Tesis Doctoral

Análisis Dinámico de las Tenso Estructuras:
Propuesta de
Metodología de Cálculo y Software Aplicado.

A Irene y Patricia.

Agradecimientos:

A mi padre Julio Muñiz Padilla. Sin su infinita dedicación y ayuda el software que desarrolla e ilustra el método de cálculo hubiera sido imposible.

A Ramón Sastre. Además de sus enseñanzas me ha brindado su confianza y desinteresada generosidad. Suyo es el entorno y apoyo imprescindible para el desarrollo de esta tesis.

Ernesto Muñiz Martín

Septiembre de 2014.

Análisis Dinámico de las Tenso Estructuras: Propuesta de Metodología de Cálculo y Software Aplicado.

Doctorando: D. Ernesto Muñiz Martín
Universidad Politécnica de Cataluña.
Departamento de Arquitectura y Tecnología de la Edificación.

Sinopsis

El análisis dinámico de los sistemas de múltiples grados de libertad se plantea tradicionalmente estudiando sus Modos Propios de oscilación o mediante los balances de energía que se establecen sobre formas de oscilación supuestas. A partir de dichas técnicas el analista determinará la sensibilidad a la resonancia del sistema ante acciónes externas dinámicas (viento, circulación de vehículos, paso humano, etc…), o acotará el potencial comportamiento del sistema en base a los espectros de respuesta (sismo).

Esta Tesis Doctoral propone una metodología de cálculo que permitirá obtener las diferentes formas modales y balances energéticos asociados a la dinámica de una malla tesa, de manera que para una tenso estructura idealizada en barras se determinarán sus frecuencias fundamentales, las formas características de los modos propios asociados a las diferentes frecuencias, así como la composición de la oscilación en “X, Y, Z”. Con dicha información el analista tendrá la base con la cual interpretar el comportamiento dinámico potencialmente resultante en una malla tesa en las situaciones ingenierilmente más comunes.

.

La propuesta se complementa con un software informático que materializa la aplicación del método.

Términos Clave: Tenso estructura, Análisis Dinámico, Análisis Modal espectral, Modos Propios, Métodos Energéticos, Rayleigh Ritz, Densidad de Fuerza, Form Finding.

Tutor y Director: Dr. Ramón Sastre i Sastre. Departamento de Arquitectura y Tecnología de la Edificación. Universidad Politécnica de Cataluña

Relación de contenidos. Índice.

Listado de Figuras

Listado de Tablas

Glosario. Notación empleada

Reglas Generales:

De forma general se denomina a un vector entre paréntesis (V) –vector “V”- y a las matrices entre corchetes [M] –Matriz “M”-

Toda variable inmediatamente acompañada de un número (sin un signo algebraico mediante) significa su potencia (W2 → “W al cuadrado”), si bien en algunos casos análogamente las potencias se expresan de la siguiente forma:

W^2=W2

Igualmente cuando una variable tiene barias componentes o puede hacer alusión a un elemento de un conjunto suele emplearse el subíndice o la combinación de estos separados mediante comas, por ejemplo:

(L)= vector de longitudes
Li = Longitud barra “i”
(L)x,y,z = Longitudes en X,Y o Z
[L]x,y,z = Matriz de longitudes X,Y,Z
Li,x = Longitud de la barra “i” en X

Variables

Establecidas estas simples reglas procedemos a describir la notación empleada:

A: Suele reservarse para definir la Amplitud (metros) de la oscilación según se defina esta en un movimiento ondulatorio del tipo:

X= A*seno(W*t)

A_c: Aceleración (m/s2)

D: Se emplea para la Densidad de Fuerza (Newton/Metro) de una barra según el método propuesto por Schek y Linkwitz:

D=Fuerza/Longitud

D_i: Densidad de Fuerza de la barra “i” (Newton/metro)

[D]: Matriz de densidades de fuerza (Newton/metro)
DDF: Densidad dinámica de fuerza. Equivale a la densidad de fuerza pero reflejándose las propiedades de sección, material y longitud de la barra que constituye la estructura (Newton/Metro).

Ec,Ep: Energía cinética y potencial (julios)

[FM]: Matriz de Formas Modales (factor multiplicador de metros), cada columna es un vector que representa una forma modal asociada a un autovalor, por ejemplo la tercera columna sería el tercer autovector que correspondería al tercer autovalor.
(Fr): Vector de Autovalores (velocidad angular al cuadrado) de la resolución del análisis modal.
Fm: Factor de masas (factor multiplicador de masas)
Fx,Fy.Fz: Fuerza en los respectivos ejes X,Y,Z (Newton)
[F]x,y,z : Matriz de fuerzas (Newton)
Finer,Felas,Fdisi: Fuerzas de Inercia, elásticas y disipativas (Newton)

[G] : Matriz de vínculos entre nudos “matriz de relaciones”
[Gt]: Matriz de relaciones transpuesta
[+Gt]: Matriz de relaciones transpuesta con todos sus elementos “positivados” (los -1 se convienten a 1)

(incL): vector de incrementos de longitudes (metros)

K: Rigidez de una barra según la Ley de Hooke (Newton/Metro)

Fuerza = K * Desplazamiento

L : Longitud de una barra (metros)
Li: Longitud de la barra “i”
Lx,y,z: Proyecciones de la longitud de la barra en X,Y,Z
(L): Vector de Longitudes de las barras

M: Masa de un cuerpo (Kilogramos)
[M]: Matriz de masas de un sistema (Kilogramos)

[P] = Matriz fruto del producto [Gt]*[D]*[G] que relaciona fuerzas y desplazamientos (Nemton/metro) según el método de la densidad de fuerza:

$$[P]*(X)=(Fx)$$

[Pdin] = Reducción de la matriz [P] a la parte que atañe sólo a los nudos móviles del sistema (Newton/Metro)

[Pdin]$_{DDF}$= Matriz de rigidez dinámica, obtenida domo la matriz [P] pero haciéndose uso de las Densidad Dinámica de Fuerza (Newton/metro).

T: Periodo se un movimiento oscilatorio según la relación

$$W = 2*Pi/T$$

t: Se reserva para la variable tiempo.

U(t): Posición en el tiempo (metros)

V_e(t): Velocidad en función del tiempo (metros/Segundo)

W: Se reserva para la velocidad angular. (Radianes/Segundo)

Wint, Wext: Trabajo de las fuerzas internas y externas (julios)

X,Y,Z : Ejes cartesianos según su notación habitual.

(X),(Y),(Z): Vectores de posición en los ejes X,Y,Z (metros)

General.

Introducción

Los planteamientos para afrontar el análisis dinámico de las tenso estructuras recogidos en la escasa bibliografía específica son ciertamente limitados. Así hasta la fecha en el mejor de los casos las metodologías se basan en reglas de diseño o en planteamientos analíticos altamente dependientes de la pericia y experiencia del autor del modelo, raramente compartidos por estos autores.

