master-Slave , Nudo a Nudo
#include <time.h>
#include <string.h>
#include <stdio.h>
#include "stdlib.h"
#include <fstream.h>
#include "iostream.h"
#include <math.h>
#include <iomanip.h>
#define NDOFN 2 /* N° Grado de Libertad por Nudo */
#define NDIM 2 /* No. of dimensions */
#define NPROP 3 /* No. de propiedades por elemento 6+4=10 + 2 = 12 */
#define NNE 3 /* No. de nudos por elemento */
#define CLK_TCK 1000.0
typedef char cstring[20];
int NN,NE;
int aBanda;
double **coord=NULL;
int **support=NULL;
double **load=NULL;
int **connect=NULL;
double **prop=NULL;
int *elemtype=NULL;
int ndof,band,ndofe;
int LABEL[NDOFN*NNE];
double ESM[NDOFN*NNE][NDOFN*NNE];
double ERHS[NDOFN*NNE];
double D[3][3], B[3][6], DB[3][6], BDB[6][6];
double **ESF=NULL;
double **A=NULL;
double **A1=NULL;
double *X=NULL;
double *BB=NULL;
double *BBx=NULL;
double *BBy=NULL;
double *BBxy=NULL;
double **EsfANP=NULL;
double **EsfANMC=NULL;
double BIGg,BIGgkr, deltaKr;
double *RHS;
double *fonzi=NULL;
double *P=NULL;
double *disp=NULL;
double tol, tolc;
int nitem, nitemc;
int *pos=NULL;
double *DIAGONAL=NULL;
double ** FILA=NULL;
double ** COLUMNA=NULL;
double ** KRG=NULL;
double *U=NULL;
int min1(int i,int j);
int max1(int i,int j);
void GetInput(cstring s);
void SetBandSystem();
void GetConnect(int elemno);
void Assemble(int elemno);
void GetStiff(int elemno);
void ApplySupport();
void Solve();
void Soluc();
void PrintTime(char file[20]);
void PrintOut(char file[20]);
void PrintOutDesp(char file[20], int iterc);
```

```

void PrintSal(char file[20], int elemno);
void PrintSalM(char file[20]);
void PrintSalM1(char file[20]);
void PrintEsf(char file[20], int iterc);
void PrintEsfAlizado(char file[20]);
void ClearMemory();
void CalculoError();
void PrintError(char file[20]);
double error1_elem, error2_ener;
double *Param_Local=NULL;
int Kposi(int i, int j);
double &Kpos(int i, int j);
void ClearVariant();
void ludcmp(double **a, int n, int *indx, double *d);
void lubksb(double **a, int n, int *indx, double b[]);
void lubksb1(int n, int *indx);
int *indx;
#define TINY 1.0e-20
int iterc; //variable para contar las veces de contacto
int ncont; //numero de elementos de contacto
double time_ini, time_fin;
////////////////////////////////////
int main(int argc, char* argv[])
{
    int i,j,k;
    int qqqq;
    char file[20];
    char archivo[40];
    char archivo1[40];
    char archivo2[40];
    cout <<"INGRESE ARCHIVO DE ENTRADA: (incluya extension)";
    cin >> file;
    tolc=0.05; //tolerancia
    iterc=1; // inicia la iteracion con 1
    nitemc=1; //numero de iteraciones (por defecto es 1, pero si hay elemcontac vale 2, luego verifica)
    GetInput(file);
    //*****
    cout <<"INGRESE NOMBRE DE ARCHIVOS DE SALIDA (sin extension): ";
    cin >> file;
    strcpy( archivo, file);
    strcat( archivo, "_matrix.jg1");
    cout << archivo << "\n";
    strcpy( archivo1, file);
    strcat( archivo1, "_out.jgv");
    cout << archivo1 << "\n";
    strcpy( archivo2, file);
    strcat( archivo2, "_adap.adh");
    cout << archivo2 << "\n";
    //*****
    time_ini=clock()/CLK_TCK;
    SetBandSystem();
    // [1].- Inicio de las Iteraciones para el Contacto
    for (;)
    {
        ClearVariant();
        for (i=0;i<NE;i++)
        {
            Assemble(i);
            PrintSal(archivo, i);
        }
        PrintSalM(archivo);
        ApplySupport();
        PrintSalM1(archivo);
        aBanda=0;
        for(i=0;i<ndof;i++)

```

```

        if (aBanda<pos[i])
            aBanda=pos[i];

        Solve();
        PrintOutDesp(archivo1,iterc);
        double normc=0;
        double normdc=0;
        for(i=0;i<ndof;j++)
            normdc += (RHS[j]-disp[i])*(RHS[j]-disp[i]);
        normdc = sqrt(normdc);
        for(i=0;i<ndof;j++)
            disp[j] = RHS[j];
        for(i=0;i<ndof;j++)
            normc += (RHS[j]*RHS[j]);
        normc = sqrt(normc);
        double errc=(normdc/normc);
        printf("Error 1 : %f\n",normdc);
        printf("Error 2 : %f\n",normc);
        printf("Norma Desplaza: %f\n",errc);
        if (errc<=tolc)
        {
            cout << " OK!!! El error calculo es menor que el requerido \n";
            cout << " el programa finalizara \n";
            break;
        }
        if (iterc>=nitemc)
        {
            cout << " :{ numero de iteraciones mayor que el maximo definido \n";
            cout << " el programa finalizara \n";
            break;
        }
        iterc=iterc+1;
        cout <<"**** Fin de " << iterc-1 << ".º Iteracion \n";
        cin >>qqqq;
    }
    // [1].- Fin de las Iteraciones para el Contacto
    ncont=0;
    for (i=0;i<NE;j++)
        if (elemtype[j]==1)
            ncont=ncont+1;

    PrintOut(archivo);
    PrintEsf(archivo1,iterc);
    PrintEsfAlizado(archivo1);
    CalculoError();
    PrintError(archivo2);
    ClearMemory();
    time_fin=clock()/CLK_TCK;
    PrintTime(archivo1);
    cout << "Tiempo total: " << (time_fin-time_ini) << " segundos\n";
    cin >> file;
    return 0;
}
/////////////////////////////////////////////////////////////////
int min1(int i,int j)
{
    if(i<j) return i;
    return j;
}
/////////////////////////////////////////////////////////////////
int max1(int i,int j)
{
    if(i>j) return i;
    return j;
}
/////////////////////////////////////////////////////////////////
void GetInput(cstring s)

```

```

{
    ifstream inp;
    int i,j,k;
    int qqqqq;
    inp.open(s,ios::in);
    inp >> NN;
    printf("N de nudos : %d\n",NN);
    inp >> NE;
    printf("N de elementos : %d\n",NE);
    inp >> iterc;
    cout << "*****";
    coord = new double*[NN];
    for (i=0;i<NN;i++)
        coord[i]=new double[NDIM];
    support = new int*[NN];
    for (i=0;i<NN;i++)
        support[i]=new int[NDOFN];
    load = new double*[NN];
    for (i=0;i<NN;i++)
        load[i]=new double[NDOFN];
    for(i=0;i<NN;i++)
    {
        inp >> k;
        k--;
        for(j=0;j<NDIM;j++)
            inp >> coord[k][j];
        for(j=0;j<NDOFN;j++)
            inp >> support[k][j];
        for(j=0;j<NDOFN;j++)
            inp >> load[k][j];
    }
    connect = new int*[NE];
    for (i=0;i<NE;i++)
        connect[i]=new int[NNE];
    prop = new double*[NE];
    for (i=0;i<NE;i++)
        prop[i]=new double[NPROP];
    elemtype = new int [NE];
    for(i=0;i<NE;i++)
    {
        inp >> k;
        k--;
        for(j=0;j<NNE;j++)
        {
            inp >> connect[k][j];
            connect[k][j]--;
        }
        inp >> elemtype[k];
        for(j=0;j<NPROP;j++)
        {
            inp >> prop[k][j];
            deltaKr=prop[k][2];
        }
    }
    inp >> BIGg;
    inp >> BIGgkr;
}
////////////////////////////////////
void SetBandSystem()
{
    int i,j,k;
    int qqqqq;
    ndof = NDOFN*NN;
    ndofe = NDOFN*NNE;
    band = 0;
}

```

```

pos = new int[ndof]; // gdl x nn

printf("ndof: %d\n",ndof);
printf("ndofe: %d\n",ndofe);
cout <<"*****";
for(i=0;i<ndof;i++)
    pos[i]=0;
for(i=0;i<NE;i++)
{
    GetConnect(i);
    if (elemtype[i]==0 || elemtype[i]==2)
    {
        for(j=0;j<ndofe;j++)
        {
            for(k=0;k<j;k++)
            {
                int l1=LABEL[j];
                int l2=LABEL[k];
                int dif=abs(l2-l1);
                int l3=max1(l1,l2);
                if (pos[l3]<dif)
                    pos[l3]=dif;
            }
        }
    }
    else
    {
        nitemc=2; //por defecto=1iter, pero como hay elementcontact iterara 10 veces, minimo=2
        for(j=0;j<NDOFN*2;j++)
        {
            for(k=0;k<j;k++)
            {
                int l1=LABEL[j];
                int l2=LABEL[k];
                int dif=abs(l2-l1);
                int l3=max1(l1,l2);
                if (pos[l3]<dif)
                    pos[l3]=dif;
            }
        }
    }
}
DIAGONAL = new double [ndof];
COLUMNA = new double* [ndof];
FILA = new double* [ndof];
for(i=0;i<ndof;i++)
{
    COLUMNA[i]=NULL;
    FILA[i]=NULL;
    j=pos[i];
    if (j>0)
    {
        COLUMNA[i]= new double[j];
        FILA[i]= new double[j];
    }
}
for(i=0;i<ndof;i++)
{
    DIAGONAL[i]=0.0;
    for(j=0;j<pos[i];j++)
    {
        COLUMNA[i][j]=0.;
        FILA[i][j]=0.;
    }
}

```

```

disp = new double [ndof];
for(i=0;i<ndof;i++)
    disp[i] = 0.0;
RHS = new double [ndof];
for(i=0;i<ndof;i++)
    RHS[i] = 0.0;
P = new double [NE];
for(i=0;i<NE;i++)
    P[i] = 0;
}
////////////////////////////////////
void GetConnect(int elemno) // LABEL[0] hasta LABEL[5] son los 2x3=6gdl por elemento
{
    int i,j;
    int aux;
    aux=1;
    if (elemtype[elemno]==0 || elemtype[elemno]==2)
    {
        for(i=0;i<NNE;i++)
            for(j=0;j<NDOFN;j++)
                LABEL[i*NDOFN+j] = connect[elemno][i]*NDOFN+j);
    }
    else
    {
        for(i=0;i<2;i++)
            for(j=0;j<NDOFN;j++)
                LABEL[i*NDOFN+j] = connect[elemno][i]*NDOFN+j);
    }
}
////////////////////////////////////
void Assemble(int elemno)
{
    int i,j;
    GetConnect(elemno);
    GetStiff(elemno);
    if (elemtype[elemno] == 0 || elemtype[elemno] == 2)
    {
        ndofe = NDOFN * NNE;
    }
    else
    {
        ndofe = NDOFN * 2;
    }
    for(i=0;i<ndofe;i++)
    {
        DIAGONAL[LABEL[i]] += ESM[i][i];
        RHS[LABEL[i]] += ERHS[i];
    }
    for(i=1;i<ndofe;i++)
    {
        for(j=0;j<i;j++)
        {
            if (LABEL[j]<=LABEL[i])
                FILA[LABEL[i]][abs(LABEL[j]-LABEL[i])-1] += ESM[i][j];
            if (LABEL[j]>LABEL[i])
                FILA[LABEL[j]][abs(LABEL[j]-LABEL[i])-1] += ESM[j][i];
        }
    }
    for(j=1;j<ndofe;j++)
    {
        for(i=0;i<j;i++)
        {
            if(LABEL[i]<=LABEL[j])
                COLUMNA[LABEL[j]][abs(LABEL[i]-LABEL[j])-1] += ESM[j][i];
            if(LABEL[i]>LABEL[j])

```

```

        COLUMNA[LABEL[i]][abs(LABEL[i]-LABEL[j])-1] += ESM[i][j];
    }
}
ndofe = NDOFN * NNE;
}
///////////////////////////////////////////////////////////////////
void GetStiff(int elemno)
{
    int i,j,k,kode;
    // variables para FEM
    double coef1,coef2;
    double x1,y1,x2,y2,x3,y3,x13,y23,x23,y13,y31,y12,x32,x21;
    double detJ,areas;
    double xx1,yy1,xx2,yy2;
    double xx1f,yy1f,xx2f,yy2f,vectnxf,vectnyf;
    double vectnx,vectny,l0,lf,kresorte;
    kode=elemtype[elemno];
    if(kode==0 || kode==2)
    {
        // prop[elemno][0] = E
        // prop[elemno][1] = Espesor
        // prop[elemno][2] = Poisson
        // prop[elemno][0] = nx
        // prop[elemno][1] = ny
        // prop[elemno][2] = kresorte
        // Inicio Matrix del Elemento Finito
        // Matrix "D"
        if(kode==0) //Para el caso de TENSION PLANA
        {
            coef1=prop[elemno][0]/(1-(prop[elemno][2]*prop[elemno][2]));
            D[0][0]=coef1*1;
            D[0][1]=coef1*prop[elemno][2];
            D[0][2]=0;
            D[1][0]=coef1*prop[elemno][2];
            D[1][1]=coef1*1;
            D[1][2]=0;
            D[2][0]=0;
            D[2][1]=0;
            D[2][2]=coef1*(1-prop[elemno][2])/2;
        }
        if(kode==2) //Para el caso de DEFORMACION PLANA
        {
            coef1=prop[elemno][0]*(1-prop[elemno][2])/(1+prop[elemno][2])/(1-2*prop[elemno][2]);
            D[0][0]=coef1*1;
            D[0][1]=coef1*prop[elemno][2]/(1-prop[elemno][2]);
            D[0][2]=0;
            D[1][0]=coef1*prop[elemno][2]/(1-prop[elemno][2]);
            D[1][1]=coef1*1;
            D[1][2]=0;
            D[2][0]=0;
            D[2][1]=0;
            D[2][2]=prop[elemno][0]/2/(1+prop[elemno][2]);
        }
        // Determinante Jacobiano
        x1=coord[connect[elemno][0]][0];
        y1=coord[connect[elemno][0]][1];
        x2=coord[connect[elemno][1]][0];
        y2=coord[connect[elemno][1]][1];
        x3=coord[connect[elemno][2]][0];
        y3=coord[connect[elemno][2]][1];
        x13=x1-x3;
        y23=y2-y3;
        x23=x2-x3;
        y13=y1-y3;
        detJ=x13*y23-x23*y13;
    }
}

```

```

if (detJ<0)
    detJ=(detJ)*(-1);
areas=0.5*detJ;
coef2=1/detJ; //2.Area=Jacobiano
y12=y1-y2;
y31=y3-y1;
x32=x3-x2;
x21=x2-x1;
// Matrix "B"
B[0][0]=coef2*y23;
B[0][1]=0;
B[0][2]=coef2*y31;
B[0][3]=0;
B[0][4]=coef2*y12;
B[0][5]=0;
B[1][0]=0;
B[1][1]=coef2*x32;
B[1][2]=0;
B[1][3]=coef2*x13;
B[1][4]=0;
B[1][5]=coef2*x21;
B[2][0]=coef2*x32;
B[2][1]=coef2*y23;
B[2][2]=coef2*x13;
B[2][3]=coef2*y31;
B[2][4]=coef2*x21;
B[2][5]=coef2*y12;
for (i=0;i<3;i++)
    for (j=0;j<6;j++)
        DB[i][j]=0;
for (i=0;i<6;i++)
    for (j=0;j<6;j++)
        BDB[i][j]=0;
// Matrix DB = D x B
for (i=0;i<3;i++)
    for (j=0;j<6;j++)
        for (k=0;k<3;k++)
            DB[i][j]=DB[i][j]+D[i][k]*B[k][j];
// Matrix B'DB = B' x DB
for (i=0;i<6;i++)
    for (j=0;j<6;j++)
        for (k=0;k<3;k++)
            BDB[i][j]=BDB[i][j]+B[k][i]*DB[k][j];
// Matrix K = espesor x area x B'DB
for (i=0;i<6;i++)
    for (j=0;j<6;j++)
        ESM[i][j]=prop[elemno][1]*areas*BDB[i][j];
// Fin Matrix del Elemento Finito
for (i=0;i<6;i++)
    ERHS[i]=0.0;
}
else
{
    // para la matrix de rigidez del elemento
    xx1=coord[connect[elemno][0]][0];
    yy1=coord[connect[elemno][0]][1];
    xx2=coord[connect[elemno][1]][0];
    yy2=coord[connect[elemno][1]][1];
    l0=sqrt((xx2-xx1)*(xx2-xx1)+(yy2-yy1)*(yy2-yy1));
    vectnx=(xx2-xx1)/l0;
    vectny=(yy2-yy1)/l0;
    if ((xx2-xx1)==0)
        if ((yy2-yy1)==0)
            {
                vectnx=prop[elemno][0]/sqrt(prop[elemno][0]*prop[elemno][0]+prop[elemno][1]*prop[elemno][1]);
            }
}

```

```

vectny=prop[elemno][1]/sqrt(prop[elemno][0]*prop[elemno][0]+prop[elemno][1]*prop[elemno][1]);
}
xx1f=disp[connect[elemno][0]*2]+xx1;
yy1f=disp[connect[elemno][0]*2+1]+yy1;
xx2f=disp[connect[elemno][1]*2]+xx2;
yy2f=disp[connect[elemno][1]*2+1]+yy2;
lf=sqrt((xx2f-xx1f)*(xx2f-xx1f)+(yy2f-yy1f)*(yy2f-yy1f));
if (iterc == 1)
    kresorte=0;
else
{
    kresorte=prop[elemno][2];
    if (l0>=0)
    {
        if (l0<=lf)
            //kresorte=BIGgkr;
        if (0<=lf && lf<l0)
            kresorte=0;
        if (lf<0)
            kresorte=0;
    }
    if (l0<0)
    {
        if (lf<=l0)
            //kresorte=BIGgkr;
        if (l0<=lf && lf<=0)
            kresorte=0;
        if (0<lf)
            kresorte=0;
    }
}
for (i=0;i<6;i++)
    for (j=0;j<6;j++)
        ESM[i][j]=0;
ESM[0][0]=kresorte*(-vectnx)*(-vectnx);
ESM[0][1]=kresorte*(-vectnx)*(-vectny);
ESM[0][2]=-ESM[0][0];
ESM[0][3]=-ESM[0][1];
ESM[1][0]=ESM[0][1];
ESM[1][1]=kresorte*(-vectny)*(-vectny);
ESM[1][2]=-ESM[1][0];
ESM[1][3]=-ESM[1][1];
ESM[2][0]=ESM[0][2];
ESM[2][1]=ESM[1][2];
ESM[2][2]=-ESM[2][0];
ESM[2][3]=-ESM[2][1];
ESM[3][0]=ESM[0][3];
ESM[3][1]=ESM[1][3];
ESM[3][2]=-ESM[0][3];
ESM[3][3]=-ESM[1][3];
// para la matrix de fuerza del elemento: F=K.n.l; l=difer(nudos)
ERHS[0]=kresorte*(-vectnx)*l0;
ERHS[1]=kresorte*(-vectny)*l0;
ERHS[2]=-ERHS[0];
ERHS[3]=-ERHS[1];
ERHS[4]=0;
ERHS[5]=0;
}
}
////////////////////////////////////
void ApplySupport()
{
    int i,j,pos;
    for(i=0;i<NN;i++)
        for(j=0;j<NDOFN;j++)

```