Las estructuras de este tipo tienden hacia grandes deformabilidades, por lo que es intuitivamente apreciable la sensibilidad que estas estructuras presentarán ante fenómenos dinámicos como los flameos por viento o acciones móviles propias del uso humano al ser su rigidez relativa baja. Las reglas y códigos de diseño recogidas en la bibliografía general habitualmente se quedan en recomendaciones de proyecto basadas en el mejor o peor comportamiento que ha tenido el catalogo de obras ejecutadas (en función de su curvatura, distancia máxima entre apoyos…), o bien en el análisis de modelos de geometría sencilla o conocida cuyas formas de oscilación han sido estudiadas (hay cierta experiencia en el análisis de puentes y pasarelas colgantes). El no tener un método sistemático que se adentre en el comportamiento dinámico de estas singulares estructuras limita la audacia que los proyectistas pueden asumir, al ser la incertidumbre asociada al comportamiento dinámico muy alta y estando las pautas de estudio más avanzadas reservadas por unos pocos autores.

Ante esta necesidad se propone una metodología de cálculo “abierta” que pretende acercar el análisis del comportamiento dinámico de estas estructuras de forma clara, que persigue ser aplicable al común de casos, y que al ser sistematizable podrá desarrollarse en forma de Software, lo que constituye uno de los principales objetivos de esta tesis.

Entendemos por **tenso estructura** aquel sistema de elementos vinculados entre si capaz de alcanzar una posición de equilibro mediante el exclusivo desarrollo de esfuerzos axiales a lo largo de sus elementos gracias a la adaptación de su forma.

Figura 1:Iilustración tomada de "Recostruyendo Cuenca" de Alberto Ballesteros Baea

La expresión más intuitiva de una tenso estructura es la catenaria, que manifiesta de forma clara la relación biunívoca entre equilibrio y forma, siendo su expresión tridimensional más simple la malla tesa, representable como un conjunto de catenarias entrelazados entre si. Una red de circo sería un buen ejemplo de malla tesa, que mediante la adaptación de su forma es capaz de asumir la caída de un trapecista.

Figura 2: Red para los trapecistas del "gran circo mundial"

No debemos confundir las tenso estructuras con las estructuras de barras articuladas tipo celosía o cercha pese a que también éstas logran el equilibrio mediante el exclusivo equilibrio de esfuerzos axiales en sus casos más sencillos.

Figura 3: Ejemplo de celosía típica: un puente de ferocarril

La geometría de las celosías es “rígida”, siendo la forma una necesaria condición de partida, mientras que en las tenso estructuras la forma es una consecuencia del equilibrio alcanzado:

Figura 4: Ejemplo de tenso estructura: Estadio olímpico de Munich (Frei Otto)

En las estructuras convencionales *para una geometría dada* se establece la imposición del equilibrio y de compatibilidad de deformaciones en cada uno de sus grados de libertad, obteniéndose un sistema de ecuaciones resoluble linealmente. En los casos más sencillos basta con la imposición del equilibrio en cada uno de sus nudos:

Barras + Coacciones = 2 * número de Nudos

Afirmándose en este caso que es un sistema isostático, cumpliéndose literalmente dicha condición. Un mecanismo sería aquel caso en el que hay más del doble de nudos que la suma de barras y coacciones, siendo un sistema inestable al haber más incógnitas que ecuaciones. Por último una estructura sería hiperestática cuando la suma de coacciones y barras supere al doble del número de nudos, resultando un sistema con más ecuaciones que incógnitas, por lo que para la determinación de las

solicitaciones internas debería recurrirse a la compatibilidad entre las deformaciones elásticas de los elementos constituyentes, lo que introduce las necesarias incógnitas adicionales.

Así pues la resolución de las estructuras convencionales suele enfocarse planteando un sistema lineal de ecuaciones agrupable según la siguiente expresión matricial:

(Vector de fuerzas) = [Matriz de rigidez] x (vector de desplazamientos)

Donde el vector de fuerzas agrupa las acciones aplicadas al sistema, la matriz de rigidez establece el vínculo entre los distintos elementos constituyentes en base a sus propiedades (geometría del sistema y propiedades de las barras) y el vector de desplazamientos agrupa el habitualmente "*pequeño movimiento diferencial*" de los nudos, necesario para determinar la compatibilidad entre elementos dentro de la geometría establecida. Este sistema impone a la vez equilibrio y compatibilidad para una geometría establecida.

El análisis dinámico de las estructuras convencionales está notablemente desarrollado e implementado en las metodologías habituales de cálculo. Dicho análisis se basa en el equilibrio en el tiempo entre las fuerzas de inercia del sistema en movimiento (asociadas a la masa del sistema -*fuerza es igual a masa por aceleración*-) frente a la oposición que a dichas fuerzas plantea la rigidez del sistema (asociadas a la matriz de rigidez –*dependiente de la geometría y propiedades de las barras*-) al alejarse el sistema de su posición de partida durante el movimiento.

[Matriz de Masas](vector de aceleración)* + [Matriz de rigidez] x (vector de desplazamientos) = *0*

Las tenso estructuras sin embargo son conceptualmente diferentes dado que por definición adaptan su forma original hasta que el equilibrio sea posible: "primero es el equilibrio y de ahí proviene la forma". Por tanto el planteamiento del sistema de fuerzas será de partida diferente al de las estructuras convencionales al no ser la forma a priori conocida, es decir *La forma es una consecuencia*. Intuitivamente se percibe que la respuesta de una malla tesa ante la variación de las condiciones establecidas por un equilibrio inicial (cambio de condiciones de carga, etc…) conllevará "grandes" desplazamientos, mucho mayores que el *"pequeño moviendo diferencial"* propio de las estructuras convencionales.

La adaptación de la forma ante un cambio de condiciones dependerá de cuánto se aleje la nueva circunstancia de equilibrio respecto de la circunstancia inicial de partida en *términos relativos*. Por tanto un cambio de forma será pequeño si la variación de solicitación asociada al cambio de condiciones es poco representativa en relación a la solicitación asociada a las condiciones de partida.

Si entendemos sintéticamente que la rigidez de un sistema se puede analizar evaluando la relación entre las acciones y los desplazamientos asociados a dichas acciones, claramente vemos que en las tenso estructuras para un mismo sistema y unas mismas acciones la respuesta variará *en función del nivel de solicitaciones inicial de dicho sistema*, es decir en base al "pretensado" que el sistema tenga antes de las acciones. Un sistema que parta de un "pretensado" mayor tendrá menores movimientos que un sistema idéntico que parta de un "pretensado" menor.
Por tanto la rigidez del sistema es implícita a su estado tensional, no dependiendo sólo de la masas y propiedades de los elementos.

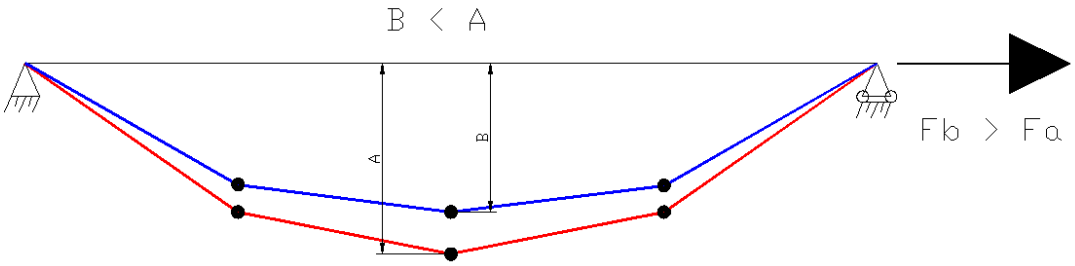


Figura 5: Ejemplo de variación de la forma con la tensión

El análisis dinámico de las tenso estructuras deberá atender a las particulares circunstancias del caso, en la que la forma de equilibrio del sistema responde a la propia acción dependientemente del estado tensional de partida. El equilibrio en el tiempo entre las fuerzas de inercia y las fuerzas derivadas de la respuesta elástica del sistema ante el movimiento habrán de plantearse atendiendo a estas singularidades que las distinguen de las estructuras convencionales, requiriéndose un desarrollo y formulación específicos. Proponer un método de análisis sencillo y programable es el objeto de esta tesis.

Objeto

El objeto de esta tesis doctoral es formular una propuesta de metodología de cálculo que permita obtener las diferentes formas modales de una malla tesa, de manera que se determine para una tenso estructura bajo unas condiciones de partida sus frecuencias fundamentales, la forma característica de los modos propios asociados a las diferentes frecuencias así como la composición de la oscilación en “X, Y, Z” a efectos de poderse caracterizar el movimiento resultante.

Las formas modales de una estructura son la base tradicional del análisis dinámico de las estructuras, tanto a efectos de analizar la sensibilidad a la resonancia ante una acción externa (viento, circulación de vehículos, paso humano, etc…), como para el estudio del comportamiento (sismo) en base a los espectros de respuesta.

Dentro del contenido de la tesis y como manifiesto del carácter sistemático de la propuesta se elabora un software informático para el desarrollo aplicado del método. Se pretende que sea la herramienta de manejo que permita abordar el común de casos de análisis.

Desamos destacar que no es objeto de esta tesis el análisis de membranas. La conversión de superficies continuas a sistemas discretizados en forma de mallas queda fuera de los objetivos de la presente tesis.

Antecedentes y estado del Arte

Los antecedentes empleados en esta Tesis agrupadamente por autores y materias son:

Conceptos generales de las Tenso Estructuras: Se podría considerar que Frei Otto plantea los conceptos estructurales básicos según su actual concepción. La actualización de dichas ideas se obtiene al cursar el “Máster en Arquitectura Textil” (UPM-Structuralia del que ha sido profesor el tutor de la presente Tesis, Ramón

Sastre) a la par que gracias la extensa bibliografía existente en la materia a la que se ha tenido acceso.

Form Finding: Linkwitz y Schek establecen en 1971 el “Método de la Densidad de Fuerza” (Force Dansity Method), punto de partida para establecer una relación entre forma, desplazamiento y esfuerzos en una tenso estructura. En esta tesis la aplicación práctica de dicha metodología se ha extraído del artículo de investigación “Topological Mapping for Tension Structures”. (E. Hernandez-Montes, R. Jurado-Piña, E. Bayo)

Análisis Dinámico Clásico: En las publiaciones de D.J. Inman, Anil K. Chopra , Amadeo Benavent-Climent se encuentran extensamente desarrollados los conceptos utilizados asociados al análisis dinámico de estructuras. Durante el programa de Doctorado en Análisis Dinámico e Ingeniería Sísmica (UNED) se imparten las esencias del análisis dinámico bajo la tutela del Dr. Enrique Alarcón, quien personalmente profundiza en los conceptos del análisis por Métodos Energéticos (Método de Rayleigh Ritz) tan necesarios en esta Tesis.

Análisis Dinámico de las tenso estructuras: Massimo Majowiecki Proporciona un método de análisis dinámico para las tenso estructuras basado en la superposición modal de formas de oscilación canónicas que el calculista deberá saber obtener como punto de partida del análisis. Leonard por su parte […] Igualmente en las publicaciones de Tensinet encontramos reglas simples de diseño encaminadas a evitar problemas dinámicos establecidas como “pautas de buen hacer”. Autores como Niels J. Gimsing Christor T. Georgakis, etc… exponen la metodología que fundamenta al análisis dinámico de puentes colgantes y de cables, problemática que guarda relación con el tema analizado. Igualmente cabe mencionar autores como Michael Seidel en cuya bibliografía se recogen métodos sónicos para la verificación de los estados tensionales de las tenso estructuras.

Respecto del estado del arte la ausencia de programas de cálculo orientados al análisis dinámico de las tenso estructuras resume la escasez de metodologías generales de análisis que sean sistemáticas y poco dependientes de la experiencia del autor del cálculo.

Podría considerarse el análisis dinámico de las estructuras colgantes –forma básica de tenso estructura- como el primer paso cara al abordaje del fenómeno analizado. Comúnmente estos casos elementales se estudian a partir del conocido problema de oscilación de una catenaria:

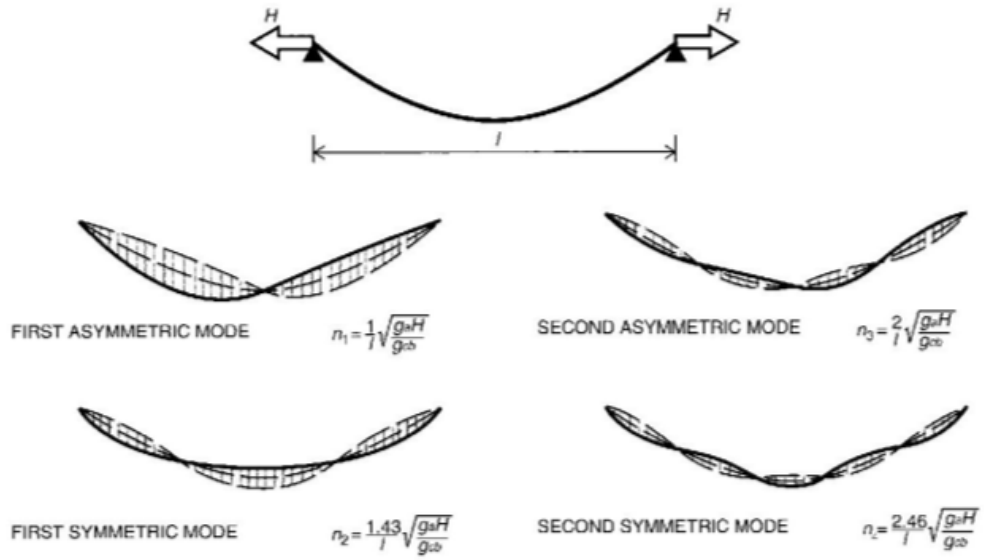


Figure 2.102 *Modes of in-plane vibration for the sagging cable*

Figura 6: Formas de vibración de una catenaria

O de una cuerda tensa:

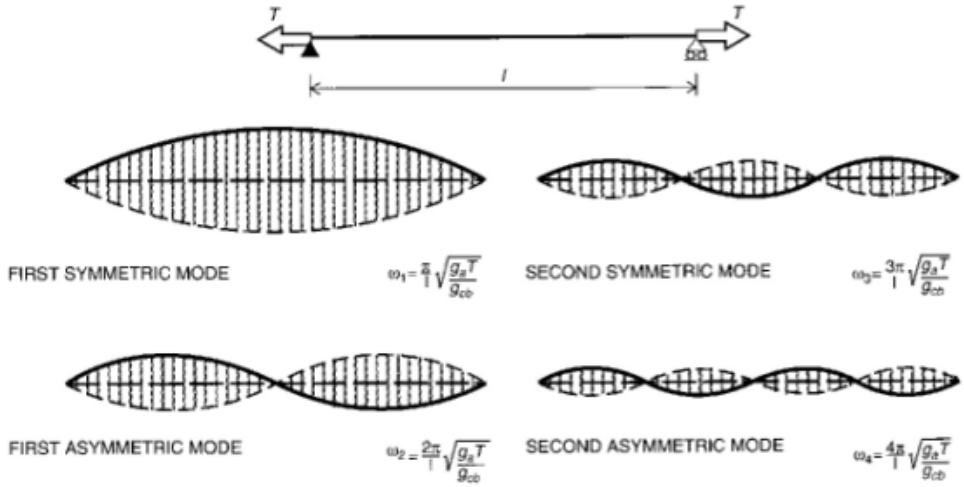


Figure 2.100 *Modes of vibration for the taut string*

Figura 7: formas de vibración de una cuerda

Fuente: Cable Supported Bridges. Niels J. Gimsing Christor T. Georgakis (2012 JohnWily & Sons)

Ga= gravedad, Gcb=peso por unidad de longitud

La generalización del comportamiento de una catenaria o cuerda a una multitud de ellas entrelazadas sería sólo orientativa a efectos de hacernos una idea muy básica y limitada del comportamiento que tendrá una tenso estructura. Como vemos en las figuras anteriores la elasticidad del material constituyente NO es un parámetro que aparezca en la formulación que define la frecuencia natural de oscilación. Ello nos

hace percatarnos de lo alejada que está esta interpretación del comportamiento general de los casos complejos en los que el movimiento está vinculado a varios elementos que habrán de compatibilizarse elástica y geométricamente entre si.

El siguiente paso sería analizar los códigos de diseño en la materia. Al respecto destacamos el apartado 7.4.6 de la “Guia Europea de Diseño de las Estructuras Superficiales Tensadas” (Brian Foster Marijke Mollaert. 2004, Tensinet), donde se dan reglas de diseño sancionadas por la experiencia. Al respecto recomiendan bordes de religa libre que no superen los 20 metros y se establece una relación matemática denominada Rigidez Superficial (Kn/m2) que se recomienda sea mayor a 0.3 y que no baje de 0.15. Se comenta que valores por debajo de 0.2 requieren de estudio especial sin profundizarse más.

D= (Pretensión Urdimbre)/(Radio de Curvatura Urdimbre) + (Pretensión Trama)/(Radio Curvatura Trama)>0.3

Como paso siguiente entramos ya en lo más avanzado en el Estado del Arte, destacando la metodología propuesta por Massimo Majowiecki en su obra “Tensostrutture Projetto e Verifica” y la publicación “Tensión Structures, Behavior & Analysys” de John William Leonard.

Majowiecki plantea el uso de la superposición modal a partir de unas formas canónicas de oscilación, lo cual guarda gran relación con la propuesta de esta tesis. En dicha obra -cláramente orientada al análisis de la respuesta de las tensos estructuras ante fenómenos aerodinámicos- (véase específicamente capítulo 6.8 de dicha obra), Majowiecki repasa las bases esenciales del análisis dinámico y describe una metodología generalista basada en la aplicación de series de Fourier, técnica que conducirá igualmente al planteamiento de un problema de autovalores, coincidiendo con nuestra metodología. Sin embargo dicha obra no va más allá, limitándose sólo a exponer los conceptos de análisis dinámico necesarios para iniciar el estudio dinámico de las tenso estructuras.

Leonard en su obra hace una estudio técnico muy interesante sobre el análisis numérico de las membranas, tratando su comportamiento dinámico en el capítulo 4 de su citada obra. Su enfoque del tema dinámico es análogamente “clásico” y conceptualmente similar al expuesto por Majowiecki, si bien en este caso la exposición de Leonard es más detallada, abordándose con detenimiento casos sencillos de pocos grados de libertad y haciendo hincapié sobre los fenómenos no lineales que rigen el fenómeno (deflexión local de la catenaria por su peso propio, etc..). Hay que destacar

la exposición que hace sobre la aplicabilidad de los teoremas energéticos para el caso de las los sistemas de cables -punto 4.4.1-, idea que se expone aplicando el principio de Hamilton de los trabajos virtuales: la variación de energía cinética y potencial elástica en un ciclo son equivalentes -siempre que se desprecien los fenómenos disipativos-. Respecto a como plantear la rigidez y masa del sistema el autor se centra en los elementos finitos tipo “cable” (capítulo 3 en lo relativo a rigidez y análisis estático, punto 4.4.2 en lo concerniente a la masa), obteniéndose una matriz fruto del ensamblaje de los elementos y de un proceso iterativo de cálculo.

La presente tesis plantea la resolución de la ecuación que rige el comportamiento de una forma alternativa a las referencias citadas. La esencial variación subyace en cómo se plantean las matrices de masas y de rigidez partiendo del método de la Densidad de Fuerza, de lo que resulta un desarrollo alternativo en relación con las referencias anteriores. En todo caso se propone una visión alejada de los planteamientos elementales de estructuras, pues según estos las tenso estructuras serían mecanismos. Integradamente a la resolución se introducen los postulados energéticos como verificación, se hace un análisis de los límites de validez del movimiento y se estudia la composición tridimensional del movimiento resultante en X, Y, Z.

Aporte de la Propuesta al Estado del Arte

La presenta Tesis aspira a contribuir al análisis de las mallas tesas mediante el desarrollo de los aspectos que se exponen a continuación:

a) Proponiendo una metodología sistemática para determinarción de los modos propios de vibración de las tenso estructuras de barras. El enfoque es alternativo a los planteamientos tradicionales, combinando los fundamentos del método de la Densidad de Fuerza, los principios tradicionales del análisis dinámico para múltiples grados de libertad y los teoremas energéticos aplicados al cálculo dinámico.
b) Formalizando la investigación realizada en un software que brinda una gran libertad de análisis. No se tiene constancia de la existencia de un software similar al alcance del común de analistas.
c) Se establece un planteamiento singular para la resolución del método de la Densidad de Fuerza, lográndose su cálculo matricial de una forma aljebraica mediante