```

        {
            pos = i*NDOFN+j;
            if(support[i][j]==1)
            {
                DIAGONAL[pos] +=BIGg;
                RHS[pos] += BIGg*load[i][j];
            }
            else
            {
                RHS[pos] += load[i][j];
            }
        }
    }
    ////////////////////////////////////////////////////
    void Solve()
    {
        int i,j,k,kk;
        double pivot;
        double suma;
        for(i=1;i<ndof;i++)
        {
            if(pos[i] != 0)
            {
                for(j=pos[i];j>0;j--)
                {
                    if(FILA[i][j]-1 != 0)
                    {
                        pivot=(FILA[i][j]-1)/DIAGONAL[i-j];
                        FILA[i][j]-1 = 0;
                        for (k=1;k<j;k++)
                            if(pos[(i-j)+k] >= k)
                                FILA[i][j]-1-k += -COLUMNA[i-j+k][k-1]*pivot;
                        if(COLUMNA[i][j]-1 != 0)
                            DIAGONAL[i] += -COLUMNA[i][j]-1*pivot;
                        kk=(i-j)+aBanda;
                        if(kk>ndof)
                            kk=ndof-1;
                        kk=kk-i;
                        for(k=0;k<kk;k++)
                        {
                            if ((k+1) < pos[i+(k+1)])
                                if(j<pos[i+k+1]-k)
                                    COLUMNA[i+k+1][k] += -COLUMNA[i+k+1][j+k]*pivot;
                        }
                        RHS[i] += -RHS[i-j]*pivot;
                    }
                }
            }
        }
        for(i=ndof-1;i>=0;i--)
        {
            double Kij;
            suma=0.0;
            for(j=i+1;j<ndof;j++)
                if(Kpos(i,j))
                {
                    Kij=Kpos(i,j);
                    suma += Kij*RHS[j];
                }
            RHS[i] = (RHS[i]-suma)/Kpos(i,i);
        }
    }
    ////////////////////////////////////////////////////
    void PrintOut(char file[20])
    {

```

```

int i,j;
ofstream out;
out.open(file,ios::app);
out << " \n";
out << "**** RESULTADOS FINALES DE MATRICES AUXILIARES ****" << " \n";
out << "MATRIZ DIAGONAL" << " \n";
for(i=0;j<ndof;i++)
    out << DIAGONAL[i] << "\t";
out << " \n";
out << " \n";
out << "MATRIZ FILA" << " \n";
for(i=1;j<ndof;i++)
{
    out << "posicion: " << i << " , Tiene [" << pos[i] << "]" \n";
    for(j=0;j<pos[i];j++)
        out << FILA[i][j] << "\t";
    out << "\n";
}
out << "\n";
out << "MATRIZ COLUMNA" << " \n";
for(i=1;j<ndof;i++)
{
    out << "posicion: " << i << " , Tiene [" << pos[i] << "]" \n";
    for(j=0;j<pos[i];j++)
        out << COLUMNA[i][j] << "\t";
    out << "\n";
}
out << "\n";
out << "MATRIZ RHS:" << " \n";
for(i=0;j<ndof;i++)
    out << "RHS [" << i << "]" : "<< RHS[i] << " \n";
out << "\n";
}
////////////////////////////////////
void PrintOutDesp(char file[20], int iterc)
{
    int i,j;
    fstream mistream;
    mistream.open(file,ios::in|ios::app);
    mistream << " \n";
    mistream << "ELEMENTO TRIANGULAR DE 3 PUNTOS - PENALIDAD" << "\n";
    mistream << " \n";
    mistream << "DESPLAZAMIENTO : Iteracion [" << iterc << "]" \n";
    mistream << "NUDO" << "\t";
    mistream << setw(14) << "UX" << "\t";
    mistream << setw(14) << "UY" << "\n";
    for (i=0;i<NN;i++)
    {
        mistream << i+1 << "\t";
        for (j=0;j<NDOFN;j++)
            mistream << setw(15) << RHS[i*NDOFN+j] << "\t";
        mistream << "\n";
    }
    for (i=0;i<NE;i++)
    {
        if (elemtipe[i]==1)
        {
            mistream << setw(15) << i << "\t";
            mistream << setw(15) << prop[i][2] << "\n";
        }
    }
    mistream << "\n";
}
////////////////////////////////////
void ClearMemory()

```

```

{
    int i;
    for (i=0;i<NN;i++)
        delete [] coord[i];
    delete [] coord;
    for (i=0;i<NN;i++)
        delete [] support[i];
    delete [] support;
    for (i=0;i<NN;i++)
        delete [] load[i];
    delete [] load;
    for (i=0;i<NE;i++)
        delete [] connect[i];
    delete [] connect;
    for (i=0;i<NE;i++)
        delete [] prop[i];
    delete [] prop;
    delete [] elemtype;
    delete [] DIAGONAL;
    for(i=0;j<ndof;i++)
    {
        if (COLUMNA[i]!=NULL)
            delete COLUMNA[i];
        if (FILA[i]!=NULL)
            delete FILA[i];
    }
    delete [] COLUMNA;
    delete [] FILA;
    for (i=0;i<NE;i++)
        delete [] ESF[i];
    delete [] ESF;
}
////////////////////////////////////
double &Kpos(int i, int j)
{
    if (i==j)
        return DIAGONAL[i];
    else if (j>i)
        return COLUMNA[j][j-i-1];
    else
        return FILA[i][i-j-1];
}
int Kposi(int i, int j)
{
    if (i==j)
        return 1;
    else if (j>i)
    {
        if ((j-i)>pos[j])
            return 0;
        else
            return 1;
    }
    else
    {
        if ((i-j)>pos[i])
            return 0;
        else
            return 1;
    }
}
////////////////////////////////////
void ClearVariant()
{
    int i,j;

```

```

        for(i=0;i<ndof;i++)
        {
            DIAGONAL[i]=0.0;
            for(j=0;j<pos[i];j++)
            {
                COLUMNA[i][j]=0.;
                FILA[i][j]=0.;
            }
        }
        for(i=0;i<ndof;i++)
            RHS[i] = 0.0;
    }
    //////////////////////////////////////
    void PrintSal(char file[20], int elemno)
    {
        int i,j,kode;
        fstream mistream;
        mistream.open(file,ios::in|ios::app);
        mistream << " \n";
        kode=elemtype[elemno];
        if(kode==0 || kode==2)
        {
            mistream << "MATRIZ DB : [" << elemno+1 << "] \n";
            for (i=0;i<3;i++)
            {
                for (j=0;j<6;j++)
                    mistream << setw(12) << DB[i][j] << "\t";
                mistream << "\n";
            }
            mistream << "\n";
            mistream << "MATRIZ BDB : [" << elemno+1 << "] \n";
            for (i=0;i<6;i++)
            {
                for (j=0;j<6;j++)
                    mistream << setw(12) << BDB[i][j] << "\t";
                mistream << "\n";
            }
            mistream << "\n";
            mistream << "MATRIZ RIGIDEZ : [" << elemno+1 << "] \n";
            for (i=0;i<6;i++)
            {
                for (j=0;j<6;j++)
                    mistream << setw(12) << ESM[i][j] << "\t";
                mistream << "\n";
            }
            mistream << "\n";
            mistream << "\n";
        }
        else
        {
            mistream << "MATRIZ RIGIDEZ RESORTE: [" << elemno+1 << "] \n";
            for (i=0;i<4;i++)
            {
                for (j=0;j<4;j++)
                    mistream << setw(12) << ESM[i][j] << "\t";
                mistream << "\t";
                mistream << setw(12) << ERHS[i] << "\n";
            }
            mistream << "\n";
            mistream << "\n";
        }
    }
    //////////////////////////////////////
    void PrintSalM(char file[20])
    {

```

```

int i,j;
ofstream out;
out.open(file,ios::app);

out << "MATRIZ FILA : " << "\n";
for (i=0;i<ndof;j++)
{
    out << pos[i] << "\t";
    for (j=0;j<pos[i];j++)
        out << FILA[i][j] << "\t";
    out << "\n";
}
out << "\n";
out << "\n";
out << "MATRIZ COLUMNA : " << "\n";
for (i=0;i<ndof;j++)
{
    out << pos[i] << "\t";
    for (j=0;j<pos[i];j++)
        out << COLUMNA[i][j] << "\t";
    out << "\n";
}
out << "\n";
out << "\n";
out << "MATRIZ DIAGONAL sin la Carga del RESORTE: " << "\n";
for(i=0;i<ndof;j++)
    out << DIAGONAL[i] << "\t";
out << "\n";
out << "\n";
out << "MATRIZ FUERZA Inicial: " << "\n";
for(i=0;i<ndof;j++)
    out << RHS[i] << "\t";
out << "\n";
out << "\n";
}
////////////////////////////////////
void PrintSalM1(char file[20])
{
    int i;
    ofstream out;
    out.open(file,ios::app);

    out << "MATRIZ DIAGONAL con la Carga del RESORTE: " << "\n";
    for(i=0;i<ndof;j++)
        out << DIAGONAL[i] << "\t";
    out << "\n";
    out << "\n";
    out << "MATRIZ FUERZA Final: " << "\n";
    for(i=0;i<ndof;j++)
        out << RHS[i] << "\t";
    out << "\n";
    out << "\n";
}
////////////////////////////////////
void PrintEsf(char file[20], int iterc)
{
    int i,j,k;
    double sxx[3],syy[3],sxy[3];
    fstream mistream;
    mistream.open(file,ios::in|ios::app);
    cout << "***** Inicia calculo de Esfuerzos \n";
    mistream << " \n";
    mistream << " \n";
    mistream << "**** TABLA DE ESFUERZO EN CADA ELEMENTO ***\n";
    mistream << "\t \t \t Sxx\t \t Syy\t \t Sxy\n";
}

```

```

ESF = new double*[NE];
for (i=0;i<NE;i++)
    ESF[i]=new double[3];
for (i=0;i<NE;i++)
{
    if (elemtype[i]==0 || elemtype[i]==2)
    {
        GetConnect(i);
        GetStiff(i);
        mistream << "Elem " << i+1 << " :";
        for(j=0;j<NNE;j++)
        {
            ESF[i][j]=0.0;
            for (k=0;k<ndofe;k++)
                ESF[i][j]=DB[j][k]*RHS[LABEL[k]]+ESF[i][j];
            mistream << setw(15) << ESF[i][j] <<" ";
        }
        mistream << "\n";
    }
}
cout <<"***** Fin de calculo de Esfuerzos \n";
}
////////////////////////////////////
void PrintEsfAlizado(char file[20])
{
    int i,j,k,l;
    int conta;
    int jj,kk;
    int NEN[10][2]; //numero de elementos en un nudo, 10 es el valor estimado y 2 es valor de elem y pos
    double x1,x2,x3,y1,y2,y3,areas,detJ;
    double M1,M2,M3;
    double mini0,mini1,mini2,maxi0,maxi1,maxi2;
    int elema0,elema1,elema2,elemi0,elemi1,elemi2;
    EsfANP = new double*[NN];
    for (i=0;i<NN;i++)
        EsfANP[i]=new double[3];
    EsfANMC = new double*[NN];
    for (i=0;i<NN;i++)
        EsfANMC[i]=new double[3];
    A = new double*[NN+ncont+1];
    for (i=0;i<=NN+ncont;i++)
        A[i]=new double[NN+ncont+1];
    A1 = new double*[NN+ncont];
    for (i=0;i<NN+ncont;i++)
        A1[i]=new double[NN+ncont];
    BB=new double[NN+ncont+1];
    BBx=new double[NN+ncont];
    BBy=new double[NN+ncont];
    BBxy=new double[NN+ncont];
    X=new double[NN+ncont];
    for (i=0;i<NN+ncont;i++)
    {
        BB[i+1]=0;
        BBx[i]=0;
        BBy[i]=0;
        BBxy[i]=0;
        X[i]=0;
        for (j=0;j<NN+ncont;j++)
        {
            A[i+1][j+1]=0;
            A1[i][j]=0;
        }
    }
    fstream mistream;
}

```

```

mistream.open(file,ios::in|ios::app);
cout <<"**** Inicia calculo de Esfuerzos Alizados : Metodo Promedio \n";
for (i=0;i<NN;i++)
{
    conta=0;
    EsfANP[i][0]=0; //para Sxx
    EsfANP[i][1]=0; //para Syy
    EsfANP[i][2]=0; //para Sxy
    for (j=0;j<10;j++)
    {
        NEN[j][0]=0;
        NEN[j][1]=0;
    }
    for (j=0;j<NE;j++)
    {
        if (elemtype[j]==0 || elemtype[j]==2)
        {
            for (k=0;k<NNE;k++)
            {
                if (connect[j][k]==i)
                {
                    NEN[conta][0]=j; //elemento
                    NEN[conta][1]=k; //una posición de los 3 nudos
                    conta=conta+1;
                }
            }
        }
    }
    for (l=0;l<conta;l++)
    {
        EsfANP[i][0]=EsfANP[i][0]+ESF[NEN[l][0]][0];
        EsfANP[i][1]=EsfANP[i][1]+ESF[NEN[l][0]][1];
        EsfANP[i][2]=EsfANP[i][2]+ESF[NEN[l][0]][2];
    }
    EsfANP[i][0]=EsfANP[i][0]/conta;
    EsfANP[i][1]=EsfANP[i][1]/conta;
    EsfANP[i][2]=EsfANP[i][2]/conta;
}
cout <<"**** Fin de calculo de Esfuerzos Alizados : Metodo Promedio \n";
cout <<"**** Inicia calculo de Esfuerzos Alizados : Metodo Minimo Cuadrado \n";
conta=0;
for (i=0;i<NE;i++)
{
    if (elemtype[i]==0 || elemtype[i]==2)
    {
        x1=coord[connect[i][0]][0];
        y1=coord[connect[i][0]][1];
        x2=coord[connect[i][1]][0];
        y2=coord[connect[i][1]][1];
        x3=coord[connect[i][2]][0];
        y3=coord[connect[i][2]][1];
        detJ=(x1-x3)*(y2-y3)-(x2-x3)*(y1-y3);
        if (detJ<0)
            detJ=(detJ)*(-1);
        areas=0.5*detJ;
        M1=areas/6;
        M2=areas/12;
        M3=areas/3;
        A1[connect[i][0]][connect[i][0]]=M1+A1[connect[i][0]][connect[i][0]];
        A1[connect[i][0]][connect[i][1]]=M2+A1[connect[i][0]][connect[i][1]];
        A1[connect[i][0]][connect[i][2]]=M2+A1[connect[i][0]][connect[i][2]];
        A1[connect[i][1]][connect[i][0]]=M2+A1[connect[i][1]][connect[i][0]];
        A1[connect[i][1]][connect[i][1]]=M1+A1[connect[i][1]][connect[i][1]];
        A1[connect[i][1]][connect[i][2]]=M2+A1[connect[i][1]][connect[i][2]];
        A1[connect[i][2]][connect[i][0]]=M2+A1[connect[i][2]][connect[i][0]];
    }
}

```

```

A1[connect[i][2]][connect[i][1]]=M2+A1[connect[i][2]][connect[i][1]];
A1[connect[i][2]][connect[i][2]]=M1+A1[connect[i][2]][connect[i][2]];
BBx[connect[i][0]]=M3*ESF[i][0]+BBx[connect[i][0]];
BBx[connect[i][1]]=M3*ESF[i][0]+BBx[connect[i][1]];
BBx[connect[i][2]]=M3*ESF[i][0]+BBx[connect[i][2]];
BBY[connect[i][0]]=M3*ESF[i][1]+BBY[connect[i][0]];
BBY[connect[i][1]]=M3*ESF[i][1]+BBY[connect[i][1]];
BBY[connect[i][2]]=M3*ESF[i][1]+BBY[connect[i][2]];
BBxy[connect[i][0]]=M3*ESF[i][2]+BBxy[connect[i][0]];
BBxy[connect[i][1]]=M3*ESF[i][2]+BBxy[connect[i][1]];
BBxy[connect[i][2]]=M3*ESF[i][2]+BBxy[connect[i][2]];
}
}
for (i=0;i<NN;i++)
{
    BB[i+1]=BBx[i];
    for (j=0;j<NN;j++)
        A[i+1][j+1]=A1[i][j];
}
Soluc();
for (i=0;i<NN;i++)
    EsfANMC[i][0]=BB[i+1]; //esfuerzo en Sxx
for (i=0;i<NN;i++)
{
    BB[i+1]=BBY[i];
    for (j=0;j<NN;j++)
        A[i+1][j+1]=A1[i][j];
}
Soluc();
for (i=0;i<NN;i++)
    EsfANMC[i][1]=BB[i+1]; //esfuerzo en Syy
for (i=0;i<NN;i++)
{
    BB[i+1]=BBxy[i];
    for (j=0;j<NN;j++)
        A[i+1][j+1]=A1[i][j];
}
Soluc();
for (i=0;i<NN;i++)
    EsfANMC[i][2]=BB[i+1]; //esfuerzo en Sxy

cout <<"**** Fin de calculo de Esfuerzos Alizados : Metodo Minimo Cuadrado \n";
mistream << "\n";
mistream << "\n";
mistream << "**** ESFUERZOS ALIZADOS EN EL NUDO : Métodos: Promedio , Minimo Cuadrado ****" << "\n";
mistream << "\n";
mistream << "\t \t \t Método Promediolt \t \t \t Método Minimo Cuadrado\n";
mistream << "\t \t \t Sxx\t \t Syy\t \t Sxy\t \t \t Sxx\t \t Syy\t \t Sxy\n";
for (i=0;i<NN;i++)
{
    mistream << "Nudo " << i+1 << " :";
    mistream << setw(15) << EsfANP[i][0] << "\t";
    mistream << setw(15) << EsfANP[i][1] << "\t";
    mistream << setw(15) << EsfANP[i][2] << "\t";
    mistream << "\t";
    mistream << setw(15) << EsfANMC[i][0] << "\t";
    mistream << setw(15) << EsfANMC[i][1] << "\t";
    mistream << setw(15) << EsfANMC[i][2] << "\t";
}
}
///////////////////////////////////////////////////////////////////
void Soluc()
{
    double d;
    int nt;

```