lo que se ha denominado "condensación de apoyos", lo que permite la resolución matricialmente directa de la forma.
d) Se plantea una matriz de rigidez para los nudos móviles del sistema a partir de la citada "condensación de apoyos", lo que permite caracterizar la estructura una vez es dimensionada asociadamente a su estado tensional -implícito a la forma de equilibrio- y a las propiedades de las barras. Adicionalmente se plasman las correcciones a la rigidez asociadas a la elasticidad tangente de los tirantes.
e) Se propone una metodología para la obtención mecánica de la matriz de masas coherentemente con la expresión de rigidez deducida. Para ello se emplean matrices desarrolladas desde la adaptación del método de la densidad de fuerza, lo que permite mecanizar todo el proceso de forma compacta.
f) Planteadas las expresiones que caracterizan la rigidez y masa del sistema se resuleve el problema de autovalores y autovectores, analizándose las relaciones que vinculan la tridimensionalidad del movimiento con la unicidad de masa y rigidez. La sincronía del movimiento en X,Y,Z dentro de un rango de posibles amplitudes en cada dirección conduce a la interpretación de la respuesta asociadamente a un espectro o rango de frecuencias, lo cual es un contrapunto con las estructuras convencionales, donde el valor tiene a ser determinista.
g) Se mecaniza una verificación energética del proceso mediante la aplicación del principio de Hamilton, cotejándose éste proceso con el estudio de ciclos completos sobre casos singulares en los que se ha aplica el teorema de Railegh Ritz.
h) Se estudia la correlación entre los resultados obtenidos aplicando el método de los autovalores y las frecuencias obtenidas por los métodos energéticos, puesto que se constatan singularidades en algunos casos. Del análisis de dichas paradojas se obtienen unas conclusiones sobre la confiabilidad del análisis, lanzándose unas recomendaciones a modo de "límites de validez" a partir de las cuales el analista debe ser consciente de los fenómenos que gobiernan la dinámica del sistema, dado que cuando el movimiento tiende a desvincularse del balance de energía interna del sistema (por darse una gran adaptación de forma sin casi haber variación de elongación de los elementos) se debe cuidar en la interpretación del fenómeno.

Además se postulan líneas de investigación complementarias.

Limitaciones y ámbito de aplicación

La propuesta presenta unas limitaciones que el analista deberá conocer y asimilar. Así los resultados deberán estar sometidos a un proceso de interpretación puesto que los valores numéricos y las formas obtenidas "literalmente aplicados" podrán no coincidir con la realidad del fenómeno al 100%. Pese a ello se entiende que el método goza de un gran ámbito de aplicación y utilidad, puesto que permite visualizar de una forma cuantificada y clara la tendencia del fenómeno, el rango de valores en los que se produce el movimiento y la sensibilidad de la respuesta a respecto a la variación de los diferentes parámetros fundamentales (masa, elasticidad, tensión, etc…) que caracterizan el modelo.

Como es sabido el técnico especializado en el análisis dinámico está habituado a que las estructuras reales sean simplificadas hacia modelos esquemáticos dotados de una linealidad y simpleza utópica. De su análisis "idealizado" se obtendrán respuestas teóricas, que manifiestarán de una forma cuantificada y clara la tendencia del fenómeno. La inexacta realidad requerirá de la prudente interpretación de los hipotéticos resultados, gracias a lo cual el especialista acotará la respuesta. Para ello cuando se estudian las estructuras convencionales se recurre al empleo de espectros dúctiles de respuesta congruentes con los diferentes grados de ductilidad y daño asociados al premeditado diseño, así como a una matemática asociada al riesgo que establecerá coeficientes de seguridad a acciones y materiales. Aún así la respuesta será imprecisa.

Esta tesis versa sobre la parte utópica del fenómeno dinámico de las mallas tesas, por lo que sobra decir que el especialista habrá de ser singularmente prudente. Las correlaciones que vinculen las tendencias teóricas simplificadas con respuestas concretas de una estructura real aún habrán de ser desarrolladas, sugiriéndose dicha labor entra las líneas de investigación propuestas.

Hay que destacar que el análisis dinámico de precisión de las estructuras convencionales normalmente requiere de lo que se denomina un "*performance based desing*". En esos singulares casos el planteamiento "típico" pasa por hacer uso de un análisis "push over" que reproduzca el comportamiento "time history" de un modelo meticulosamente desarrollado, en el que además de un refinado análisis de las masas y componentes del sistema, se introducirá la no linealidad de las diferentes partes

resistentes (a nivel de secciónes críticas se implementan los diagramas de respuesta que modelizan la plastificación y fisuración). Es excepcional hacer este tipo de modelado, siendo sólo justificable para casos en los que es esencial que la distorsión esté muy acotada (telescopios en zonas sísmicas, centrales nucleares, etc…). Por supuesto ello también sería realizable sobre una tenso estructura, si bien la labor y experiencia requerida será igualmente singular.

Por tanto esta Tesis Doctoral aporta una técnica que permite cuantificar la tendencia teórica del movimiento de los casos más generales, estableciendo una forma de oscilación concreta vinculada a una frecuencia determinada (forma modal). Para ello realiza una serie de simplificaciones que “idealizan” el modelo y que los antecedentes ya citados (Majowiecki, Leonard) comparten con esta tesis:

- No se atiende a la variación del módulo de elasticidad tangente/secante que se producirá en las barras durante el ciclo de oscilación (aunque SI se considera en esta tesis el fenómeno a la hora de evaluar la elasticidad real del sistema): Como se sabe dichos módulos de elasticidad están intrínsecamente asociados a la tensión de la barra para una posición y propiedades de la misma establecidas. La elasticidad del elemento variará de forma no lineal según sea la parte del ciclo a tracción o a compresión y ello perturbará la respuesta real respecto de la estimada, si bien ello será poco apreciable al inicio del fenómeno de oscilación.
- La amplitud de oscilación de las estructuras reales normalmente tenderá a dejar sin tensión algunas de las barras en aquella parte del ciclo que se acorten respecto de su posición de equilibrio. Esta situación no es modelizada, las barras “no desaparecen” ni se comprimen, por lo que realmente el modelo será congruente para amplitudes de oscilación muy pequeñas (dicho valor de amplitud máxima lo calcula el programa). En la realidad se tenderá a producir una disminución de la rigidez del sistema respecto de la predicha por el programa (puesto que por el contrario el aumento de tensión no tiende a producir un significativo de la rigidez a nivel barra) y por ende una desviación a la baja de las frecuencias respecto de las modales, dado que para oscilaciones altas siempre en alguna parte del ciclo habrá barras “flojas” por lo que la rigidez de conjunto tenderá a ser menor que la teórica.
- No se modeliza la plastificación de los materiales, si bien "como tal" entendemos que su consideración en rigor tendría una importancia bastante limitada respecto al análisis dinámico. (Si es fundamental en cambio para el

dimensionado de las membranas) Una oscilación tradicional (simétrica) implica un recorrido de “ida y vuelta" en la deformación, por lo que entendiendo como “ida” la parte asociada a la “carrera de estiramiento" en la “vuelta” dicha magnitud será recuperada e incluso DESTRACCIONADA. Como se sabe la histéresis de un material asociada a su plastificación habitualmente tiene una recuperación más o menos equivalente a su estiramiento elástico "noval" por lo que la parte del recorrido asociada a la plastificación inicial (que aprovecha el material) importa relativamente poco. Lo que realmente interesa es la parte elástica del proceso. La parte no elástica se traducirá en los fenómenos anteriormente tratados.