```

nt=NN;
indx=new int[nt];
ludcmp(A,nt,indx,&d);
lubksb1(nt,indx);
}
//*****
void ludcmp(double **a, int n, int *indx, double *d)
{
    int i,imax,j,k;
    float big,dum,sum,temp;
    float *vv;
    vv=new float[n+1];
    for (i=1;i<=n;i++)
        vv[i]=1;
    *d=1.0;
    for (i=1;i<=n;i++) {
        big=0.0;
        for (j=1;j<=n;j++)
            if ((temp=fabs(a[i][j])) > big)
                big=temp;
        if (big == 0.0)
            cout<<"Singular matrix in routine ludcmp \n";
        vv[i]=1.0/big;
    }
    for (j=1;j<=n;j++)
    {
        for (i=1;i<j;i++) {
            sum=a[i][j];
            for (k=1;k<i;k++) sum -= a[i][k]*a[k][j];
            a[i][j]=sum;
        }
        big=0.0;
        for (i=j;i<=n;i++)
        {
            sum=a[i][j];
            for (k=1;k<j;k++)
                sum -= a[i][k]*a[k][j];
            a[i][j]=sum;
            if ( (dum=vv[i]*fabs(sum)) >= big) {
                big=dum;
                imax=i;
            }
        }
        if (j != imax) {
            for (k=1;k<=n;k++) {
                dum=a[imax][k];
                a[imax][k]=a[j][k];
                a[j][k]=dum;
            }
            *d = -*d;
            vv[imax]=vv[j];
        }
        indx[j]=imax;
        if (a[j][j] == 0.0)
            a[j][j]=TINY;
        if (j != n) {
            dum=1.0/(a[j][j]);
            for (i=j+1;i<=n;i++) a[i][j] *= dum;
        }
    }
}
//*****
void lubksb1(int n, int *indx)
{
    int i,ii=0,ip,j;

```

```

float sum;
for (i=1;i<=n;i++)
{
    ip=indx[i];
    sum=BB[ip];
    BB[ip]=BB[i];
    if (ii)
        for (j=ii;j<=i-1;j++) sum -= A[i][j]*BB[j];
    else if (sum) ii=i;
    BB[i]=sum;
}
for (i=n;i>=1;i--) {
    sum=BB[i];
    for (j=i+1;j<=n;j++) sum += A[i][j]*BB[j];
    BB[i]=sum/A[i][i];
}
}
//*****
void CalculoError()
{
    int i,j;
    double Dinv[3][3];
    double factor;
    double Esfx,Esfy,Esfx,Esfy[NNE];
    double norma_esf;
    double *Norma_Elem=NULL;
    double *Norma_Energia=NULL;
    double EsfD1,EsfD2,EsfD3;
    double x1,y1,x2,y2,x3,y3,area;
    Norma_Elem=new double[NE];
    Norma_Energia=new double[NE];
    Param_Local=new double[NE];
    for (i=0;i<3;i++)
    {
        for (j=0;j<3;j++)
            Dinv[i][j]=0;
    }
    norma_esf = 0;
    for (i=0;i<NE;i++)
    {
        if (elemtype[i]==0 || elemtype[i]==2)
        {
            x1 = coord[connect[i][0]][0];
            y1 = coord[connect[i][0]][1];
            x2 = coord[connect[i][1]][0];
            y2 = coord[connect[i][1]][1];
            x3 = coord[connect[i][2]][0];
            y3 = coord[connect[i][2]][1];
            area = (x1-x3)*(y2-y3)-(x2-x3)*(y1-y3);
            if (area<0)
                area = (area)*(-1);
            area = 0.5*area;
            if(elemtype[i]==0)
            {
                factor = 1/prop[i][0];
                Dinv[0][0] = factor*1;
                Dinv[0][1] = factor*(-1*prop[i][2]);
                Dinv[1][0] = Dinv[0][1];
                Dinv[1][1] = Dinv[0][0];
                Dinv[2][2] = factor*(2)*(1+prop[i][2]);
            }
            if(elemtype[i]==2)
            {
                factor = (prop[i][2]+1)/prop[i][0];
                Dinv[0][0] = factor*(1-prop[i][2]);
            }
        }
    }
}

```

```

        Dinv[0][1] = factor*(-1)*(prop[i][2]);
        Dinv[1][0] = Dinv[0][1];
        Dinv[1][1] = Dinv[0][0];
        Dinv[2][2] = factor*2;
    }

    norma_esf=0;
    for (j=0;j<NNE;j++)
    {
        Esfx = (EsfANP[connect[i][j]][0] - Esf[i][0]);
        Esfy = (EsfANP[connect[i][j]][1] - Esf[i][1]);
        Esfxy = (EsfANP[connect[i][j]][2] - Esf[i][2]);
        EsfD1 = Esfx*Dinv[0][0] + Esfy*Dinv[0][1] + Esfxy*Dinv[0][2];
        EsfD2 = Esfx*Dinv[1][0] + Esfy*Dinv[1][1] + Esfxy*Dinv[1][2];
        EsfD3 = Esfx*Dinv[2][0] + Esfy*Dinv[2][1] + Esfxy*Dinv[2][2];
        Esf[j] = EsfD1*Esfx + EsfD2*Esfy + EsfD3*Esfxy;
        norma_esf = norma_esf + Esf[j];
    }

    Norma_Elem[i] = norma_esf*prop[i][1]*area;
    error1_elem = error1_elem + Norma_Elem[i];
    norma_esf=0;
    for (j=0;j<NNE;j++)
    {
        Esfx = EsfANP[connect[i][j]][0];
        Esfy = EsfANP[connect[i][j]][1];
        Esfxy = EsfANP[connect[i][j]][2];
        EsfD1 = Esfx*Dinv[0][0] + Esfy*Dinv[0][1] + Esfxy*Dinv[0][2];
        EsfD2 = Esfx*Dinv[1][0] + Esfy*Dinv[1][1] + Esfxy*Dinv[1][2];
        EsfD3 = Esfx*Dinv[2][0] + Esfy*Dinv[2][1] + Esfxy*Dinv[2][2];
        Esf[j] = EsfD1*Esfx + EsfD2*Esfy + EsfD3*Esfxy;
        norma_esf = norma_esf + Esf[j];
    }

    Norma_Energia[i] = norma_esf*prop[i][1]*area;
    error2_ener = error2_ener + Norma_Energia[i];
}

error1_elem = sqrt(error1_elem);
error2_ener = sqrt(error2_ener);
for(i=0;i<NE;i++)
{
    if (elemtype[i]==0 || elemtype[i]==2)
        Param_Local[i] = (sqrt(Norma_Elem[i]))*(sqrt(NE-ncont))/(tolc*error2_ener);
}

printf (" \n");
printf("Error Esfuerzo e: %f\n",error1_elem);
printf("Error Energia U : %f\n",error2_ener);
printf("Tolerancia Error tol: %f\n",tolc);
printf("Error Total : (e/[U.tol]) %f\n",(error1_elem)/(tolc*error2_ener));
printf("----- \n");
printf (" \n");
}
//*****
void PrintTime(char file[20])
{
    ofstream out;
    out.open(file,ios::app);
    out << " \n";
    out << " TIEMPO DE EJECUCION = " << (time_fin - time_ini) << " segundos \n";
    out << " \n";
}
//*****
void PrintError(char file[20])
{
    int i;
    ofstream mistream;
}

```

```
mistream.open(file,ios::in|ios::app);
mistream << "Error Esfuerzo e : " << error1_elem << "\n";
mistream << "Error Energia U : " << error2_ener << "\n";
mistream << "Tolerancia Error tol : " << tolc << "\n";
mistream << "Error Total : " << (error1_elem)/(tolc*error2_ener) << "\n";
mistream << "-----\n";
for(i=0;i<(NE-ncont);i++)
{
    mistream << "Param_Local\t";
    mistream << i+1 << "\t";
    mistream << setw(10) << Param_Local[i] << "\n";
}
```

```

// Método de Elemento Finito, Malla Triangular de 6 Nudos
// Método de Elementos de Contacto , Master-Slave , Nudo a Nudo
#include <time.h>
#include <string.h>
#include <stdio.h>
#include "stdlib.h"
#include <fstream.h>
#include "iostream.h"
#include <math.h>
#include <iomanip.h>
#define NDOFN 2 /* N° Grado de Libertad por Nudo */
#define NDIM 2 /* No. of dimensions */
#define NPROP 3 /* No. de propiedades por elemento */
#define NNE 6 /* No. de nudos por elemento */
#define CLK_TCK 1000.0
typedef char cstring[20];
int NN,NE;
int aBanda;
double **coord=NULL;
int **support=NULL;
double **load=NULL;
int **connect=NULL;
double **prop=NULL;
int *elemtype=NULL;
int ndof,band,ndofe;
int LABEL[NDOFN*NNE];
double ESM[NDOFN*NNE][NDOFN*NNE];
double ERHS[NDOFN*NNE];
double D[3][3], B1[3][2], B2[3][2], DB[3][2], BDB[2][2],Bmat[3][12],BDBmat[12][12];
double **ESF=NULL;
double ***NudoGauss=NULL;
double ***ESF_Elem=NULL;
double **ESFNudo=NULL;
double **A=NULL;
double **A1=NULL;
double **A2=NULL;
double **A3=NULL;
double *X=NULL;
double *BB=NULL;
double *BB2=NULL;
double *BBx=NULL;
double *BBY=NULL;
double *BBxy=NULL;
double **EsfANP=NULL;
double **EsfANMC=NULL;
double BIGg,BIGkr;
double *RHS;
double *fonzi=NULL;
double *P=NULL;
double *disp=NULL;
double tol, tolc;
int nitem, nitemc;
int *pos=NULL;
double *DIAGONAL=NULL;
double ** FILA=NULL;
double ** COLUMNNA=NULL;
double ** KRG=NULL;
double *U=NULL;
int min1(int i,int j);
int max1(int i,int j);
void GetInput(cstring s);
void SetBandSystem();
void GetConnect(int elemno);
void Assemble(int elemno);
    
```

```

void GetStiff(int elemno);
void ApplySupport();
void Solve();
void Soluc();
void PrintTime(char file[20]);
void PrintOut(char file[20]);
void PrintOutDesp(char file[20], int iterc);
void PrintSal(char file[20], int elemno);
void PrintSal1(char file[20], int elemno, int gauss, int iterc);
void PrintSal2(char file[20], int elemno);
void PrintSal3(char file[20], int elemno, int iterc);
void PrintRigidez1(char file[20], int elemno);
void PrintSalM(char file[20]);
void PrintSalM1(char file[20]);
void PrintEsf(char file[20], int iterc);
void solveEsf(int elemno);
void PrintEsfAlizado(char file[20]);
void ClearMemory();
int Kposi(int i, int j);
double &Kpos(int i, int j);
void ClearVariant();
void ludcmp(double **a, int n, int *indx, double *d);
void ludcmp2(double **a2, int n, int *indx2, double *d);
void lubksb(double **a, int n, int *indx, double b[]);
void lubksb1(int n, int *indx);
void lubksb2(int n, int *indx2);
int *indx;
int *indx2;
#define TINY 1.0e-20
int iterc;
int ncont;
char file[20];
char archivo[40];
char archivo1[40];
double time_ini, time_fin;

int main(int argc, char* argv[])
{
    int i;
    cout <<"INGRESE ARCHIVO DE ENTRADA: ";
    cin >> file;
    tolc=0.05; //tolerancia
    iterc=1; // inicia la iteracion con 1
    nitemc=1; //numero de iteraciones (por defecto es 1, pero si hay elemcontac vale 2, luego verifica)
    GetInput(file);
    cout <<"INGRESE NOMBRE DE ARCHIVOS DE SALIDA (s/extension): ";
    cin >> file;
    strcpy( archivo, file);
    strcat( archivo, "_matrix.jg1");
    cout << archivo << "\n";
    strcpy( archivo1, file);
    strcat( archivo1, "_out.jgv");
    cout << archivo1 << "\n";
    time_ini=clock()/CLK_TCK;
    SetBandSystem();
    // [1].- Inicio de las Iteraciones para el Contacto
    for (;)
    {
        ClearVariant();
        for (i=0;i<NE;i++)
            Assemble(i);
        PrintSalM(archivo);
        ApplySupport();
        PrintSalM1(archivo);
        aBanda=0;
    }
}

```

```

for(i=0;i<ndof;i++)
    if (aBanda<pos[i])
        aBanda=pos[i];

Solve();
PrintOutDesp(archivo1,iterc);
double normc=0;
double normdc=0;
for(i=0;i<ndof;i++)
    normdc += (RHS[i]-disp[i])*(RHS[i]-disp[i]);
normdc = sqrt(normdc);
for(i=0;i<ndof;i++)
    disp[i] = RHS[i];
for(i=0;i<ndof;i++)
    normc += (RHS[i]*RHS[i]);
normc = sqrt(normc);
double errc=(normdc/normc);
printf("Error 1 : %f\n",normdc);
printf("Error 2 : %f\n",normc);
printf("Norma Desplaza: %f\n",errc);
if (errc<=tolc)
{
    cout << " OK!!! El error calculo es menor que el requerido \n";
    cout << " el programa finalizara \n";
    break;
}
if (iterc>=nitemc)
{
    cout << " : ( numero.de iteraciones mayor que el maximo definido \n";
    cout << " el programa finalizara \n";
    break;
}
}
iterc=iterc+1;
cout <<"**** Fin de " << iterc-1 << " ° Iteracion \n";
}
// [1].- Fin de las Iteraciones para el Contacto
ncont=0;
for (i=0;i<NE;i++)
    if (elemtype[i]==1)
        ncont=ncont+1;

PrintOut(archivo);
PrintEsf(archivo1,iterc);
PrintEsfAlizado(archivo1);
ClearMemory();
time_fin=clock()/CLK_TCK;
PrintTime(archivo);
cout << "Tiempo total: " << (time_fin-time_ini) << " segundos\n";
return 0;
}
////////////////////////////////////
int min1(int i,int j)
{
    if(i<j) return i;
    return j;
}
////////////////////////////////////
int max1(int i,int j)
{
    if(i>j) return i;
    return j;
}
////////////////////////////////////
void GetInput(cstring s)
{
    ifstream inp;
    int i,j,k;

```

```

inp.open(s,ios::in);
inp >> NN;
printf("N de nudos : %d\n",NN);
inp >> NE;
printf("N de elementos : %d\n",NE);
inp >> iterc;
printf("N de Iteraciones : %d\n",iterc);
coord = new double*[NN];
for (i=0;i<NN;i++)
    coord[i]=new double[NDIM];
support = new int*[NN];
for (i=0;i<NN;i++)
    support[i]=new int[NDOFN];
load = new double*[NN];
for (i=0;i<NN;i++)
    load[i]=new double[NDOFN];
for(i=0;i<NN;i++)
{
    inp >> k;
    k--;
    for(j=0;j<NDIM;j++)
        inp >> coord[k][j];
    for(j=0;j<NDOFN;j++)
        inp >> support[k][j];
    for(j=0;j<NDOFN;j++)
        inp >> load[k][j];
}
connect = new int*[NE];
for (i=0;i<NE;i++)
    connect[i]=new int[NNE];
prop = new double*[NE];
for (i=0;i<NE;i++)
    prop[i]=new double[NPROP];
elemtype = new int [NE];
for(i=0;i<NE;i++)
{
    inp >> k;
    k--;
    for(j=0;j<NNE;j++)
    {
        inp >> connect[k][j];
        connect[k][j]-;
    }
    inp >> elemtype[k];
    for(j=0;j<NPROP;j++)
        inp >> prop[k][j];
}
inp >> BIGg;
inp >> BIGkr;
}
////////////////////////////////////
void SetBandSystem()
{
    int i,j,k;
    ndof = NDOFN*NN;
    ndofe = NDOFN*NNE;
    band = 0;
    pos = new int[ndof]; // gdl x nn
    printf("ndof: %d\n",ndof);
    printf("ndofe: %d\n",ndofe);
    for(i=0;i<ndof;i++)
        pos[i]=0;
    for(i=0;i<NE;i++)
    {
        GetConnect(i);
    }
}

```

```

if (elemtype[i]==0 || elemtype[i]==2)
{
    for(j=0;j<ndofe;j++)
    {
        for(k=0;k<j;k++)
        {
            int l1=LABEL[j];
            int l2=LABEL[k];
            int dif=abs(l2-l1);
            int l3=max1(l1,l2);
            if (pos[l3]<dif)
                pos[l3]=dif;
        }
    }
}
else
{
    nitemc=2; //por defecto es 1 iter, pero como hay elementcontact iterara 2 veces
    for(j=0;j<NDOFN*2;j++)
    {
        for(k=0;k<j;k++)
        {
            int l1=LABEL[j];
            int l2=LABEL[k];
            int dif=abs(l2-l1);
            int l3=max1(l1,l2);
            if (pos[l3]<dif)
                pos[l3]=dif;
        }
    }
}
}
DIAGONAL = new double [ndof];
COLUMNA = new double* [ndof];
FILA = new double* [ndof];
for(i=0;i<ndof;i++)
{
    COLUMNA[i]=NULL;
    FILA[i]=NULL;
    j=pos[i];
    if (j>0)
    {
        COLUMNA[i]= new double[j];
        FILA[i]= new double[j];
    }
}
for(i=0;i<ndof;i++)
{
    DIAGONAL[i]=0.0;
    for(j=0;j<pos[i];j++)
    {
        COLUMNA[i][j]=0.0;
        FILA[i][j]=0.0;
    }
}
disp = new double [ndof];
for(i=0;i<ndof;i++)
    disp[i] = 0.0;
RHS = new double [ndof];
for(i=0;i<ndof;i++)
    RHS[i] = 0.0;
ESF = new double*[NE];
for(i=0;i<NE;i++)
    ESF[i]=new double[3*3];

```

```

NudoGauss = new double**[NE];
for(i=0;i<NE;i++)
{
    NudoGauss[i]=new double*[3];
    for(j=0;j<3;j++)
        NudoGauss[i][j]=new double[2];
}
ESF_Elem= new double**[NE];
for(i=0;i<NE;i++)
{
    ESF_Elem[i]=new double*[6];
    for(j=0;j<6;j++)
        ESF_Elem[i][j]=new double[3];
}
ESFNudo = new double*[NE];
for(i=0;i<NE;i++)
    ESFNudo[i]=new double[3*3];
for(i=0;i<NE;i++)
    for(j=0;j<(3*3);j++)
    {
        ESF[i][j]=0.0;
        ESFNudo[i][j]=0.0;
    }
}
////////////////////////////////////
void GetConnect(int elemno) // LABEL[0] hasta LABEL[5] son los 2x3=6gdl por elemento
{
    int i,j;
    int aux;
    aux=1;
    if (elemtype[elemno]==0 || elemtype[elemno]==2)
    {
        for(i=0;i<NNE;i++)
            for(j=0;j<NDOFN;j++)
                LABEL[i*NDOFN+j] = connect[elemno][i]*NDOFN+(j);
    }
    else
    {
        for(i=0;i<2;i++)
            for(j=0;j<NDOFN;j++)
                LABEL[i*NDOFN+j] = connect[elemno][i]*NDOFN+(j);
    }
}
////////////////////////////////////
void Assemble(int elemno)
{
    int i,j;
    GetConnect(elemno);
    GetStiff(elemno);
    for(i=0;i<ndofe;i++)
    {
        DIAGONAL[LABEL[i]] += ESM[i][i];
        RHS[LABEL[i]] += ERHS[i];
    }
    for(i=1;i<ndofe;i++)
    {
        for(j=0;j<i;j++)
        {
            if (LABEL[j]<=LABEL[i])
                FILA[LABEL[i]][abs(LABEL[j]-LABEL[i])-1] += ESM[i][j];
            if (LABEL[j]>LABEL[i])
                FILA[LABEL[j]][abs(LABEL[j]-LABEL[i])-1] += ESM[i][j];
        }
    }
}

```

```

for(j=1;j<ndofe;j++)
{
    for(i=0;i<j;i++)
    {
        if(LABEL[i]<=LABEL[j])
            COLUMNA[LABEL[j]][abs(LABEL[i]-LABEL[j])-1] += ESM[j][i];
        if(LABEL[i]>LABEL[j])
            COLUMNA[LABEL[i]][abs(LABEL[i]-LABEL[j])-1] += ESM[j][i];
    }
}
}
////////////////////////////////////////////////////
void GetStiff(int elemno)
{
    int i,ii,iii,j,k,l,II,III,kode;
    // variables para FEM
    double coef1;
    double xx1,yy1,xx2,yy2;
    double xx1f,yy1f,xx2f,yy2f;
    double vectnx,vectny,l0,lf,kresorte;
    double L1[3], L2[3], peso[3];
    double dN[2][6], MInvJ[2][2];
    double sumN1x, sumN1y, sumN2x, sumN2y;
    double deterJ;
    kode=elemtype[elemno];
    if(kode==0 || kode==2)
    {
        // prop[elemno][0] = E
        // prop[elemno][1] = Espesor
        // prop[elemno][2] = Poisson
        // prop[elemno][0] = nx
        // prop[elemno][1] = ny
        // prop[elemno][2] = Kresorte
        // Inicio Matrix del Elemento Finito
        // Matrix "D"
        if(kode==0) //Para el caso de TENSION PLANA
        {
            coef1=prop[elemno][0]/(1-(prop[elemno][2]*prop[elemno][2]));
            D[0][0]=coef1*1;
            D[0][1]=coef1*prop[elemno][2];
            D[0][2]=0;
            D[1][0]=coef1*prop[elemno][2];
            D[1][1]=coef1*1;
            D[1][2]=0;
            D[2][0]=0;
            D[2][1]=0;
            D[2][2]=coef1*(1-prop[elemno][2])/2;
        }
        if(kode==2) //Para el caso de DEFORMACION PLANA
        {
            coef1=prop[elemno][0]*(1-prop[elemno][2])/(1+prop[elemno][2])/(1-2*prop[elemno][2]);
            D[0][0]=coef1*1;
            D[0][1]=coef1*prop[elemno][2]/(1-prop[elemno][2]);
            D[0][2]=0;
            D[1][0]=coef1*prop[elemno][2]/(1-prop[elemno][2]);
            D[1][1]=coef1*1;
            D[1][2]=0;
            D[2][0]=0;
            D[2][1]=0;
            D[2][2]=prop[elemno][0]/2/(1+prop[elemno][2]);
        }
        for (ii=0;ii<ndofe;ii++)
            for (iii=0;iii<ndofe;iii++)
                ESM[ii][iii]=0.0;
        L1[0]=0.5;
    }
}

```

```

L2[0]=0.5;
peso[0]=0.1666666666666666666666666666667;
L1[1]=0;
L2[1]=0.5;
peso[1]=0.1666666666666666666666666666667;
L1[2]=0.5;
L2[2]=0;
peso[2]=0.5-peso[0]-peso[1];
for (ii=0;ii<3;ii++)
{
    dN[0][0]=4*L1[ii]-1;
    dN[0][1]=0;
    dN[0][2]=4*L1[ii]+4*L2[ii]-3;
    dN[0][3]=4*L2[ii];
    dN[0][4]=-4*L2[ii];
    dN[0][5]=4-8*L1[ii]-4*L2[ii];
    dN[1][0]=0;
    dN[1][1]=4*L2[ii]-1;
    dN[1][2]=4*L1[ii]+4*L2[ii]-3;
    dN[1][3]=4*L1[ii];
    dN[1][4]=4-4*L1[ii]-8*L2[ii];
    dN[1][5]=-4*L1[ii];
    sumN1x=0;
    sumN1y=0;
    sumN2x=0;
    sumN2y=0;
    for(i=0;i<6;i++)
    {
        sumN1x = sumN1x + dN[0][i] * coord[connect[elemno][i]][0];
        sumN1y = sumN1y + dN[0][i] * coord[connect[elemno][i]][1];
        sumN2x = sumN2x + dN[1][i] * coord[connect[elemno][i]][0];
        sumN2y = sumN2y + dN[1][i] * coord[connect[elemno][i]][1];
    }
    deterJ = (sumN1x*sumN2y) - (sumN1y*sumN2x);
    if (deterJ<0)
        deterJ=(deterJ)*(-1);
    MInvJ[0][0] = sumN2y/deterJ;
    MInvJ[0][1] = -sumN1y/deterJ;
    MInvJ[1][0] = -sumN2x/deterJ;
    MInvJ[1][1] = sumN1x/deterJ;
    for (j=0;j<6;j++)
    {
        Bi[0][0] = (MInvJ[0][0]*dN[0][j]) + (MInvJ[0][1]*dN[1][j]);
        Bi[0][1] = 0;
        Bi[1][0] = 0;
        Bi[1][1] = (MInvJ[1][0]*dN[0][j]) + (MInvJ[1][1]*dN[1][j]);
        Bi[2][0] = Bi[1][1];
        Bi[2][1] = Bi[0][0];
        //Bmat = matrix de impresion.
        Bmat[0][j*2] = Bi[0][0];
        Bmat[0][j*2+1] = Bi[0][1];
        Bmat[1][j*2] = Bi[1][0];
        Bmat[1][j*2+1] = Bi[1][1];
        Bmat[2][j*2] = Bi[2][0];
        Bmat[2][j*2+1] = Bi[2][1];
        for (k=0;k<6;k++)
        {
            Bj[0][0] = (MInvJ[0][0]*dN[0][k]) + (MInvJ[0][1]*dN[1][k]);
            Bj[0][1] = 0;
            Bj[1][0] = 0;
            Bj[1][1] = (MInvJ[1][0]*dN[0][k]) + (MInvJ[1][1]*dN[1][k]);
            Bj[2][0] = Bj[1][1];
            Bj[2][1] = Bj[0][0];
            for (ll=0;ll<3;ll++)
                for (ll=0;ll<2;ll++)
    }

```

```

                                DB[i][i] = 0.0;
                                for(l=0;l<3;l++)
                                    for(ll=0;ll<2;ll++)
                                        for(III=0;III<3;III++)
                                            DB[i][i] = DB[i][i] + D[i][III]*Bj[III][i];
                                for(l=0;l<2;l++)
                                    for(ll=0;ll<2;ll++)
                                        BDB[i][i] = 0.0;
                                for(l=0;l<2;l++)
                                    for(ll=0;ll<2;ll++)
                                        for(III=0;III<3;III++)
                                            BDB[i][i] = BDB[i][i] + Bi[III][i]*DB[III][i];
                                BDBmat[j*2][k*2]=BDB[0][0];
                                BDBmat[j*2+1][k*2]=BDB[1][0];
                                BDBmat[j*2][k*2+1]=BDB[0][1];
                                BDBmat[j*2+1][k*2+1]=BDB[1][1];
                                ESM[j*2][k*2] += BDB[0][0]*prop[elemno][1]*deterJ*peso[i];
                                ESM[j*2+1][k*2+1] += BDB[0][1]*prop[elemno][1]*deterJ*peso[i];
                                ESM[j*2][k*2+1] += BDB[1][0]*prop[elemno][1]*deterJ*peso[i];
                                ESM[j*2+1][k*2+1] += BDB[1][1]*prop[elemno][1]*deterJ*peso[i];
                            }
                        }
                    PrintSal1(archivo, elemno, ii, iterc);
                }
            // Fin Matrix del Elemento Finito
            for (i=0;i<(NDOFN*NNE);i++)
                ERHS[i]=0.0;
        }
    else
        { // para la matrix de rigidez del elemento
            xx1=coord[connect[elemno][0]][0];
            yy1=coord[connect[elemno][0]][1];
            xx2=coord[connect[elemno][1]][0];
            yy2=coord[connect[elemno][1]][1];
            l0=sqrt((xx2-xx1)*(xx2-xx1)+(yy2-yy1)*(yy2-yy1));
            vectnx=(xx2-xx1)/l0;
            vectny=(yy2-yy1)/l0;
            if ((xx2-xx1)==0)
            {
                if ((yy2-yy1)==0)
                {
                    vectnx=prop[elemno][0]/sqrt(prop[elemno][0]*prop[elemno][0]+prop[elemno][1]*prop[elemno][1]);
                    vectny=prop[elemno][1]/sqrt(prop[elemno][0]*prop[elemno][0]+prop[elemno][1]*prop[elemno][1]);
                }
            }
            xx1f=disp[connect[elemno][0]*2]+xx1;
            yy1f=disp[connect[elemno][0]*2+1]+yy1;
            xx2f=disp[connect[elemno][1]*2]+xx2;
            yy2f=disp[connect[elemno][1]*2+1]+yy2;
            lf=sqrt((xx2f-xx1f)*(xx2f-xx1f)+(yy2f-yy1f)*(yy2f-yy1f));
            if (iterc == 1)
                kresorte=0;
            else
            { //prop[elemno][2]=prop[elemno][2]*2;
                kresorte=prop[elemno][2];
                if (l0>=0)
                {
                    if (l0<=lf)
                        //kresorte=BIGkr;
                    if (0<=lf && lf<l0)
                        kresorte=0;
                    if (lf<0)
                        kresorte=0;
                }
            }
            if (l0<0)

```

```

        {
            if (lf<=l0)
                //kresorte=BIgGr;
            if (l0<=lf && lf<=0)
                kresorte=0;
            if (0<lf)
                kresorte=0;
        }
    }
    for (ii=0;ii<ndofe;ii++)
        for (iii=0;iii<ndofe;iii++)
            ESM[ii][iii]=0.0;
    ESM[0][0]=kresorte*(-vectnx)*(-vectnx);
    ESM[0][1]=kresorte*(-vectnx)*(-vectny);
    ESM[0][2]=-ESM[0][0];
    ESM[0][3]=-ESM[0][1];
    ESM[1][0]=ESM[0][1];
    ESM[1][1]=kresorte*(-vectny)*(-vectny);
    ESM[1][2]=-ESM[1][0];
    ESM[1][3]=-ESM[1][1];
    ESM[2][0]=ESM[0][2];
    ESM[2][1]=ESM[1][2];
    ESM[2][2]=-ESM[2][0];
    ESM[2][3]=-ESM[2][1];
    ESM[3][0]=ESM[0][3];
    ESM[3][1]=ESM[1][3];
    ESM[3][2]=-ESM[0][3];
    ESM[3][3]=-ESM[1][3];
    PrintSal3(archivo, elemno, iterc);
    ERHS[0]=kresorte*(-vectnx)*l0;
    ERHS[1]=kresorte*(-vectny)*l0;
    ERHS[2]=-ERHS[0];
    ERHS[3]=-ERHS[1];
    ERHS[4]=0;
    ERHS[5]=0;
}
}
////////////////////////////////////
void ApplySupport()
{
    int i,j,pos;
    for(i=0;i<NN;i++)
        for(j=0;j<NDOFN;j++)
        {
            pos = i*NDOFN+j;
            if(support[i][j]==1)
            {
                DIAGONAL[pos] +=BIGg;
                RHS[pos] += BIGg*load[i][j];
            }
            else
            {
                RHS[pos] += load[i][j];
            }
        }
}
////////////////////////////////////
void Solve()
{
    int i,j,k,kk;
    double pivot;
    double suma;
    for(i=1;i<ndof;j++)
    {
        if(pos[i] != 0)

```

```

    {
        for(j=pos[i];j>0;j--)
        {
            if(FILA[i][j-1] != 0)
            {
                pivot=(FILA[i][j-1])/DIAGONAL[i-j];
                FILA[i][j-1] = 0;
                for (k=1;k<j;k++)
                    if(pos[(i-j)+k] >= k)
                        FILA[i][j-1-k] += -COLUMNA[i-j+k][k-1]*pivot;
                if(COLUMNA[i][j-1] != 0)
                    DIAGONAL[i] += -COLUMNA[i][j-1]*pivot;
                kk=(i-j)+aBanda;
                if(kk>ndof)
                    kk=ndof-1;
                kk=kk-j;
                for(k=0;k<kk;k++)
                {
                    if ((k+1) < pos[i+(k+1)])
                        if(j<pos[i+k+1]-k)
                            COLUMNA[i+k+1][k] += -COLUMNA[i+k+1][j+k]*pivot;
                }
                RHS[i] += -RHS[i-j]*pivot;
            }
        }
    }
}
for(i=ndof-1;i>=0;i--)
{
    double Kij;
    suma=0.0;
    for(j=i+1;j<ndof;j++)
        if(Kpos(i,j))
        {
            Kij=Kpos(i,j);
            suma += Kij*RHS[j];
        }
    RHS[i] = (RHS[i]-suma)/Kpos(i,i);
}
}
////////////////////////////////////////////////////////////////////
void PrintOut(char file[20])
{
    int i,j;
    ofstream out;
    out.open(file,ios::app);
    out << "**** RESULTADOS FINALES DE MATRICES AUXILIARES ****" << "\n";
    out << "MATRIZ DIAGONAL" << "\n";
    for(i=0;i<ndof;i++)
        out << DIAGONAL[i] << "\t";
    out << "\n";
    out << "\n";
    out << "MATRIZ FILA" << "\n";
    for(i=1;i<ndof;i++)
    {
        out << "posicion: " << i << " , Tiene [" << pos[i] << "]" << "\n";
        for(j=0;j<pos[i];j++)
            out << FILA[i][j] << "\t";
        out << "\n";
    }
    out << "\n";
    out << "MATRIZ COLUMNA" << "\n";
    for(i=1;i<ndof;i++)
    {
        out << "posicion: " << i << " , Tiene [" << pos[i] << "]" << "\n";
    }
}

```

```

        for(j=0;j<pos[i];j++)
            out << COLUMNA[i][j] << "\t";
        out << "\n";
    }
    out << "\n";
}
void PrintTime(char file[20])
{
    ofstream out;
    out.open(file,ios::app);
    out << "\n";
    out << " TIEMPO DE EJECUCION = " << (time_fin - time_ini) << " segundos \n";
    out << "\n";
}
////////////////////////////////////
void PrintOutDesp(char file[20], int iterc)
{
    int i,j;
    ofstream mistream;
    mistream.open(file,ios::in|ios::app);
    mistream << "\n";
    mistream << "ELEMENTO TRIANGULAR DE 6 PUNTOS - PENALIDAD" << "\n";
    mistream << "\n";
    mistream << "DESPLAZAMIENTO : Iteracion [" << iterc << "]" \n";
    mistream << "NUDO" << "\t";
    mistream << setw(14) << "UX" << "\t";
    mistream << setw(14) << "UY" << "\n";
    for (i=0;i<NN;i++)
    {
        mistream << i+1 << "\t";
        for (j=0;j<NDOFN;j++)
            mistream << setw(15) << RHS[i*NDOFN+j] << "\t";
        mistream << "\n";
    }
    for (i=0;i<NE;i++)
    {
        if (elemtype[i]==1)
        {
            mistream << setw(15) << i << "\t";
            mistream << setw(15) << prop[i][2] << "\n";
        }
    }
    mistream << "\n";
}
////////////////////////////////////
void ClearMemory()
{
    int i;
    for (i=0;i<NN;i++)
        delete [] coord[i];
    delete [] coord;
    for (i=0;i<NN;i++)
        delete [] support[i];
    delete [] support;
    for (i=0;i<NN;i++)
        delete [] load[i];
    delete [] load;
    for (i=0;i<NE;i++)
        delete [] connect[i];
    delete [] connect;
    for (i=0;i<NE;i++)
        delete [] prop[i];
    delete [] prop;
    delete [] elemtype;
    delete [] DIAGONAL;
}

```

```

for(i=0;i<ndof;i++)
{
    if (COLUMNA[i]!=NULL)
        delete COLUMNA[i];
    if (FILA[i]!=NULL)
        delete FILA[i];
}
delete [] COLUMNA;
delete [] FILA;
for (i=0;i<NE;i++)
    delete [] ESF[i];
delete [] ESF;
}
////////////////////////////////////
double &Kpos(int i, int j)
{
    if (i==j)
        return DIAGONAL[i];
    else if (j>i)
        return COLUMNA[j][i-j-1];
    else
        return FILA[i][i-j-1];
}
int Kposi(int i, int j)
{
    if (i==j)
        return 1;
    else if (j>i)
    {
        if ((j-i)>pos[j])
            return 0;
        else
            return 1;
    }
    else
    {
        if ((i-j)>pos[i])
            return 0;
        else
            return 1;
    }
}
////////////////////////////////////
void ClearVariant()
{
    int i,j;
    for(i=0;i<ndof;i++)
    {
        DIAGONAL[i]=0.0;
        for(j=0;j<pos[i];j++)
        {
            COLUMNA[i][j]=0.;
            FILA[i][j]=0.;
        }
    }
    for(i=0;i<ndof;i++)
        RHS[i] = 0.0;
}
////////////////////////////////////
void PrintSal(char file[20], int elemno)
{
    int i,j,kode;
    ofstream mistream;
    mistream.open(file,ios::in|ios::app);
    mistream << " \n";
}

```

```

kode=elemtype[elemno];
if(kode==0 || kode==2)
{
    mistream << "MATRIZ DB : [" << elemno+1 << "]" \n";
    for (i=0;i<3;i++)
    {
        for (j=0;j<2;j++)
            mistream << setw(12) << DB[i][j] << "\t";
        mistream << "\n";
    }
    mistream << "\n";
    mistream << "MATRIZ BDB : [" << elemno+1 << "]" \n";
    for (i=0;i<2;i++)
    {
        for (j=0;j<2;j++)
            mistream << setw(12) << BDB[i][j] << "\t";
        mistream << "\n";
    }
    mistream << "\n";
    mistream << "MATRIZ RIGIDEZ : [" << elemno+1 << "]" \n";
    for (i=0;i<ndofe;i++)
    {
        for (j=0;j<ndofe;j++)
            mistream << setw(12) << ESM[i][j] << "\t";
        mistream << "\n";
    }
    mistream << "\n";
    mistream << "\n";
}
else
{
    mistream << "MATRIZ RIGIDEZ RESORTE: [" << elemno+1 << "]" \n";
    for (i=0;i<4;i++)
    {
        for (j=0;j<4;j++)
            mistream << setw(12) << ESM[i][j] << "\t";
        mistream << "\t";
        mistream << setw(12) << ERHS[i] << "\n";
    }
    mistream << "\n";
    mistream << "\n";
}
}
////////////////////////////////////
void PrintRigidez1(char file[20], int elemno)
{
    int i,j;
    fstream mistream;
    mistream.open(file,ios::in|ios::app);
    mistream << "\n";
    mistream << "**** Bi Bj : Elemento : [" << elemno+1 << "]" \n";
    for(i=0;i<3;i++)
    {
        for(j=0;j<2;j++)
            mistream << setw(12) << Bi[i][j] << "\t";
        mistream << "\t";
        for(j=0;j<2;j++)
            mistream << setw(12) << Bj[i][j] << "\t";
        mistream << "\n";
    }
    mistream << "\n";
    for(i=0;i<3;i++)
    {
        for(j=0;j<2;j++)
            mistream << setw(12) << DB[i][j] << "\t";

```