En el análisis dinámico lo determinante es generalmente evaluar la sensibilidad de la respuesta de la estructura ante las posibles frecuencias de excitación, dado que se pretende normalmente predecir si es posible que se produzca una respuesta “fuera de control” ante una acción cíclica que normalmente estará asociada a un rango de frecuencias según la naturaleza de la acción (viento, tráfico, humanos, máquinas…). Se analiza pues la posibilidad de que se dé un acople entre acción y reacción. El conjunto de fenómenos que se han simplificado son poco relevantes en el inicio del fenómeno dinámico, donde las variaciones de tensión son leves. Además dichos fenómenos tenderían precisamente a variar la frecuencia a la baja respecto de la propia inicial (puesto que normalmente la tendencia es disminuir la rigidez en la parte del ciclo en el que la barra se acorta respecto de su posición de equilibrio inicial). Esa imprecisión aún siendo indeseable es manejable, puesto que toda tendencia de comportamiento que tienda a alejar la respuesta respecto de su frecuencia nicial será “protectora” de los fenómenos de resonancia, puesto que el fenómeno siempre comienza por un movimiento leve. Dicho de otra forma, si lo que no comienza no sucede, lo que no sucede en el modelo ideal difícilmente ocurrirá en el modelo real.

En cualquier caso como se pretenderá mostrar la respuesta modal de las tenso estructuras está fuertemente condicionada por la dimensionalidad del movimiento en relación con la dimensionalidad de la rigidez. Esta afirmación tan abstracta se traduce en que la respuesta de frecuencias asociada a una forma modal no se corresponde a un valor singular sino a un *rango* de frecuencias.

Por último y aún habiéndose mencionado en la parte de “Objeto” se hace hincapié en que no es objeto específico de esta tesis el análisis dinámico de superficies continuas (membranas). Es a veces común la “identificación” de un sistema “trenzado de barras”

(malla tesa) con una superficie continua (membranas), dado que incluso hay programas que “automatizan” dicha conversión asignando a las barras determinadas propiedades en función de la geometría del sistema, los estados tensionales y los materiales constituyentes. Dicha conversión queda fuera de los objetivos de esta Tesis en el presente grado de desarrollo.

Realizada la citada conversión a barras el método determinaría los resultados dinámicos conforme las limitaciones descritas, si bien se vuelve a insistir respecto a que establecer dicha conversión e interpretar la sensibilidad del modelo será cosa del autor de la simplificación.

Principios y desarrollo teórico.

Fundamentos

La metodología propuesta se basa en los siguientes fundamentos del cálculo de estructuras.

- Método de la Densidad de Fuerza.
- Dinámica clásica de sistemas estructurales.
- Análisis Modal Espectral.
- Planteamiento Energético. Rayleigh Ritz.

A continuación se intentarán sintetizar las ideas esenciales necesarias para la compresión conceptual de la metodología de cálculo propuesta en esta tesis doctoral.

Método de la densidad de fuerza.

Método de cálculo que permite deducir la de forma de equilibrio de una tenso estructura establecidas unas condiciones de contorno y acciones que actúan sobre una malla tesa de propiedades conocidas. Constituye la “esencia” numérica del análisis de las tenso estructuras, entendiéndolas como un entramado de barras de forma libre que trabajan axialmente. Previamente a la introducción de este método por Schek el análisis de las tenso estructuras se enfocaba mediante la creación de maquetas (de elásticos, tela de medias), pompas de jabón, deducción gráfica de formas….

Sus principios básicos son:

- En cada nudo en el que convergen barras se impondrá el equilibrio, compensándose entre si tanto las acciones transmitidas desde las barras como las actuantes directamente sobre el propio nudo.
- La forma es “libre” y a priori desconocida, siendo ésta consecuencia de la búsqueda de equilibrio. Lo único conocido son las acciones sobre los nudos, las coacciones en los apoyos y los vínculos que relacionan los nudos entre si (las barras).

- Cada barra enlazará dos nudos entre si caracterizándose exclusivamente por su *densidad de fuerza* “D”. Por tanto D es propia de cada barra, constante y se define como la relación entre la Fuerza actuante sobre la barra y su Longitud (D=F/L –constante-)

Así para una barra determinada “i” según la definición de Densidad de Fuerza “D” la acción en la barra será:

$$D_i = \frac{F_i}{L_i} \rightarrow F_i = D_i \times L_i$$

(df.1)

Si una barra se entendiera como un “chicle ideal” una mayor longitud de barra significará que proporcionalmente actúa una mayor fuerza sobre la misma. Si quisiéramos conocer las fuerzas de una barra por proyecciones “X, Y,Z”, la proyección sobre el eje “X” -por ejemplo- se definiría como:

$$F_i \times \frac{\Delta x_i}{Li}$$

(df.2)

(donde el incremento de X del numerador es la variación de coordenadas entre el fin e inicio de la barra)

Si sustituimos (1) en (2) tendríamos que para una proyección determinada (en este caso X) la fuerza descompuesta sobre dicho eje sería:

$$F_{i,x} = D_i \times L_i \times \frac{\Delta x_i}{L_i} = D_i \times \Delta x_i$$

(df.3)

Es decir que directamente la proyección del esfuerzo de una barra será el producto de la Densidad de Fuerza de dicha barra por la longitud proyectada de la misma: “$D*(X_{nudo\_final} – X_{nudo\_inicial})$”.

Ello permite plantear directamente un sistema de ecuaciones estableciendo el equilibrio nudo a nudo separadamente por componentes “X,Y,Z”, lo que permite deducir para unas acciones dadas las coordenadas “X, Y, Z” de los nudos. Al ser las incógnitas las coordenadas de los nudos y al haber tantas ecuaciones como nudos se tratará de un sistema resoluble y lineal para cada componente por separado.

La forma de equilibrio resultante dependerá de las acciones y coacciones -serían las condiciones de contorno del problema-, de cómo se conciben los vínculos entre los nudos -sería la geometría “básica” de la tensoestructura- y de la Densidad de Fuerza -

parámetro que representa las propiedades los materiales y el grado de pretensado asignado-.

La Densidad de Fuerza tiene un gran significado conceptual pues es el parámetro normalmente usado para lograr un “cierto ajuste” de la forma para una geometría de barras vínculadas entre si en equilibrio. La forma es una consecuencia del equilibrio, pero dentro de esta situación el sistema es en cierta magnitud moldeable “rigidizando” más o menos las barras entre si, lo cual se logra maniplando las densidades de fuerza de cada barra. Es un ajuste de forma siempre gobernado por el equilibrio y por tanto restringido.

Dinámica clásica de sistemas estructurales.

La caracterización dinámica de un sistema se plantea estableciendo el equilibrio en el tiempo entre las:

- Fuerzas de inercia (F_{iner}): causada por la aceleración de las masas móviles.
- Fuerzas disipativas (F_{disi}): debidas al amortiguamiento de un sistema que se desplaza a cierta velocidad.
- Fuerzas elásticas (F_{elas}): derivadas del desplazamiento de un sistema respecto de su posición de equilibrio estático.

$$F_{iner}+F_{disi}+F_{elas}=0 \quad \textbf{(dc.1)}$$

Al ser el objeto de análisis la oscilación del sistema entorno a su posición de equilibrio el movimiento de un punto podría caracterizarse mediante una función periódica sinusoidal tipo:

$$U(t)= A*seno(W*t) \quad \textbf{(dc.2)}$$

Donde $U(t)$ es la posición en un instante “t”, “A” es la amplitud máxima al desplazarse el nudo respecto de su posición de equilibrio y “W” es la velocidad angular (2PI/periodo) del movimiento armónico de periodo “T”.