```

        mistream << "\n";
    }
    mistream << "\n";
    for(i=0;i<2;i++)
    {
        for(j=0;j<2;j++)
            mistream << setw(12) << BDB[i][j] << "\t";
        mistream << "\n";
    }
    mistream << "\n";
}
/////////////////////////////////////////////////////////////////
void PrintSal1(char file[20], int elemno, int gauss, int iterc) // Imprime Matrices: Bi[3x2], Bi.D.Bj y K
{
    int i,j;
    ofstream mistream;
    mistream.open(file,ios::in|ios::app);
    if( iterc == 1) //imprime solo cuando realiza la 1º iteración; no es necesario mostrar mas veces, solo la rigidez.
    {
        mistream << "\n";
        mistream << "***Elemento: [" << elemno+1 << "]" - Pto de Gauss [" << gauss+1 << "]"**\n";
        mistream << " Matriz: Bi [3x2]: [i=1..6] \n";
        for(i=0;i<3;i++)
        {
            for(j=0;j<ndofe;j++)
                mistream << setw(10) << Bmat[i][j] << "\t";
            mistream << "\n";
        }
        mistream << "\n";
        mistream << " Matriz: Bi.D.Bj [2x2]: [i=1..6][j=1..6]\n";
        for (i=0;i<ndofe;i++)
        {
            for (j=0;j<ndofe;j++)
                mistream << setw(14) << BDBmat[i][j] << "\t";
            mistream << "\n";
        }
        mistream << "\n";
        mistream << " Matriz Rigidez Local Acumulado [" << gauss+1 << "ptos Gauss]:
K=Bi.D.Bj.t.Peso.DetJ [12x12]: [i=1..6][j=1..6] \n";
        for (i=0;i<ndofe;i++)
        {
            for (j=0;j<ndofe;j++)
                mistream << setw(14) << ESM[i][j] << "\t";
            mistream << "\n";
        }
        mistream << "\n";
        mistream << "\n";
    }
}
/////////////////////////////////////////////////////////////////
void PrintSal2(char file[20], int elemno)
{
    ofstream mistream;
    mistream.open(file,ios::in|ios::app);
    mistream << " \n";
}
/////////////////////////////////////////////////////////////////
void PrintSal3(char file[20], int elemno, int iterc)
{
    int i,j;
    ofstream mistream;
    mistream.open(file,ios::in|ios::app);
    mistream << " \n";
    mistream << " \n";
    mistream << "Elemento: [" << elemno+1 << "]" - Iteración [" << iterc << "]" \n";
}

```

```

mistream << "Matriz Rigidez Local del Resorte: \n";

mistream << "\n";
for (i=0;i<ndofe;i++)
{
    for (j=0;j<ndofe;j++)
        mistream << setw(12) << ESM[i][j] << "\t";
    mistream << "\n";
}
mistream << "\n";
}
////////////////////////////////////
void PrintSalM(char file[20])
{
    int i,j;
    ofstream out;
    out.open(file,ios::app);
    out << "MATRIZ FILA : " << " \n";
    for (i=0;i<ndof;i++)
    {
        out << pos[i] << "\t";
        for (j=0;j<pos[i];j++)
            out << FILA[i][j] << "\t";
        out << "\n";
    }
    out << "\n";
    out << "\n";
    out << "MATRIZ COLUMNA : " << " \n";
    for (i=0;i<ndof;i++)
    {
        out << pos[i] << "\t";
        for (j=0;j<pos[i];j++)
            out << COLUMNA[i][j] << "\t";
        out << "\n";
    }
    out << "\n";
    out << "\n";
    out << "MATRIZ DIAGONAL Inicial: " << " \n";
    for (i=0;i<ndof;i++)
        out << DIAGONAL[i] << "\t";
    out << "\n";
    out << "\n";
    out << "MATRIZ FUERZA Inicial: " << " \n";
    for (i=0;i<ndof;i++)
        out << RHS[i] << "\t";
    out << "\n";
    out << "\n";
}
////////////////////////////////////
void PrintSalM1(char file[20])
{
    int i;
    ofstream out;
    out.open(file,ios::app);
    out << "MATRIZ DIAGONAL Final: " << " \n";
    for (i=0;i<ndof;i++)
        out << DIAGONAL[i] << "\t";
    out << "\n";
    out << "\n";
    out << "MATRIZ FUERZA Final: " << " \n";
    for (i=0;i<ndof;i++)
        out << RHS[i] << "\t";
    out << "\n";
    out << "\n";
}

```

```

}
////////////////////////////////////////////////////////////////
void PrintEsf(char file[20], int iterc)
{
    int i,j,k;
    indx2=new int[3];
    fstream mistream;
    mistream.open(file,ios::in|ios::app);
    cout <<"***** Inicia calculo de Esfuerzos \n";
    mistream <<" \n";
    mistream <<" \n";
    for (i=0;i<NE;i++)
    {
        if (elemtype[i]==0 || elemtype[i]==2)
        {
            GetConnect(i);
            solveEsf(i);
        }
    }
    mistream <<"**** TABLA DE ESFUERZO EN CADA PUNTO DE GAUSS (3 Puntos por cada Elemento) ****\n";
    mistream <<" \n";
    mistream <<"Punto de Gauss   Coord X   Coord Y   Sxx   Syy   Sxy\n";
    mistream <<"-----\n";
    for(i=0;i<NE;i++)
    {
        for(j=0;j<3;j++)
        {
            mistream <<"P.Gauss[" << (i*3+j)+1 << "]" <<"\t";
            mistream << setw(10) << NudoGauss[i][j][0] <<"\t";
            mistream << setw(10) << NudoGauss[i][j][1] <<"\t";
            mistream << setw(15) << ESF[i][j]*3+0 <<"\t";
            mistream << setw(15) << ESF[i][j]*3+1 <<"\t";
            mistream << setw(15) << ESF[i][j]*3+2 <<"\t";
            mistream <<"\n";
        }
    }
    mistream <<" \n";
    mistream <<" \n";
    mistream <<"**** TABLA DE ESFUERZO EN CADA ELEMENTO ****\n";
    mistream <<" \n";
    mistream <<"Elemento   Sxx[1]   Syy[1]   Sxy[1]   Sxx[2]   Syy[2]   Sxy[2]";
    Sxx[3]   Syy[3]   Sxy[3]\n";
    mistream <<"-----\n";
    for(i=0;i<NE;i++)
    {
        mistream <<"E" << i+1 <<" [" << connect[i][0]+1 << ", " << connect[i][1]+1 << ", " << connect[i][2]+1 << "]:\t";
        for(j=0;j<3;j++)
            for(k=0;k<3;k++)
                mistream << setw(10) << ESF_Elem[i][j][k] <<"\t";
        mistream <<"\n";
        mistream <<"E" << i+1 <<" [" << connect[i][3]+1 << ", " << connect[i][4]+1 << ", " << connect[i][5]+1 << "]:\t";
        for(j=3;j<6;j++)
            for(k=0;k<3;k++)
                mistream << setw(10) << ESF_Elem[i][j][k] <<"\t";
        mistream <<"\n";
    }
    cout <<"***** Fin de calculo de Esfuerzos \n";
    mistream <<" \n";
    mistream <<" \n";
}
////////////////////////////////////////////////////////////////
void solveEsf(int elemno)
{
    int i,ii,k,l,ll,lll,kode ;

```

```

// variables para FEM
double coef1;
double L1[3], L2[3], peso[3], coord_gauss1[3], coord_gauss2[3];
double p_gauss, s, t, spto[6], tpto[6];
double dN[2][6], MlnvJ[2][2];
double sumN1x, sumN1y, sumN2x, sumN2y;
double deterJ;
kode=elemtype[elemno];
if(kode==0 || kode==2)
{
// prop[elemno][0] = E
// prop[elemno][1] = Espesor
// prop[elemno][2] = Poisson
// prop[elemno][0] = nx
// prop[elemno][1] = ny
// prop[elemno][2] = 0
// Inicio Matrix del Elemento Finito
// Matrix "D"
coef1=prop[elemno][0]/(1-(prop[elemno][2]*prop[elemno][2]));
D[0][0]=coef1*1;
D[0][1]=coef1*prop[elemno][2];
D[0][2]=0;
D[1][0]=coef1*prop[elemno][2];
D[1][1]=coef1*1;
D[1][2]=0;
D[2][0]=0;
D[2][1]=0;
D[2][2]=coef1*(1-prop[elemno][2])/2;
for (ii=0;ii<(3*3);ii++)
    ESF[elemno][ii]=0.0;
p_gauss = 0.16666666666666667; // 1/6
coord_gauss1[0] = 1*p_gauss; // s
coord_gauss2[0] = 1*p_gauss; // t
peso[0]=0.16666666666666667;
coord_gauss1[1] = 4*p_gauss; //s
coord_gauss2[1] = 1*p_gauss; //t
peso[1]=0.16666666666666667;
coord_gauss1[2] = 1*p_gauss; //s
coord_gauss2[2] = 4*p_gauss; //t
peso[2]=0.5-peso[0]-peso[1];
// L1=1-s-t L2=s L3=t
L1[0]=1-coord_gauss1[0]-coord_gauss2[0];
L2[0]=coord_gauss1[0];
L1[1]=1-coord_gauss1[1]-coord_gauss2[1];
L2[1]=coord_gauss1[1];
L1[2]=1-coord_gauss1[2]-coord_gauss2[2];
L2[2]=coord_gauss1[2];
for (ii=0;ii<3;ii++)
{
    dN[0][0]=4*L1[ii]-1;
    dN[0][1]=0;
    dN[0][2]=4*L1[ii]+4*L2[ii]-3;
    dN[0][3]=4*L2[ii];
    dN[0][4]=-4*L2[ii];
    dN[0][5]=4-8*L1[ii]-4*L2[ii];
    dN[1][0]=0;
    dN[1][1]=4*L2[ii]-1;
    dN[1][2]=4*L1[ii]+4*L2[ii]-3;
    dN[1][3]=4*L1[ii];
    dN[1][4]=4-4*L1[ii]-8*L2[ii];
    dN[1][5]=-4*L1[ii];
    sumN1x=0;
    sumN1y=0;
    sumN2x=0;
    sumN2y=0;
}

```

```

for(i=0;i<6;i++)
{
    sumN1x = sumN1x + dN[0][i] * coord[connect[elemno][i]][0];
    sumN1y = sumN1y + dN[0][i] * coord[connect[elemno][i]][1];
    sumN2x = sumN2x + dN[1][i] * coord[connect[elemno][i]][0];
    sumN2y = sumN2y + dN[1][i] * coord[connect[elemno][i]][1];
}
deterJ = (sumN1x*sumN2y) - (sumN1y*sumN2x);
if (deterJ<0)
    deterJ=(deterJ)*(-1);
MInvJ[0][0] = sumN2y/deterJ;
MInvJ[0][1] = -sumN1y/deterJ;
MInvJ[1][0] = -sumN2x/deterJ;
MInvJ[1][1] = sumN1x/deterJ;
for (k=0;k<6;k++)
{
    Bj[0][0] = (MInvJ[0][0]*dN[0][k]) + (MInvJ[0][1]*dN[1][k]);
    Bj[0][1] = 0;
    Bj[1][0] = 0;
    Bj[1][1] = (MInvJ[1][0]*dN[0][k]) + (MInvJ[1][1]*dN[1][k]);
    Bj[2][0] = Bj[1][1];
    Bj[2][1] = Bj[0][0];
    for(l=0;l<3;l++)
        for(ll=0;ll<2;ll++)
            DB[l][ll] = 0.0;
    for(l=0;l<3;l++)
        for(ll=0;ll<2;ll++)
            for(ill=0;ill<3;ill++)
                DB[l][ill] = DB[l][ll] + D[l][ill]*Bj[ill][l];
}
ESF[elemno][ii*3+0]=ESF[elemno][ii*3+0]+(DB[0][0]*RHS[LABEL[k*2]]+DB[0][1]*RHS[LABEL[k*2+1]]);
//Sxx

ESF[elemno][ii*3+1]=ESF[elemno][ii*3+1]+(DB[1][0]*RHS[LABEL[k*2]]+DB[1][1]*RHS[LABEL[k*2+1]]);
//Syy

ESF[elemno][ii*3+2]=ESF[elemno][ii*3+2]+(DB[2][0]*RHS[LABEL[k*2]]+DB[2][1]*RHS[LABEL[k*2+1]]);
//Sxy
}

NudoGauss[elemno][ii][0]=(L1[ii]*coord[connect[elemno][0]][0]+(L2[ii]*coord[connect[elemno][1]][0]+(1-L1[ii]-L2[ii])*coord[connect[elemno][2]][0]);

NudoGauss[elemno][ii][1]=(L1[ii]*coord[connect[elemno][0]][1]+(L2[ii]*coord[connect[elemno][1]][1]+(1-L1[ii]-L2[ii])*coord[connect[elemno][2]][1]);
}
//calculo de los esfuerzos en los 6 puntos del elemento triang
spto[0]=0;
tpto[0]=0;
spto[1]=1;
tpto[1]=0;
spto[2]=0;
tpto[2]=1;
spto[3]=0.5;
tpto[3]=0;
spto[4]=0.5;
tpto[4]=0.5;
spto[5]=0;
tpto[5]=0.5;
for (i=0;i<6;i++)
{
    s = (spto[i]-p_gauss)/(3*p_gauss);
    t = (tpto[i]-p_gauss)/(3*p_gauss);
    ESF_Elem[elemno][i][0] = ESF[elemno][0*3+0]*(1-s-t)+ESF[elemno][1*3+0]*s+ESF[elemno][2*3+0]*t);
    ESF_Elem[elemno][i][1] = ESF[elemno][0*3+1]*(1-s-t)+ESF[elemno][1*3+1]*s+ESF[elemno][2*3+1]*t);
}

```

```

        ESF_Elem[elemno][i][2] = ESF[elemno][0*3+2]*(1-s-t)+ESF[elemno][1*3+2]*(s)+ESF[elemno][2*3+2]*(t);
    }
}
////////////////////////////////////
void PrintEsfAlizado(char file[20])
{
    int i,j,k,l;
    int conta;
    int jj,kk;
    int NEN[10][2]; //numero de elementos en un nudo, 10 es el valor estimado y 2 es valor de elem y pos
    double x1,x2,x3,y1,y2,y3,areas,detJ;
    double M1,M2,M3,M4,M5,M6;
    int jota;
    EsfANP = new double*[NN];
    for (i=0;i<NN;i++)
        EsfANP[i]=new double[3];
    EsfANMC = new double*[NN];
    for (i=0;i<NN;i++)
        EsfANMC[i]=new double[3];
    A = new double*[NN+ncont+1];
    for (i=0;i<=NN+ncont;i++)
        A[i]=new double[NN+ncont+1];
    A1 = new double*[NN+ncont];
    for (i=0;i<NN+ncont;i++)
        A1[i]=new double[NN+ncont];
    BB=new double[NN+ncont+1];
    BBx=new double[NN+ncont];
    BBy=new double[NN+ncont];
    BBxy=new double[NN+ncont];
    X=new double[NN+ncont];
    for (i=0;i<NN+ncont;i++)
    {
        BB[i+1]=0;
        BBx[i]=0;
        BBy[i]=0;
        BBxy[i]=0;
        X[i]=0;
        for (j=0;j<NN+ncont;j++)
        {
            A[i+1][j+1]=0;
            A1[i][j]=0;
        }
    }
    ofstream mistream;
    mistream.open(file,ios::in|ios::app);
    cout <<"***** Inicia calculo de Esfuerzos Alizados : Metodo Promedio \n";
    for (i=0;i<NN;i++)
    {
        conta=0;
        EsfANP[i][0]=0; //para Sxx
        EsfANP[i][1]=0; //para Syy
        EsfANP[i][2]=0; //para Sxy
        for (j=0;j<10;j++)
        {
            NEN[j][0]=0;
            NEN[j][1]=0;
        }
        for (j=0;j<NE;j++)
        {
            if (elemtype[j]==0 || elemtype[j]==2)
            {
                for (k=0;k<NNE;k++)
                {
                    if (connect[j][k]==i)

```

```

        {
            NEN[conta][0]=j; // numero de elemento
            NEN[conta][1]=k; // la posicion de dicho nudos del elemento j
            conta=conta+1;
        }
    }
}
for (ll=0;ll<conta;ll++)
{
    EsfANP[ll][0] = EsfANP[ll][0] + ESF_Elem[NEN[ll][0]][NEN[ll][1]][0];
    EsfANP[ll][1] = EsfANP[ll][1] + ESF_Elem[NEN[ll][0]][NEN[ll][1]][1];
    EsfANP[ll][2] = EsfANP[ll][2] + ESF_Elem[NEN[ll][0]][NEN[ll][1]][2];
}
EsfANP[ll][0]=EsfANP[ll][0]/conta;
EsfANP[ll][1]=EsfANP[ll][1]/conta;
EsfANP[ll][2]=EsfANP[ll][2]/conta;
}
cout << "**** Fin de calculo de Esfuerzos Alizados : Metodo Promedio \n";
mistream << "\n";
mistream << "\n";
mistream << "**** ESFUERZOS ALIZADOS EN LOS NUDO : Metodo Promedio Aritmetico **** << "\n";
mistream << "\n";
mistream << "\t \t \t Sxx\t \t Syy\t \t Sxy\n";
for (i=0;i<NN;i++)
{
    mistream << "Nudo " << i+1 << " :\t";
    mistream << setw(15) << EsfANP[i][0] << "\t";
    mistream << setw(15) << EsfANP[i][1] << "\t";
    mistream << setw(15) << EsfANP[i][2] << "\t";
    mistream << "\n";
}
}
///////////////////////////////////////////////////////////////////
void Soluc()
{
    double d;
    int nt;
    nt=NN+ncont;
    indx=new int[nt];
    ludcmp(A,nt,indx,&d);
    lubksb1(nt,indx);
}
//*****
void ludcmp(double **a, int n, int *indx, double *d)
{
    int i,imax,j,k;
    float big,dum,sum,temp;
    float *vv;
    vv=new float[n+1];
    for (i=1;i<=n;i++)
        vv[i]=1;
    *d=1.0;
    for (i=1;i<=n;i++) {
        big=0.0;
        for (j=1;j<=n;j++)
            if ((temp=fabs(a[i][j])) > big)
                big=temp;
        if (big == 0.0)
            cout<<"Singular matrix in routine ludcmp \n";
        vv[i]=1.0/big;
    }
    for (j=1;j<=n;j++)
    {
        for (i=1;i<j;i++) {

```

```

        sum=a[i][j];
        for (k=1;k<i;k++) sum -= a[i][k]*a[k][j];
        a[i][j]=sum;
    }
    big=0.0;
    for (i=j;i<=n;i++)
    {
        sum=a[i][j];
        for (k=1;k<j;k++)
            sum -= a[i][k]*a[k][j];
        a[i][j]=sum;
        if ( (dum=vv[i]*fabs(sum)) >= big)
        {
            big=dum;
            imax=i;
        }
    }
    if (j != imax) {
        for (k=1;k<=n;k++) {
            dum=a[imax][k];
            a[imax][k]=a[j][k];
            a[j][k]=dum;
        }
        *d = -*d;
        vv[imax]=vv[j];
    }
    indx[j]=imax;
    if (a[j][j] == 0.0)
        a[j][j]=TINY;
    if (j != n) {
        dum=1.0/(a[j][j]);
        for (i=j+1;i<=n;i++) a[i][j] *= dum;
    }
}
//*****
void lubksb(double **a, int n, int *indx, double b[])
{
    int i,ii=0,ip,j;
    float sum;
    for (i=1;i<=n;i++)
    {
        ip=indx[i];
        sum=b[ip];
        b[ip]=b[i];
        if (ii)
            for (j=ii;j<=i-1;j++) sum -= a[i][j]*b[j];
        else if (sum) ii=i;
        b[i]=sum;
    }
    for (i=n;i>=1;i--) {
        sum=b[i];
        for (j=i+1;j<=n;j++) sum -= a[i][j]*b[j];
        b[i]=sum/a[i][i];
    }
}
//*****
void lubksb1(int n, int *indx)
{
    int i,ii=0,ip,j;
    float sum;
    for (i=1;i<=n;i++)
    {
        ip=indx[i];
        sum=BB[ip];

```

```

        BB[ip]=BB[i];
        if (ii)
            for (j=ii;j<=i-1;j++) sum -= A[i][j]*BB[j];
        else if (sum) ii=i;
        BB[i]=sum;
    }
    for (i=n;i>=1;i--) {
        sum=BB[i];
        for (j=i+1;j<=n;j++) sum -= A[i][j]*BB[j];
        BB[i]=sum/A[i][i];
    }
}
//*****
void lubksb2(int n, int *indx2)
{
    int i,ii=0,ip,j;
    float sum;
    for (i=1;i<=n;i++)
    {
        ip=indx2[i];
        sum=BB2[ip];
        BB2[ip]=BB2[i];
        if (ii)
            for (j=ii;j<=i-1;j++) sum -= A2[i][j]*BB2[j];
        else if (sum) ii=i;
        BB2[i]=sum;
    }
    for (i=n;i>=1;i--) {
        sum=BB2[i];
        for (j=i+1;j<=n;j++) sum -= A2[i][j]*BB2[j];
        BB2[i]=sum/A2[i][i];
    }
}
//*****
void ludcmp2(double **a2, int n, int *indx2, double *d)
{
    int i,imax,j,k;
    float big,dum,sum,temp;
    float *vv;
    vv=new float[n+1];
    for (i=1;i<=n;i++)
        vv[i]=1;
    *d=1.0;
    for (i=1;i<=n;i++) {
        big=0.0;
        for (j=1;j<=n;j++)
            if ((temp=fabs(a2[i][j])) > big)
                big=temp;
        if (big == 0.0)
            cout<<"Singular matrix in routine ludcmp2 ln";
        vv[i]=1.0/big;
    }
    for (j=1;j<=n;j++)
    {
        for (i=1,i<j;i++) {
            sum=a2[i][j];
            for (k=1;k<i;k++) sum -= a2[i][k]*a2[k][j];
            a2[i][j]=sum;
        }
        big=0.0;
        for (i=j;i<=n;i++)
        {
            sum=a2[i][j];
            for (k=1;k<j;k++)
                sum -= a2[i][k]*a2[k][j];

```

```
        a2[j][j]=sum;
        if ( (dum=vv[j]*fabs(sum)) >= big)
        {
            big=dum;
            imax=j;
        }
    }
    if (j != imax) {
        for (k=1;k<=n;k++) {
            dum=a2[imax][k];
            a2[imax][k]=a2[j][k];
            a2[j][k]=dum;
        }
        *d = -(*d);
        vv[imax]=vv[j];
    }
    indx2[j]=imax;
    if (a2[j][j] == 0.0)
        a2[j][j]=TINY;
    if (j != n) {
        dum=1.0/(a2[j][j]);
        for (i=j+1;i<=n;i++) a2[i][j] *= dum;
    }
}
```

```
// Método de Elemento Finito, Malla Triangular de 3 Nudos, Análisis No Lineal Geométrico
// Método de Elementos de Contacto , Master-Slave , Nudo a Nudo
#include <time.h>
#include <string.h>
#include <stdio.h>
#include "stdlib.h"
#include <fstream.h>
#include "iostream.h"
#include <math.h>
#include <iomanip.h>
#define NDOFN 2
#define NDIM 2
#define NPROP 3
#define NNE 3
#define CLK_TCK 1000.0
typedef char cstring[20];
int NN,NE;
int aBanda;
double **coord=NULL;
double **coordini=NULL;
int **support=NULL;
double **load=NULL;
int **connect=NULL;
double **prop=NULL;
int *elemtype=NULL;
int ndof,band,ndofe;
int LABEL[NDOFN*NNE];
double ESM[NDOFN*NNE][NDOFN*NNE];
double ERHS[NDOFN*NNE];
double D[3][3], B[3][6], DB[3][6], BDB[6][6], Go[4][6], So[4][4], GS[6][4], GSG[6][6], Kg[6][6], KI[6][6];
double **ESF=NULL;
double **A=NULL;
double **A1=NULL;
double *X=NULL;
double *BB=NULL;
double *BBx=NULL;
double *BBy=NULL;
double *BBxy=NULL;
double **EsfANP=NULL;
double **EsfANMC=NULL;
double **ESFUER=NULL;
double BIGg,BIGgkr;
double Scomp,Strac;
double Spark[21][2];
double *RHS;
double *DESPLA;
double *fonzi=NULL;
double **P=NULL; //arreglo de carga incremental
double *disp=NULL;
double **vectorc1; //vector direccional del contacto inicial
double **vectorc2; //vector direccional del contacto final
double tol, tolc;
int nitepp, //# numero de iteraciones paso-paso
int iterpp; //variable para contar las veces de paso-paso
int *pos=NULL;
double *DIAGONAL=NULL;
double ** FILA=NULL;
double ** COLUMNA=NULL;
double ** KRG=NULL;
double *U=NULL;
int min1(int i,int j);
int max1(int i,int j);
void GetInput(cstring s);
void SetBandSystem();
void GetConnect(int elemno);
```

```
void Assemble(int elemno);
void GetStiff(int elemno);
void ApplySupport();
void Solve();
void Soluc();
void PrintTime(char file[20]);
void PrintData(char file[20]);
void PrintOut(char file[20]);
void PrintOutDesp(char file[20], int iterc);
void PrintSal(char file[20], int elemno);
void PrintSalM(char file[20]);
void PrintSalM1(char file[20]);
void PrintEsf(char file[20], int iterc);
void PrintEsfAlizado(char file[20]);
void ClearMemory();
void PrintResult(char file[20]);
void PrintElas(char file[20], int iterc);
void CalculoError();
void PrintError(char file[20]);
double error1_elem, error2_ener;
double *Param_Local=NULL;
int Kposi(int i, int j);
double &Kpos(int i, int j);
void ClearVariant();
void ludcmp(double **a, int n, int *indx, double *d);
void lubksb(double **a, int n, int *indx, double b[]);
void lubksb1(int n, int *indx);
int *indx;
#define TINY 1.0e-20
int iterc; //variable para contar las veces de contacto
int nitemc; // # de iteracion metodo contacto
int ncont; //numero de elementos de contacto
double time_ini, time_fin;
int continua, activa;

int main(int argc, char* argv[])
{
    int i,j,k,is,conta1pp,ipp;
    int qqqq;
    double xfi,xfj,yfi,yfj;
    int elem_1;
    double x1,x2,x3,y1,y2,y3,are_1,are_2,are_3;
    char file[20];
    char archivo[40];
    char archivo1[40];
    char archivo2[40];
    cout <<"INGRESE ARCHIVO DE ENTRADA: (incluya extension)";
    cin >> file;
    tolc=0.05; //tolerancia
    iterc=1; // inicia la iteracion con 1
    nitemc=1; //numero de iteraciones (por defecto es 1, pero si hay elemcontac vale 2, luego verifica)
    iterpp=1;
    activa=0;
    continua=1;
    GetInput(file);
    cout <<"INGRESE NOMBRE DE ARCHIVOS DE SALIDA (sin extension): ";
    cin >> file;
    strcpy( archivo, file);
    strcat( archivo, "_matrix.jg1");
    cout << archivo << "\n";
    strcpy( archivo1, file);
    strcat( archivo1, "_out.jgv");
    cout << archivo1 << "\n";
    strcpy( archivo2, file);
    strcat( archivo2, "_adap.adh");
```

```

cout << archivo2 << "\n";
time_ini=clock()/CLK_TCK;
SetBandSystem();
PrintData(archivo1);
for(i=0;i<NE;i++) // verifica al inicio si los cuerpos estan contactados
{
    if(elemtype[i]==1)
    {
        xxfi=DESPLA[connect[i][0]*2]+coordini[connect[i][0]][0];
        xxfj=DESPLA[connect[i][1]*2]+coordini[connect[i][1]][0];
        yyfi=DESPLA[connect[i][0]*2+1]+coordini[connect[i][0]][1];
        yyfj=DESPLA[connect[i][1]*2+1]+coordini[connect[i][1]][1];
        vectorc1[i][0] = xxfj-xxfi;
        vectorc1[i][1] = yyfj-yyfi;
        if(vectorc1[i][0]==0 && vectorc1[i][1]==0)
        {
            activa=1;
            printf("Activa=1: \n");
        }
    }
}
// [1].- Inicio de las Iteraciones para el Contacto
for (ipp=0;ipp<nitepp;ipp++)
{
    ClearVariant();
    for (i=0;i<NE;i++)
    {
        Assemble(i);
        PrintSal(archivo, i);
    }
    PrintSalM(archivo);
    ApplySupport();
    PrintSalM1(archivo);
    // calcula el vector direccional del contacto inicial
    if(continua==1)
    {
        for(i=0;i<NE;i++)
        {
            if(elemtype[i]==1)
            {
                xxfi=DESPLA[connect[i][0]*2]+coordini[connect[i][0]][0];
                xxfj=DESPLA[connect[i][1]*2]+coordini[connect[i][1]][0];
                yyfi=DESPLA[connect[i][0]*2+1]+coordini[connect[i][0]][1];
                yyfj=DESPLA[connect[i][1]*2+1]+coordini[connect[i][1]][1];
                vectorc1[i][0] = xxfj-xxfi;
                vectorc1[i][1] = yyfj-yyfi;
            }
        }
        aBanda=0;
        for(i=0;i<ndof;i++)
            if (aBanda<pos[i])
                aBanda=pos[i];
        Solve();
        for(i=0;i<ndof;i++)
            DESPLA[i] = DESPLA[i] + RHS[i];
        // verifica si hay cambio en el vector direccion del contacto y/o existe penetracion de
        // un pto a un elemento triangular
        // calcula el vector direccional del contacto inicial
        for(i=0;i<NE;i++)
        {
            if(elemtype[i]==1)
            {
                xxfi=DESPLA[connect[i][0]*2]+coordini[connect[i][0]][0];
                xxfj=DESPLA[connect[i][1]*2]+coordini[connect[i][1]][0];
            }
        }
    }
}

```

```

        yyfi=DESPLA[connect[i][0]*2+1]+coordini[connect[i][0]][1];
        yyfj=DESPLA[connect[i][1]*2+1]+coordini[connect[i][1]][1];
        vectorc2[i][0]=xxfj-xxfi;
        vectorc2[i][1]=yyfj-yyfi;
    }
}
activa=0;
for(i=0;i<NE;i++)
{
    if(elemtype[i]==1)
    {
        printf("Vector 1 x: %f\n",vectorc1[i][0]);
        printf("Vector 1 y: %f\n",vectorc1[i][1]);
        printf("Vector 2 x: %f\n",vectorc2[i][0]);
        printf("Vector 2 y: %f\n",vectorc2[i][1]);
        if(vectorc1[i][0]*vectorc2[i][0]<0)
        {
            continua=0;
            ipp=ipp-1;
            activa=1;
            printf("Error, sale por la 1º condicion: \n");
            break;
        }
        if(vectorc1[i][1]*vectorc2[i][1]<0)
        {
            continua=0;
            ipp=ipp-1;
            activa=1;
            printf("Error, sale por la 2º condicion: \n");
            break;
        }
        if(vectorc1[i][0]*vectorc2[i][0]>=0 && vectorc1[i][1]*vectorc2[i][1]>=0)
        {
            continua=1;
            printf("Esta todo bien, pasa a la sgte iteracion: \n");
        }
    }
}
if(continua==0)
    for(i=0;i<ndof;i++)
        DESPLA[i] = DESPLA[i] - RHS[i];
if(continua==1)
{
    PrintOutDesp(archivo1,iterc);
    PrintEsf(archivo1,iterc);
    for(i=0;i<NN;i++)
    {
        for(j=0;j<NDOFN;j++)
        {
            P[i][j]=load[i][j]/nitepp;
            coord[i][0]=coordini[i][0]+DESPLA[i]*2;
            coord[i][1]=coordini[i][1]+DESPLA[i]*2+1;
        }
    }
    iterc=iterc+1;
}
cout <<"**** Fin de " << iterc-1 << "º Iteracion \n";
cin >>qqqqq;
}
// [1].- Fin de las Iteraciones para el Contacto
PrintResult(archivo1);
PrintOut(archivo);
PrintEsfAlizado(archivo1);
CalculoError();
PrintError(archivo2);

```

```

ClearMemory();
time_fin=clock()/CLK_TCK;
PrintTime(archivo1);
cout << "Tiempo total: " << (time_fin-time_ini) << " segundos\n";
cin >> file;
return 0;
}
////////////////////////////////////
int min1(int i,int j)
{
    if(i<j) return i;
    return j;
}
////////////////////////////////////
int max1(int i,int j)
{
    if(i>j) return i;
    return j;
}
////////////////////////////////////
void GetInput(cstring s)
{
    ifstream inp;
    int i,j,k;
    int qqqqq;
    inp.open(s,ios::in);
    inp >> NN;
    printf("N de nudos : %d\n",NN);
    inp >> NE;
    printf("N de elementos : %d\n",NE);
    inp >> nitepp;
    cout << "*****";
    coord = new double*[NN];
    for (i=0;i<NN;i++)
        coord[i]=new double[NDIM];
    coordini= new double*[NN];
    for (i=0;i<NN;i++)
        coordini[i]=new double[NDIM];
    support = new int*[NN];
    for (i=0;i<NN;i++)
        support[i]=new int[NDOFN];
    load = new double*[NN];
    for (i=0;i<NN;i++)
        load[i]=new double[NDOFN];
    P = new double*[NN];
    for (i=0;i<NN;i++)
        P[i] = new double[NDOFN];
    for(i=0;i<NN;i++)
    {
        inp >> k;
        k--;
        for(j=0;j<NDIM;j++)
            inp >> coord[k][j];
        for(j=0;j<NDOFN;j++)
            inp >> support[k][j];
        for(j=0;j<NDOFN;j++)
            inp >> load[k][j];
    }
    connect = new int*[NE];
    for (i=0;i<NE;i++)
        connect[i]=new int[NNE];
    prop = new double*[NE];
    for (i=0;i<NE;i++)
        prop[i]=new double[NPROP];
    elemtype = new int [NE];
}

```

```

for(i=0;i<NE;i++)
{
    inp >> k;
    k--;
    for(j=0;j<NNE;j++)
    {
        inp >> connect[k][j];
        connect[k][j]--;
    }
    inp >> elemtype[k];
    for(j=0;j<NPROP;j++)
        inp >> prop[k][j];
}
inp >> BIGg;
inp >> BIGgkr;
}
////////////////////////////////////
void SetBandSystem()
{
    int i,j,k;
    int qqqqq;
    ndof = NDOFN*NN;
    ndofe = NDOFN*NNE;
    band = 0;
    pos = new int[ndof];
    printf("ndof: %d\n",ndof);
    printf("ndofe: %d\n",ndofe);
    cout <<"*****";
    for(i=0;i<ndof;j++)
        pos[i]=0;
    for(i=0;i<NE;i++)
    {
        GetConnect(i);
        if (elemtype[i]==0 || elemtype[i]==2)
        {
            for(j=0;j<ndofe;j++)
            {
                for(k=0;k<j;k++)
                {
                    int l1=LABEL[j];
                    int l2=LABEL[k];
                    int dif=abs(l2-l1);
                    int l3=max1(l1,l2);
                    if (pos[l3]<dif)
                        pos[l3]=dif;
                }
            }
        }
        else
        {
            nitemc=10; //por defecto es 1 iter, pero como hay elementcontact iterara 10 veces
            for(j=0;j<NDOFN*2;j++)
            {
                for(k=0;k<j;k++)
                {
                    int l1=LABEL[j];
                    int l2=LABEL[k];
                    int dif=abs(l2-l1);
                    int l3=max1(l1,l2);
                    if (pos[l3]<dif)
                        pos[l3]=dif;
                }
            }
        }
    }
}

```

```

DIAGONAL = new double [ndof];
COLUMNA = new double* [ndof];
FILA = new double* [ndof];
for(i=0;i<ndof;i++)
{
    COLUMNA[i]=NULL;
    FILA[i]=NULL;
    j=pos[i];
    if (j>0)
    {
        COLUMNA[i]= new double[j];
        FILA[i]= new double[j];
    }
}
for(i=0;i<ndof;i++)
{
    DIAGONAL[i]=0.0;
    for(j=0;j<pos[i];j++)
    {
        COLUMNA[i][j]=0.;
        FILA[i][j]=0.;
    }
}
disp = new double [ndof];
for(i=0;i<ndof;i++)
    disp[i] = 0.0;
RHS = new double [ndof];
for(i=0;i<ndof;i++)
    RHS[i] = 0.0;
for(i=0;i<NN;i++)
    for(j=0;j<NDOFN;j++)
        P[i][j]=load[i][j]/nitepp;
DESPLA= new double [ndof];
for(i=0;i<ndof;i++)
    DESPLA[i] = 0.0;
ESFUER = new double*[NE];
for (i=0;i<NE;i++)
    ESFUER[i]=new double[NNE];
for (i=0;i<NE;i++)
    for (j=0;j<NNE;j++)
        ESFUER[i][j]=0.0;
ncont=0;
for (i=0;i<NE;i++)
    if (elemtype[i]==1)
        ncont=ncont+1;
for(i=0;i<NN;i++)
    for(j=0;j<NDIM;j++)
        coordini[i][j] = coord[i][j];
vectorc1 = new double*[NE];
for (i=0;i<NE;i++)
    vectorc1[i] = new double[2];
vectorc2 = new double*[NE];
for (i=0;i<NE;i++)
    vectorc2[i] = new double[2];
}
////////////////////////////////////
void GetConnect(int elemno)
{
    int i,j;
    int aux;
    aux=1;
    if (elemtype[elemno]==0 || elemtype[elemno]==2)
    {
        for(i=0;i<NNE;i++)
            for(j=0;j<NDOFN;j++)

```