Dado que el objeto es determinar las frecuencias y formas fundamentales (libres) de oscilación se puede dejar de considerar en este momento del estudio la influencia de los fenómenos disipativos asociados al amortiguamiento, siendo por tanto la ecuación:

$$F_{iner}+F_{elas}=0$$

(dc.3)

Al ser la aceleración la derivada segunda respecto del tiempo de la posición podríamos concluir (derivando la expresión 2) que la aceleración es:

$$Ac(t)=-W2*U(t)$$

(dc.4)

Por tanto la fuerza de inercia asociada a una masa móvil será:

$$F_{iner}=M*Ac=-M*W2*U(t)$$

(dc.5)

La fuerza elástica asociada a un desplazamiento “U” respecto de la posición de equilibrio para un elemento cuya vinculación se caracteriza por una constante elástica “K” será:

$$F_{elas}= K*U$$

(dc.6)

Asignando valores (expresiones 5 y 6 en 3) e igualando tenemos:

$$F_{iner}+F_{elas} = -M*W2*U+K*U=(-M*W2+K)*U=0$$
$$K=M*W2$$

(dc.7)

De lo que podemos concluir que una masa “M” vinculada con una rigidez “K” oscila entorno a su posición de equilibrio con una velocidad angular:

$$W=2*PI/T=Raiz\ (K/M)$$
$$T=2*Pi*Raiz(M/K)$$

(8)

Caracterizándose así el movimiento para un grado de libertad.

Análisis Modal Espectral.

Es una herramienta esencial para el estudio dinámico de los sistemas de múltiples grados de libertad. Estos sistemas pueden oscilar libremente en torno a una posición inicial de equilibrio, pero siempre dicho movimiento será la composición de una serie de figuras “básicas” cada una de las cuales está asociada a una velocidad angular “W_i” características. Para la determinación de estas formas básicas llamados “modos propios” se suele recurrir a establecer un problema de Autovalores sobre la ecuación de equilibrio entre fuerzas de inercia y elásticas (establecida por la dinámica clásica) planteada según su notación matricial.

Así el movimiento de los distintos nudos podría exponerse tal que así:

$$(X)= (X1,X2,\ldots Xn) = (A1,A2,\ldots An)*seno(W*t)$$

(am.1)

Por lo que su aceleración sería (sustituyendo y derivando):

$$(Ac)= (Ac1,Ac2,\ldots Acn) = -W2*(A1,A2,\ldots An)*seno(W*t)=-W2*(X1,X2,\ldots Xn)$$

(am.2)

Así las fuerzas de inercia para la matriz de masas [M] y las fuerzas elásticas para la matriz de rigidez [K] serían respectivamente:

$$(F_{iner})=[M](Ac)=-W2*[M](X)$$
$$(F_{elas})=[K](X)$$

(am.3)

Lo que conduciría a la siguiente ecuación:

$$-W2*[M](X)+[K](X)=0$$
$$(-W2[M]+[K])(X)=0$$
$$(-W2+[M-1][K])(X)=0$$

(am4)

Que conduce a la siguiente igualdad:

$$[M-1][K](X)=W2*(X)$$

(am.5)

Los vectores (X) que satisfagan la anterior ecuación serán los que permitan satisfacer la igualdad (am.4).

Se dice que un vector (V) es un autovector de la matriz [Ma] si al realizarse el producto [Ma](V) se obtiene un vector (Z) proporcial a (V) tal que (Z)=Au(V).

$$[Ma](V)=(Z)=Au(V) \quad \textbf{(am.6)}$$

(Z) y (V) son ambos autovectores de la matriz [Ma] y “Au” es el autovalor asociado a dicho vector. Si se aplica esta definición a la ecuación (am.5) tenemos que el cuadrado de la velocidad angular del sistema coincidirá con los autovalores de la matriz [M-1]*[K], siendo los autovectores correspondientes a dichos valores las formas modales asociadas.

Por tanto el sistema tenderá a oscilar de “forma natural” conforme a figuras proporcionales a las definidas por los autovectores y con una velocidad angular para cada caso establecida por la raíz del correspondiente autovalor de la matriz [M-1][K].

Ello es sumamente importante para el análisis de los fenómenos de resonancia y para la composición de movimientos, dado que cualquier posible forma de oscilación que tenga la estructura será como una combinación lineal de las oscilaciones naturales que de esta forma son obtenidas.

Planteamiento Energético. Rayleigh Ritz.

En un sistema conservativo, es decir en un caso como el nuestro en el que se desprecia el amortiguamiento, se verificará el principio de conservación de la energía, lo que implicará:

- Que el trabajo realizado por las fuerzas externas se equipara con el trabajo realizado por las fuerzas internas (Rayleigh Ritz), pues la energía se conserva en cada momento:

$$Wext+Wint=0 \quad \text{(pe.1)}$$

- Que la máxima energía cinética será equivalente a la máxima energía potencial del sistema (Principio de conservación de la energía).

$$Ec(max)=Ep(max) \quad \text{(pe.2)}$$

Ambos conceptos conducen a lo mismo: supuesta (o impuesta) una forma de oscilación (llamando $U_{(t)}$ a la posición en el tiempo), calculando la energía/trabajo del sistema podremos establecer una igualdad de la cual podemos deducir la frecuencia de oscilación.

Véase que al *imponer* una forma de movimiento periódica que nos sea *conocida* podemos desarrollar los anteriores conceptos de forma alternativa:

- Rayleigh Ritz: Las fuerzas externas son el producto de la masa por la aceleración, siendo la aceleración la derivada segunda respecto del tiempo de la posición, por lo que en una función sinusoidal ésta será el cuadrado de la posición por la velocidad angular al cuadrado cambiada de signo.

$$Ac_{(t)}= -W*W*U_{(t)}$$
$$Wext = Fuerza_{(t)}*Desplazamiento_{(t)}=-M*(W*W*U_{(t)})*U_{(t)} \quad \text{(pe.3)}$$

  El trabajo de las fuerzas internas se calculará a partir de la fuerza asociada a la elongación de los elementos, lo cual depende de la elongación de los elementos (Ley de Hooke), es decir:

$$Fb_{(t)}=Kb* incL$$
$$Wint = Fuerza_{(t)}*Desplazamiento_{(t)}=Kb* incL * incL \quad \text{(pe.4)}$$

  Para una oscilación completa el balance de ambos trabajos deberán ser nulo:

$$Integral[Wext+Wint]=0$$
$$Int[M*(W*W*U_{(t)})*U_{(t)}]dt=Int[Kb* incL * incL]dt$$
$$W*W= Int[Kb* incL * incL]dt/ Int[M*(W*W*U_{(t)})*U_{(t)}]dt \quad \text{(pe.5)}$$

  Lo que nos ha permitido establecer una relación de la que deducir la velocidad angular W supuesta una forma de oscilación.