```

        LABEL[i*NDOFN+j] = connect[elemno][i]*NDOFN+(j);
    }
    else
    {
        for(i=0;i<2;i++)
            for(j=0;j<NDOFN;j++)
                LABEL[i*NDOFN+j] = connect[elemno][i]*NDOFN+(j);
    }
}
////////////////////////////////////
void Assemble(int elemno)
{
    int i,j;
    GetConnect(elemno);
    GetStiff(elemno);
    for(i=0;i<ndofe;i++)
    {
        DIAGONAL[LABEL[i]] += ESM[i][i];
        RHS[LABEL[i]] += ERHS[i];
    }
    for(i=1;i<ndofe;i++)
    {
        for(j=0;j<i;j++)
        {
            if (LABEL[j]<=LABEL[i])
                FILA[LABEL[i]][abs(LABEL[j]-LABEL[i])-1] += ESM[i][j];
            if (LABEL[j]>LABEL[i])
                FILA[LABEL[j]][abs(LABEL[j]-LABEL[i])-1] += ESM[i][j];
        }
    }
    for(j=1;j<ndofe;j++)
    {
        for(i=0;i<j;i++)
        {
            if(LABEL[i]<=LABEL[j])
                COLUMNA[LABEL[j]][abs(LABEL[i]-LABEL[j])-1] += ESM[j][i];
            if(LABEL[i]>LABEL[j])
                COLUMNA[LABEL[i]][abs(LABEL[i]-LABEL[j])-1] += ESM[j][i];
        }
    }
}
////////////////////////////////////
void GetStiff(int elemno)
{
    int i,j,k,kode;
    double coef1,coef2;
    double x1,y1,x2,y2,x3,y3,x13,y23,x23,y13,y31,y12,x32,x21;
    double detJ,areas;
    double xx1,yy1,xx2,yy2;
    double xx1f,yy1f,xx2f,yy2f,vectnxf,vectnyf;
    double vectnx,vectny,l0,lf,kresorte;
    kode=elemtype[elemno];
    if(kode==0 || kode==2)
    {
        // Inicio Matrix del Elemento Finito
        // Matrix "D"
        if(kode==0) //Para el caso de TENSION PLANA
        {
            coef1=prop[elemno][0]/(1-(prop[elemno][2]*prop[elemno][2]));
            D[0][0]=coef1*1;
            D[0][1]=coef1*prop[elemno][2];
            D[0][2]=0;
            D[1][0]=coef1*prop[elemno][2];
            D[1][1]=coef1*1;
            D[1][2]=0;
        }
    }
}

```

```

D[2][0]=0;
D[2][1]=0;
D[2][2]=coef1*(1-prop[elemno][2])/2;
}
if(kode==2) //Para el caso de DEFORMACION PLANA
{
coef1=prop[elemno][0]*(1-prop[elemno][2])/(1+prop[elemno][2])/(1-2*prop[elemno][2]);
D[0][0]=coef1*1;
D[0][1]=coef1*prop[elemno][2]/(1-prop[elemno][2]);
D[0][2]=0;
D[1][0]=coef1*prop[elemno][2]/(1-prop[elemno][2]);
D[1][1]=coef1*1;
D[1][2]=0;
D[2][0]=0;
D[2][1]=0;
D[2][2]=prop[elemno][0]/2/(1+prop[elemno][2]);
}
// Determinante Jacobiano
x1=coord[connect[elemno][0]][0];
y1=coord[connect[elemno][0]][1];
x2=coord[connect[elemno][1]][0];
y2=coord[connect[elemno][1]][1];
x3=coord[connect[elemno][2]][0];
y3=coord[connect[elemno][2]][1];
x13=x1-x3;
y23=y2-y3;
x23=x2-x3;
y13=y1-y3;
detJ=x13*y23-x23*y13;
if (detJ<0)
detJ=(detJ)*(-1);
areas=0.5*detJ;
coef2=1/detJ;
y12=y1-y2;
y31=y3-y1;
x32=x3-x2;
x21=x2-x1;
// Matrix "B"
B[0][0]=coef2*y23;
B[0][1]=0;
B[0][2]=coef2*y31;
B[0][3]=0;
B[0][4]=coef2*y12;
B[0][5]=0;
B[1][0]=0;
B[1][1]=coef2*x32;
B[1][2]=0;
B[1][3]=coef2*x13;
B[1][4]=0;
B[1][5]=coef2*x21;
B[2][0]=coef2*x32;
B[2][1]=coef2*y23;
B[2][2]=coef2*x13;
B[2][3]=coef2*y31;
B[2][4]=coef2*x21;
B[2][5]=coef2*y12;
for (i=0;i<3;i++)
for (j=0;j<6;j++)
DB[i][j]=0;
for (i=0;i<6;i++)
for (j=0;j<6;j++)
BDB[i][j]=0;
// Matrix DB = D x B
for (i=0;i<3;i++)
for (j=0;j<6;j++)

```

```

        for (k=0;k<3;k++)
            DB[i][j]=DB[i][j]+D[i][k]*B[k][j];
// Matrix B'DB = B' x DB
for (i=0;i<6;i++)
    for (j=0;j<6;j++)
        for (k=0;k<3;k++)
            BDB[i][j]=BDB[i][j]+B[k][i]*DB[k][j];
// Matrix KI = espesor x area x B'DB
for (i=0;i<6;i++)
    for (j=0;j<6;j++)
        KI[i][j]=prop[elemno][1]*areas*BDB[i][j];
//Calculo de la Matrix Geometrica Kg=Go.Esf.Go
// Matrix Go
Go[0][0]=B[0][0];
Go[0][1]=0;
Go[0][2]=B[0][2];
Go[0][3]=0;
Go[0][4]=B[0][4];
Go[0][5]=0;
Go[1][0]=B[1][1];
Go[1][1]=0;
Go[1][2]=B[1][3];
Go[1][3]=0;
Go[1][4]=B[1][5];
Go[1][5]=0;
Go[2][0]=0;
Go[2][1]=B[0][0];
Go[2][2]=0;
Go[2][3]=B[0][2];
Go[2][4]=0;
Go[2][5]=B[0][4];
Go[3][0]=0;
Go[3][1]=B[1][0];
Go[3][2]=0;
Go[3][3]=B[1][2];
Go[3][4]=0;
Go[3][5]=B[1][4];
//Matrix Esfuerzo de Cauchy
So[0][0]=ESFUER[elemno][0];
So[0][1]=ESFUER[elemno][2];
So[0][2]=0;
So[0][3]=0;
So[1][0]=ESFUER[elemno][2];
So[1][1]=ESFUER[elemno][1];
So[1][2]=0;
So[1][3]=0;
So[2][0]=0;
So[2][1]=0;
So[2][2]=ESFUER[elemno][0];
So[2][3]=ESFUER[elemno][2];
So[3][0]=0;
So[3][1]=0;
So[3][2]=ESFUER[elemno][2];
So[3][3]=ESFUER[elemno][1];
for(i=0;i<6;i++)
    for(j=0;j<4;j++)
        GS[i][j]=0;
for(i=0;i<6;i++)
{
    for(j=0;j<4;j++)
        for (k=0;k<4;k++)
            GS[i][j]=GS[i][j]+Go[k][i]*So[k][j];
}
for(i=0;i<6;i++)
    for(j=0;j<6;j++)

```

```

                                GSG[i][j]=0;
for(i=0;j<6;j++)
{
    for(j=0;j<6;j++)
        for (k=0;k<4;k++)
            GSG[i][j]=GSG[i][j]+GS[i][k]*Go[k][j];
}
// Matrix Kg = espesor x area x GSG
for(i=0;i<6;i++)
    for(j=0;j<6;j++)
        Kg[i][j]=0;
for (i=0;i<6;i++)
    for (j=0;j<6;j++)
        Kg[i][j]=prop[elemno][1]*areas*GSG[i][j];
// Matrix Ktotal = Kl + Kg
for (i=0;i<6;i++)
    for (j=0;j<6;j++)
        ESM[i][j]=Kl[i][j]+Kg[i][j];
// Fin Matrix del Elemento Finito
for (i=0;i<6;i++)
    ERHS[i]=0.0;
}
else
{
    for (i=0;i<6;i++)
        for (j=0;j<6;j++)
            ESM[i][j]=0;
    for (j=0;j<6;j++)
        ERHS[j]=0;
    if(activa==1)
    {
        //incrementa 10% unid para cumplir la condic impenetra
        prop[elemno][2]=prop[elemno][2]*10;
        kresorte=prop[elemno][2];
        l0=sqrt(vectorc2[elemno][0]*vectorc2[elemno][0]+vectorc2[elemno][1]*vectorc2[elemno][1]);
        printf("Lo: %f\n",l0);
        printf("Kresorte: %f\n",kresorte);
        ESM[0][0]=kresorte*(-vectorc2[elemno][0])*(-vectorc2[elemno][0])/l0/l0;
        ESM[0][1]=kresorte*(-vectorc2[elemno][0])*(-vectorc2[elemno][1])/l0/l0;
        ESM[0][2]=-ESM[0][0];
        ESM[0][3]=-ESM[0][1];
        ESM[1][0]=ESM[0][1];
        ESM[1][1]=kresorte*(-vectorc2[elemno][1])*(-vectorc2[elemno][1])/l0/l0;
        ESM[1][2]=-ESM[1][0];
        ESM[1][3]=-ESM[1][1];
        ESM[2][0]=ESM[0][2];
        ESM[2][1]=ESM[1][2];
        ESM[2][2]=-ESM[2][0];
        ESM[2][3]=-ESM[2][1];
        ESM[3][0]=ESM[0][3];
        ESM[3][1]=ESM[1][3];
        ESM[3][2]=-ESM[0][3];
        ESM[3][3]=-ESM[1][3];
        // para la matrix de fuerza del elemento: F=K.n.l; l=difer(nudos)
        ERHS[0]=kresorte*(-vectorc2[elemno][0])/l0*10;
        ERHS[1]=kresorte*(-vectorc2[elemno][1])/l0*10;
        ERHS[2]=-ERHS[0];
        ERHS[3]=-ERHS[1];
        ERHS[4]=0;
        ERHS[5]=0;
    }
}
}
////////////////////////////////////
void ApplySupport()
{

```

```

int i,j,pos;
for(i=0;i<NN;i++)
    for(j=0;j<NDOFN;j++)
    {
        pos = i*NDOFN+j;
        if(support[i][j]==1)
        {
            DIAGONAL[pos] +=BIGg;
            RHS[pos] += BIGg*P[i][j];
        }
        else
            RHS[pos] += P[i][j];
    }
}
////////////////////////////////////
void Solve()
{
    int i,j,k,kk;
    double pivot;
    double suma;
    for(i=1;i<ndof;i++)
    {
        if(pos[i] != 0)
        {
            for(j=pos[i];j>0;j--)
            {
                if(FILA[i][j-1] != 0)
                {
                    pivot=(FILA[i][j-1])/DIAGONAL[i-j];
                    FILA[i][j-1] = 0;
                    for (k=1;k<j;k++)
                        if(pos[(i-j)+k] >= k)
                            FILA[i][j-1-k] += -COLUMNA[i-j+k][k-1]*pivot;
                    if(COLUMNA[i][j-1] != 0)
                        DIAGONAL[i] += -COLUMNA[i][j-1]*pivot;
                    kk=(i-j)+aBanda;
                    if(kk>ndof)
                        kk=ndof-1;
                    kk=kk-i;
                    for(k=0;k<kk;k++)
                        if ((k+1) < pos[i+(k+1)])
                            if(j<pos[i+k+1]-k) // verifica que tenga valor
                                COLUMNA[i+k+1][k] += -COLUMNA[i+k+1][j+k]*pivot;
                    RHS[i] += -RHS[i-j]*pivot;
                }
            }
        }
    }
    for(i=ndof-1;i>=0;i--)
    {
        double Kij;
        suma=0.0;
        for(j=i+1;j<ndof;j++)
            if(Kpos(i,j)) {
                Kij=Kpos(i,j);
                suma += Kij*RHS[j];
            }
        RHS[i] = (RHS[i]-suma)/Kpos(i,i);
    }
}
////////////////////////////////////
void PrintOut(char file[20])
{
    int i,j;
    ofstream out;
}

```

```

out.open(file,ios::app);
out << " \n";
out << "**** RESULTADOS FINALES DE MATRICES AUXILIARES **** << " \n";
out << "MATRIZ DIAGONAL" << " \n";
for(i=0;i<ndof;i++)
    out << DIAGONAL[i] << "\t";
out << " \n";
out << " \n";
out << "MATRIZ FILA" << " \n";
for(i=1;j<ndof;j++)
{
    out << "posicion: " << i << " , Tiene [" << pos[i] << "] \n";
    for(j=0;j<pos[i];j++)
        out << FILA[i][j] << "\t";
    out << " \n";
}
out << " \n";
out << "MATRIZ COLUMNA" << " \n";
for(i=1;j<ndof;j++)
{
    out << "posicion: " << i << " , Tiene [" << pos[i] << "] \n";
    for(j=0;j<pos[i];j++)
        out << COLUMNA[i][j] << "\t";
    out << " \n";
}
out << " \n";
out << "MATRIZ RHS:" << " \n";
for(i=0;i<ndof;i++)
    out << "RHS [" << i << "]: " << RHS[i] << " \n";
out << " \n";
}
////////////////////////////////////
void PrintOutDesp(char file[20], int iterc)
{
    int i,j;
    ofstream mistream;
    mistream.open(file,ios::in|ios::app);
    mistream << " \n";
    mistream << "ELEMENTO TRIANGULAR DE 3 PUNTOS - PENALIDAD" << " \n";
    mistream << " \n";
    mistream << "DESPLAZAMIENTO : Iteracion [" << iterc << "] \n";
    mistream << "NUDO" << "\t";
    mistream << setw(14) << "UX" << "\t";
    mistream << setw(14) << "UY" << " \n";
    for (i=0;i<NN;i++)
    {
        mistream << i+1 << "\t";
        for (j=0;j<NDOFN;j++)
            mistream << setw(15) << RHS[i*NDOFN+j] << "\t";
        mistream << " \n";
    }
    for (i=0;i<NE;i++)
    {
        if (elemtype[i]==1)
        {
            mistream << setw(15) << i << "\t";
            mistream << setw(15) << prop[i][2] << " \n";
        }
    }
    mistream << " \n";
}
////////////////////////////////////
void ClearMemory()
{
    int i;

```

```

    for (i=0;i<NN;i++)
        delete [] coord[i];
    delete [] coord;
    for (i=0;i<NN;i++)
        delete [] support[i];
    delete [] support;
    for (i=0;i<NN;i++)
        delete [] load[i];
    delete [] load;
    for (i=0;i<NE;i++)
        delete [] connect[i];
    delete [] connect;
    for (i=0;i<NE;i++)
        delete [] prop[i];
    delete [] prop;
    delete [] elemtype;
    delete [] DIAGONAL;
    for(i=0;i<ndof;i++)
    {
        if (COLUMNA[i]!=NULL)
            delete COLUMNA[i];
        if (FILA[i]!=NULL)
            delete FILA[i];
    }
    delete [] COLUMNA;
    delete [] FILA;
    for (i=0;i<NE;i++)
        delete [] ESF[i];
    delete [] ESF;
}
/////////////////////////////////////////////////////////////////
double &Kpos(int i, int j)
{
    if (i==j)
        return DIAGONAL[i];
    else if (j>i)
        return COLUMNA[j][i-i-1];
    else
        return FILA[i][i-j-1];
}
int Kposi(int i, int j)
{
    if (i==j)
        return 1;
    else if (j>i)
    {
        if ((j-i)>pos[j])
            return 0;
        else
            return 1;
    }
    else
    {
        if ((i-j)>pos[i])
            return 0;
        else
            return 1;
    }
}
/////////////////////////////////////////////////////////////////
void ClearVariant()
{
    int i,j;
    for(i=0;i<ndof;i++)
    {

```

```

        DIAGONAL[i]=0.0;
        for(j=0;j<npos[i];j++)
        {
            COLUMNA[i][j]=0.;
            FILA[i][j]=0.;
        }
    }
    for(i=0;i<ndof;i++)
        RHS[i] = 0.0;
}
////////////////////////////////////
void PrintSal(char file[20], int elemno)
{
    int i,j,kode;
    ofstream mistream;
    mistream.open(file,ios::in|ios::app);
    mistream << "\n";
    kode=elemtype[elemno];
    if(kode==0 || kode==2)
    {
        mistream << "MATRIZ DB : [" << elemno+1 << "]"<< "\n";
        for (i=0;i<3;i++)
        {
            for (j=0;j<6;j++)
                mistream << setw(12.2) << DB[i][j] << "\t";
            mistream << "\n";
        }
        mistream << "\n";
        mistream << "MATRIZ BDB : [" << elemno+1 << "]"<< "\n";
        for (i=0;i<6;i++)
        {
            for (j=0;j<6;j++)
                mistream << setw(12) << BDB[i][j] << "\t";
            mistream << "\n";
        }
        mistream << "\n";
        mistream << "MATRIZ RIGIDEZ LINEAL: [" << elemno+1 << "]"<< "\n";
        for (i=0;i<6;i++)
        {
            for (j=0;j<6;j++)
                mistream << setw(12) << KI[i][j] << "\t";
            mistream << "\n";
        }
        mistream << "\n";
        mistream << "MATRIZ RIGIDEZ GEOMETRICA: [" << elemno+1 << "]"<< "\n";
        for (i=0;i<6;i++)
        {
            for (j=0;j<6;j++)
                mistream << setw(12) << GSG[i][j] << "\t";
            mistream << "\n";
        }
        mistream << "\n";
        mistream << "\n";
    }
    else
    {
        mistream << "MATRIZ RIGIDEZ RESORTE: [" << elemno+1 << "]"<< "\n";
        for (i=0;i<4;i++)
        {
            for (j=0;j<4;j++)
                mistream << setw(12.2) << ESM[i][j] << "\t";
            mistream << "\n";
            mistream << setw(12.2) << ERHS[i] << "\n";
        }
        mistream << "\n";
    }
}

```

```

        mistream << "\n";
    }
}
////////////////////////////////////
void PrintSalM(char file[20])
{
    int i,j;
    ofstream out;
    out.open(file,ios::app);

    out << "MATRIZ FILA : " << "\n";
    for (i=0;i<ndof;i++)
    {
        out << pos[i] << "\t";
        for (j=0;j<pos[i];j++)
            out << FILA[i][j] << "\t";
        out << "\n";
    }
    out << "\n";
    out << "\n";
    out << "MATRIZ COLUMNA : " << "\n";
    for (i=0;i<ndof;i++)
    {
        out << pos[i] << "\t";
        for (j=0;j<pos[i];j++)
            out << COLUMNA[i][j] << "\t";
        out << "\n";
    }
    out << "\n";
    out << "\n";
    out << "MATRIZ DIAGONAL sin la Carga del RESORTE: " << "\n";
    for(i=0;i<ndof;i++)
        out << DIAGONAL[i] << "\t";
    out << "\n";
    out << "\n";
    out << "MATRIZ FUERZA Inicial: " << "\n";
    for(i=0;i<ndof;i++)
        out << RHS[i] << "\t";
    out << "\n";
    out << "\n";
}
////////////////////////////////////
void PrintSalM1(char file[20])
{
    int i;
    ofstream out;
    out.open(file,ios::app);
    out << "MATRIZ DIAGONAL con la Carga del RESORTE: " << "\n";
    for(i=0;i<ndof;i++)
        out << DIAGONAL[i] << "\t";
    out << "\n";
    out << "\n";
    out << "MATRIZ FUERZA Final: " << "\n";
    for(i=0;i<ndof;i++)
        out << RHS[i] << "\t";
    out << "\n";
    out << "\n";
}
////////////////////////////////////
void PrintEsf(char file[20], int iterc)
{
    int i,j,k;
    double sxx[3],syy[3],sxy[3];
    fstream mistream;
    mistream.open(file,ios::in|ios::app);

```

```

cout << "**** Inicia calculo de Esfuerzos \n";
mistream << " \n";
mistream << " \n";
mistream << "**** TABLA DE ESFUERZO EN CADA ELEMENTO ***\n";
mistream << "\t \t Sxx\t \t Syy\t \t Sxy\n";
ESF = new double*[NE];
for (i=0;i<NE;i++)
    ESF[i]=new double[3];

for (i=0;i<NE;i++)
{
    if (elemtype[i]==0 || elemtype[i]==2)
    {
        GetConnect(i);
        GetStiff(i);
        mistream << "Elem " << i+1 << " :\t";
        for(j=0;j<NNE;j++)
        {
            ESF[i][j]=0.0;
            for (k=0;k<ndofe;k++)
                ESF[i][j]=DB[i][k]*RHS[LABEL[k]]+ESF[i][j];
            mistream << setw(15) << ESF[i][j] << "\t";
        }
        mistream << "\n";
    }
}
cout << "**** Fin de calculo de Esfuerzos \n";
}
////////////////////////////////////
void PrintEsfAlizado(char file[20])
{
    int i,j,k,l;
    int conta;
    int jj,kk;
    int NEN[10][2]; //numero de elementos en un nudo, 10 es el valor estimado y 2 es valor de elem y pos
    double x1,x2,x3,y1,y2,y3,areas,detJ;
    double M1,M2,M3;
    double mini0,mini1,mini2,maxi0,maxi1,maxi2;
    int elema0,elema1,elema2,elemi0,elemi1,elemi2;
    EsfANP = new double*[NN];
    for (i=0;i<NN;i++)
        EsfANP[i]=new double[3];
    EsfANMC = new double*[NN];
    for (i=0;i<NN;i++)
        EsfANMC[i]=new double[3];
    A = new double*[NN+ncont+1];
    for (i=0;i<=NN+ncont;i++)
        A[i]=new double[NN+ncont+1];
    A1 = new double*[NN+ncont];
    for (i=0;i<NN+ncont;i++)
        A1[i]=new double[NN+ncont];
    BB=new double[NN+ncont+1];
    BBx=new double[NN+ncont];
    BBy=new double[NN+ncont];
    BBxy=new double[NN+ncont];
    X=new double[NN+ncont];
    for (i=0;i<NN+ncont;i++)
    {
        BB[i+1]=0;
        BBx[i]=0;
        BBy[i]=0;
        BBxy[i]=0;
        X[i]=0;
        for (j=0;j<NN+ncont;j++)
    
```

```

    {
        A[i+1][j+1]=0;
        A1[i][j]=0;
    }
}
fstream mistream;
mistream.open(file,ios::in|ios::app);
cout <<"***** Inicia calculo de Esfuerzos Alizados : Metodo Promedio \n";
for (i=0;i<NN;i++)
{
    conta=0;
    EsfANP[i][0]=0; //para Sxx
    EsfANP[i][1]=0; //para Syy
    EsfANP[i][2]=0; //para Sxy
    for (j=0;j<10;j++)
    {
        NEN[j][0]=0;
        NEN[j][1]=0;
    }
    for (j=0;j<NE;j++)
    {
        if (elemtype[j]==0 || elemtype[j]==2)
        {
            for (k=0;k<NNE;k++)
            {
                if (connect[j][k]==i)
                {
                    NEN[conta][0]=j; //elemento
                    NEN[conta][1]=k; //una posicion de los 3 nudos
                    conta=conta+1;
                }
            }
        }
    }
    for (l=0;l<conta;l++)
    {
        EsfANP[i][0]=EsfANP[i][0]+ESF[NEN[l][0]][0];
        EsfANP[i][1]=EsfANP[i][1]+ESF[NEN[l][0]][1];
        EsfANP[i][2]=EsfANP[i][2]+ESF[NEN[l][0]][2];
    }
    EsfANP[i][0]=EsfANP[i][0]/conta;
    EsfANP[i][1]=EsfANP[i][1]/conta;
    EsfANP[i][2]=EsfANP[i][2]/conta;
}
cout <<"***** Fin de calculo de Esfuerzos Alizados : Metodo Promedio \n";
cout <<"***** Inicia calculo de Esfuerzos Alizados : Metodo Minimo Cuadrado \n";
conta=0;
for (i=0;i<NE;i++)
{
    if (elemtype[i]==0 || elemtype[i]==2)
    {
        x1=coord[connect[i][0]][0];
        y1=coord[connect[i][0]][1];
        x2=coord[connect[i][1]][0];
        y2=coord[connect[i][1]][1];
        x3=coord[connect[i][2]][0];
        y3=coord[connect[i][2]][1];
        detJ=(x1-x3)*(y2-y3)-(x2-x3)*(y1-y3);
        if (detJ<0)
            detJ=(detJ)*(-1);
        areas=0.5*detJ;
        M1=areas/6;
        M2=areas/12;
        M3=areas/3;
        A1[connect[i][0]][connect[i][0]]=M1+A1[connect[i][0]][connect[i][0]];
    }
}

```

```

A1[connect[i][0]][connect[i][1]]=M2+A1[connect[i][0]][connect[i][1]];
A1[connect[i][0]][connect[i][2]]=M2+A1[connect[i][0]][connect[i][2]];
A1[connect[i][1]][connect[i][0]]=M2+A1[connect[i][1]][connect[i][0]];
A1[connect[i][1]][connect[i][1]]=M1+A1[connect[i][1]][connect[i][1]];
A1[connect[i][1]][connect[i][2]]=M2+A1[connect[i][1]][connect[i][2]];
A1[connect[i][2]][connect[i][0]]=M2+A1[connect[i][2]][connect[i][0]];
A1[connect[i][2]][connect[i][1]]=M2+A1[connect[i][2]][connect[i][1]];
A1[connect[i][2]][connect[i][2]]=M1+A1[connect[i][2]][connect[i][2]];
BBx[connect[i][0]]=M3*ESF[i][0]+BBx[connect[i][0]];
BBx[connect[i][1]]=M3*ESF[i][0]+BBx[connect[i][1]];
BBx[connect[i][2]]=M3*ESF[i][0]+BBx[connect[i][2]];
BBy[connect[i][0]]=M3*ESF[i][1]+BBy[connect[i][0]];
BBy[connect[i][1]]=M3*ESF[i][1]+BBy[connect[i][1]];
BBy[connect[i][2]]=M3*ESF[i][1]+BBy[connect[i][2]];
BBxy[connect[i][0]]=M3*ESF[i][2]+BBxy[connect[i][0]];
BBxy[connect[i][1]]=M3*ESF[i][2]+BBxy[connect[i][1]];
BBxy[connect[i][2]]=M3*ESF[i][2]+BBxy[connect[i][2]];
}
}
for (i=0;i<NN;i++)
{
    BB[i+1]=BBx[i];
    for (j=0;j<NN;j++)
        A[i+1][j+1]=A1[i][j];
}
Soluc();
for (i=0;i<NN;i++)
    EsfANMC[i][0]=BB[i+1]; //esfuerzo en Sxx

for (i=0;i<NN;i++)
{
    BB[i+1]=BBy[i];
    for (j=0;j<NN;j++)
        A[i+1][j+1]=A1[i][j];
}
Soluc();
for (i=0;i<NN;i++)
    EsfANMC[i][1]=BB[i+1]; //esfuerzo en Syy

for (i=0;i<NN;i++)
{
    BB[i+1]=BBxy[i];
    for (j=0;j<NN;j++)
        A[i+1][j+1]=A1[i][j];
}
Soluc();
for (i=0;i<NN;i++)
    EsfANMC[i][2]=BB[i+1]; //esfuerzo en Sxy

cout <<"***** Fin de calculo de Esfuerzos Alizados : Metodo Minimo Cuadrado \n";
mistream << "\n";
mistream << "\n";
mistream << "**** ESFUERZOS ALIZADOS EN EL NUDO : Métodos: Método Promedio , Minimo Cuadrado
**** << "\n";
mistream << "\n";
mistream << "\t \t \t Método Promedio\t \t \t \t Método Minimo Cuadrado\n";
mistream << "\t \t \t Sxx\t \t Syy\t \t Sxy\t \t \t Sxx\t \t Syy\t \t Sxy\n";
for (i=0;i<NN;i++)
{
    mistream << "Nudo " << i+1 << " :\n";
    mistream << setw(15) << EsfANP[i][0] << "\n";
    mistream << setw(15) << EsfANP[i][1] << "\n";
    mistream << setw(15) << EsfANP[i][2] << "\n";
    mistream << "\n";
    mistream << setw(15) << EsfANMC[i][0] << "\n";
    mistream << setw(15) << EsfANMC[i][1] << "\n";
    mistream << setw(15) << EsfANMC[i][2] << "\n";
}

```

```

    }
}
////////////////////////////////////
void Soluc()
{
    double d;
    int nt;
    nt=NN;
    indx=new int[nt];
    ludcmp(A,nt,indx,&d);
    lubksb1(nt,indx);
}
//*****
void ludcmp(double **a, int n, int *indx, double *d)
{
    int i,imax,j,k;
    float big,dum,sum,temp;
    float *vv;
    vv=new float[n+1];
    for (i=1;i<=n;i++)
    {
        vv[i]=1;
    }
    *d=1.0;
    for (i=1;i<=n;i++) {
        big=0.0;
        for (j=1;j<=n;j++)
            if ((temp=fabs(a[i][j])) > big)
                big=temp;
        if (big == 0.0)
            cout<<"Singular matrix in routine ludcmp \n";
        vv[i]=1.0/big;
    }
    for (j=1;j<=n;j++)
    {
        for (i=1;i<j;i++) {
            sum=a[i][j];
            for (k=1;k<i;k++) sum -= a[i][k]*a[k][j];
            a[i][j]=sum;
        }
        big=0.0;
        for (i=j;i<=n;i++)
        {
            sum=a[i][j];
            for (k=1;k<j;k++)
                sum -= a[i][k]*a[k][j];
            a[i][j]=sum;
            if ( (dum=vv[i]*fabs(sum)) >= big)
            {
                big=dum;
                imax=i;
            }
        }
        if (j != imax) {
            for (k=1;k<=n;k++) {
                dum=a[imax][k];
                a[imax][k]=a[j][k];
                a[j][k]=dum;
            }
            *d = -(*d);
            vv[imax]=vv[j];
        }
        indx[j]=imax;
        if (a[j][j] == 0.0)
            a[j][j]=TINY;
    }
}

```

```

        if (j != n) {
            dum=1.0/(a[j][j]);
            for (i=j+1;i<=n;i++) a[i][j] *= dum;
        }
    }
}
//*****
void lubksb(double **a, int n, int *indx, double b[])
{
    int i,ii=0,ip,j;
    float sum;
    for (i=1;i<=n;i++)
    {
        ip=indx[i];
        sum=b[ip];
        b[ip]=b[i];
        if (ii)
            for (j=ii;j<=i-1;j++) sum -= a[j][i]*b[j];
        else if (sum) ii=i;
        b[i]=sum;
    }
    for (i=n;i>=1;i--) {
        sum=b[i];
        for (j=i+1;j<=n;j++) sum -= a[j][i]*b[j];
        b[i]=sum/a[i][i];
    }
}
//*****
void lubksb1(int n, int *indx)
{
    int i,ii=0,ip,j;
    float sum;
    for (i=1;i<=n;i++)
    {
        ip=indx[i];
        sum=BB[ip];
        BB[ip]=BB[i];
        if (ii)
            for (j=ii;j<=i-1;j++) sum -= A[j][i]*BB[j];
        else if (sum) ii=i;
        BB[i]=sum;
    }
    for (i=n;i>=1;i--) {
        sum=BB[i];
        for (j=i+1;j<=n;j++) sum -= A[j][i]*BB[j];
        BB[i]=sum/A[i][i];
    }
}
//*****
void CalculoError()
{
    int i,j;
    double Dinv[3][3];
    double factor;
    double Esfx,Esfy,Esfx,Esfy[NNE];
    double norma_esf;
    double *Norma_Elem=NULL;
    double *Norma_Energia=NULL;
    double EsfD1,EsfD2,EsfD3;
    double x1,y1,x2,y2,x3,y3,area;
    Norma_Elem=new double[NE];
    Norma_Energia=new double[NE];
    Param_Local=new double[NE];
    for (i=0;i<3;i++)

```

```

{
    for (j=0;j<3;j++)
        Dinv[i][j]=0;
}
norma_esf = 0;
for (i=0;i<NE;i++)
{
    if (elemtype[i]==0 || elemtype[i]==2)
    {
        x1 = coord[connect[i][0]][0];
        y1 = coord[connect[i][0]][1];
        x2 = coord[connect[i][1]][0];
        y2 = coord[connect[i][1]][1];
        x3 = coord[connect[i][2]][0];
        y3 = coord[connect[i][2]][1];
        area = (x1-x3)*(y2-y3)-(x2-x3)*(y1-y3);
        if (area<0)
            area = (area)*(-1);
        area = 0.5*area;
        if(elemtype[i]==0)
        {
            factor = (-2)*prop[i][0]/(prop[i][2]-1);
            Dinv[0][0] = factor*1/(2*(1+prop[i][2]));
            Dinv[0][1] = factor*prop[i][2]/(2*(1+prop[i][2]));
            Dinv[1][0] = Dinv[0][1];
            Dinv[1][1] = Dinv[0][0];
            Dinv[2][2] = factor*(-1)*(1+prop[i][2])*(prop[i][2]-1)/(prop[i][0]*prop[i][0]);
        }
        if(elemtype[i]==2)
        {
            factor = (-2)*prop[i][0]/(prop[i][2]-1);
            Dinv[0][0] = factor*1/(2*(1+prop[i][2]));
            Dinv[0][1] = factor*prop[i][2]/(2*(1+prop[i][2]));
            Dinv[1][0] = Dinv[0][1];
            Dinv[1][1] = Dinv[0][0];
            Dinv[2][2] = factor*(-1)*(1+prop[i][2])*(prop[i][2]-1)/(prop[i][0]*prop[i][0]);
        }
        norma_esf=0;
        for (j=0;j<NNE;j++)
        {
            Esfx = (EsfANP[connect[i][j]][0] - Esf[i][0]);
            Esfy = (EsfANP[connect[i][j]][1] - Esf[i][1]);
            Esfxy = (EsfANP[connect[i][j]][2] - Esf[i][2]);
            EsfD1 = Esfx*Dinv[0][0] + Esfy*Dinv[0][1] + Esfxy*Dinv[0][2];
            EsfD2 = Esfx*Dinv[1][0] + Esfy*Dinv[1][1] + Esfxy*Dinv[1][2];
            EsfD3 = Esfx*Dinv[2][0] + Esfy*Dinv[2][1] + Esfxy*Dinv[2][2];
            Esf[j] = EsfD1*Esfx + EsfD2*Esfy + EsfD3*Esfxy;
            norma_esf = norma_esf + Esf[j];
        }
        Norma_Elem[i] = norma_esf*prop[i][1]*area;
        error1_elem = error1_elem + Norma_Elem[i];
        norma_esf=0;
        for (j=0;j<NNE;j++)
        {
            Esfx = EsfANP[connect[i][j]][0];
            Esfy = EsfANP[connect[i][j]][1];
            Esfxy = EsfANP[connect[i][j]][2];
            EsfD1 = Esfx*Dinv[0][0] + Esfy*Dinv[0][1] + Esfxy*Dinv[0][2];
            EsfD2 = Esfx*Dinv[1][0] + Esfy*Dinv[1][1] + Esfxy*Dinv[1][2];
            EsfD3 = Esfx*Dinv[2][0] + Esfy*Dinv[2][1] + Esfxy*Dinv[2][2];
            Esf[j] = EsfD1*Esfx + EsfD2*Esfy + EsfD3*Esfxy;
            norma_esf = norma_esf + Esf[j];
        }
        Norma_Energia[i] = norma_esf*prop[i][1]*area;
        error2_ener = error2_ener + Norma_Energia[i];
    }
}

```

```

    }
}
error1_elem = sqrt(error1_elem);
error2_ener = sqrt(error2_ener);
for(i=0;i<NE;i++)
    if (elemtype[i]==0 || elemtype[i]==2)
        Param_Local[i] = (sqrt(Norma_Elem[i]))*(sqrt(NE-ncont))/(tolc*error2_ener);
printf("Error Esfuerzo e: %f\n",error1_elem);
printf("Error Energia U : %f\n",error2_ener);
printf("Error Total : %f\n",(error1_elem)/(tolc*error2_ener));
}
//*****
void PrintTime(char file[20])
{
    ofstream out;
    out.open(file,ios::app);
    out << "\n";
    out << "** TIEMPO DE EJECUCION = " << (time_fin - time_ini) << " segundos \n";
    out << "\n";
}
//*****
void PrintError(char file[20])
{
    int i;
    fstream mistream;
    mistream.open(file,ios::in|ios::app);
    mistream << "Error Esfuerzo e: " << error1_elem << "\n";
    mistream << "Error Energia U : " << error2_ener << "\n";
    mistream << "Error Total : " << (error1_elem)/(tolc*error2_ener) << "\n";
    for(i=0;i<(NE-ncont);i++)
    {
        mistream << "Param_Local\n";
        mistream << i+1 << ".\n";
        mistream << setw(10) << Param_Local[i] << "\n";
    }
}
void PrintResult(char file[20])
{
    int i,j;
    fstream mistream;
    mistream.open(file,ios::in|ios::app);
    mistream << "\n";
    mistream << "\n";
    mistream << "**** RESUMEN DE RESULTADOS DE DESPLAZAMIENTOS ***\n";
    for (i=0;i<NND;i++)
    {
        mistream << i+1 << "\n";
        mistream << setw(15) << coordini[i][0] << "\n";
        mistream << setw(15) << coordini[i][1] << "\n";
        mistream << setw(15) << coord[i][0] << "\n";
        mistream << setw(15) << coord[i][1] << "\n";
        for (j=0;j<NDOFN;j++)
            mistream << setw(15) << DESPLA[i*NDOFN+j] << "\n";
        mistream << "\n";
    }
    mistream << "\n";
    mistream << "**** RESUMEN DE RESULTADOS DE ESFUERZOS ***\n";
    mistream << "\n";
    for (i=0;i<NE;i++)
    {
        if (elemtype[i]==0 || elemtype[i]==2)
        {
            mistream << "Elem " << i+1 << ".\n";
            for(j=0;j<NNE;j++)
            {

```

```

        mistream << setw(15) << ESFUER[i][j] << "\t";
        ESF[i][j] = ESFUER[i][j];
    }
}
mistream << "\n";
}
}
//*****
void PrintData(char file[20])
{
    ofstream out;
    out.open(file,ios::app);
    out << nitepp << "\n";
    out << "Numero de Iteraciones Paso-Paso \n";
    out << "\n";
}
//*****
void PrintElas(char file[20], int iterc)
{
    int i,j,k;
    double sxx[3],syy[3],sxy[3];
    fstream mistream;
    mistream.open(file,ios::in|ios::app);
    cout << "**** Inicia calculo de Esfuerzos \n";
    mistream << " \n";
    mistream << " \n";
    mistream << "**** TABLA DE ELASTICIDAD EN CADA ELEMENTO ***\n";
    for (i=0;i<NE;i++)
    {
        if (elemtype[i]==0 || elemtype[i]==2)
        {
            mistream << "Elem " << i+1 << " :\n";
            mistream << setw(15) << ESFUER[i][0] << "\t";
            mistream << setw(15) << prop[i][0] << "\t";
        }
        mistream << "\n";
    }
    cout << "**** Fin de calculo de Elasticidad \n";
}
}

```