- Conservación de la energía: Un oscilador simple (por ejemplo un péndulo) a lo largo de su movimiento verifica que la energía cinética del sistema es máxima cuando la energía potencial es nula y viceversa. La energía cinética se deduce de la velocidad máxima de la masa en movimiento:

$$V_{(max)}=W*U_{(max)}$$

$$Ec_{(max)}= 0.5*M*V_{(max)}*V_{(max)}=0.5*M*W*W*U_{(max)}*U_{(max)}$$

(pe.6)

Mientras que la energía potencial del sistema se deduce de la posición y de la rigidez:

$$Ep_{(max)}=0.5*Kb*U_{(max)}*U_{(max)}$$

(pe.7)

Igualando ambos extremos tenemos:

$$Ec_{(max)}= Ep_{(max)}$$

$$0.5*M*W*W*U_{(max)}*U_{(max)}= 0.5*Kb*U_{(max)}*U_{(max)}$$

$$W*W= [Kb*U_{(max)}*U_{(max)}]/[M*U_{(max)}*U_{(max)}]$$

(pe.8)

Lo que nos ha permitido establecer una relación de la que deducir la velocidad angular W supuesta una forma de oscilación.

Como se ha mencionado gracias a los principios energéticos anteriores podremos deducir la velocidad angular de oscilación para una forma de vibración. Esta forma la podríamos conocer por haber aplicado un método de cálculo que nos permitiera deducir la forma, o por ser impuesta como hipótesis de partida, lo cual es muy útil para conocer si una forma de oscilación impuesta puede provocar fenómenos de resonancia. En nuestro método de cálculo estos conceptos tendrán una fundamental utilidad a efectos de verificación, pues como deduciremos formas y frecuencias de oscilación podremos verificar energéticamente si la velocidad angular que se asocia a dicha forma es coincidente con la que predice el método.

Concepto

Partiendo de los fundamentos anteriormente expuestos realizaremos una secuenciada asociación de principios que nos permitirá deducir y verificar las formas modales de vibración y las frecuencias asociadas a una tenso estructura dada. El proceso sigue la siguiente pauta:

- Deducción de la *rigidez dinámica* a partir del método de la Densidad de Fuerza: Se realizará una adaptación del método general de cálculo que nos permitirá plantear para los nudos móviles del sistema una relación de rigidez entre los desplazamientos y las fuerzas aplicadas. Ello se logra gracias a la "condensación en X,Y,Z" de la parte del sistema asociada a los apoyos, y a la deducción de unas "densidades dinámicas de fuerza" que vinculan -para un pretensado determinado- la tensión y la longitud de la barra en equilibrio con la sección y el material correspondiente a la estructura real. Se relacionan pues las coacciones externas, la geometría, el pretensado de equilibrio, y las propiedades de las barras que vinculan entre si a los nudos en la estructura real. Entendemos por nudo móvil del sistema aquellos que no son apoyos y que por tanto son susceptible de oscilación.
- Dinámica clásica de sistemas estructurales y Análisis Modal: Planteada la citada "rigidez dinámica" estableceremos una asociación de masas (basada en la geometría, materiales y cargas actuantes) que nos permitirá desarrollar el problema clásico de análisis dinámico para múltiples grados de libertad, pero atendiendo a las singularidades del caso por ser un movimiento cuyo análisis se realiza de forma disociada en "X, Y, Z". Se deducirán de ello las formas y frecuencias modales de oscilación (idénticas para los distintos ejes por ser el movimiento forzosamente sincronizado en los ejes) y se analizará la influencia de la composición del movimiento "X, Y, Z" en la frecuencia.
- Principios energéticos: Realizaremos la verificación energética para los resultados obtenidos de la dinámica clásica.

La exposición y justificación detallada de la propuesta procurará plasmarse en un caso representativo. A lo largo de la exposición se comentará el sentido físico de las operaciones y consideraciones a medida que éstas se establecen.

Etapas del proceso. Exposición de ejemplo.

Establecida una tenso estructura cuyo comportamiento dinámico se desea estudiar los pasos a recorrer son:

- Planteamiento del sistema según la metodología de la Densidad de Fuerza para la geometría y vínculos establecidos.
- Reducción y resolución del sistema orientadamente al análisis dinámico. Análisis y condensación de los vínculos externos según “X, Y, Z”.
- Dimensionado y planteamiento de las matrices de masas y rigidez dinámica.
- Planteamiento y resolución del problema de autovalores.
- Verificación energética de la resolución modal.
- Determinación de las frecuencias y modos propios. Composición del movimiento.
- Análisis del movimiento. Límites de validez.

Para ilustrar dicho recorrido se plantea una tenso estructura suficientemente representativa como para exponer el método pero sin que se incurra en una matemática excesivamente farragosa, pues como se verá el número de variables y el tamaño de las matrices que se han de manejar es considerable.

Se plantea una estructura de 9 nudos y 16 barras, relacionados entre si tal que así:

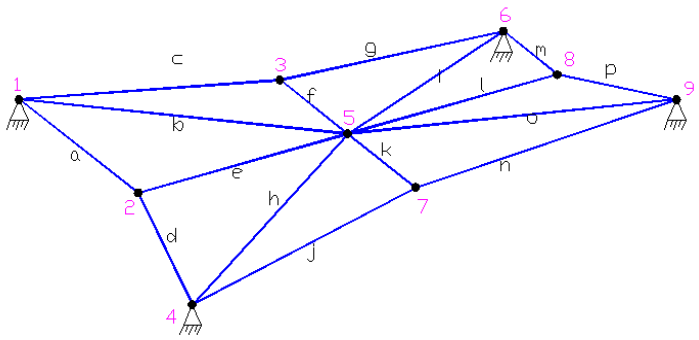


Figura 8: Caso elemental de estudio

Como puede apreciarse en la anterior imagen los nudos se han numerado del 1 al 9 y las barras se han denominado por letras de la “a” a la “p”.

La geometría de partida es un cuadrado de 10 metros de lado con un nudo central. Los nudos quedan definidos tal que habrá uno en el centro del cuadrado, uno en cada esquina y por último uno en la mitad de cada lado del contorno. Las barras por un lado enlazarán los nudos del contorno con el central y por otro lado constituirán el perímetro enlazando los nudos sucesivamente entre si.

Se han establecido apoyos en los vértices (nudos 1, 4, 6 y 9), quedando estos coaccionados frente a todo movimiento en “X, Y, Z” en una posición establecida. Los restantes nudos 2,3,5,7 y 8 tienen los movimientos permitidos, por los que los denominaremos nudos “libres”, si bien su movimiento siempre será de forma vinculada al resto de estructura dadas las barras que enlazan a los nudos entre si según se ha descrito.

Todos los nudos son susceptibles de recibir cargas, si bien toda carga aplicada en apoyo directamente se transformará en reacción, por lo que entendemos que a efectos de cálculo las cargas (“X,Y,Z”) se aplicarán en los nudos libres.

Planteamiento del equilibrio según el Método de la Densidad de Fuerza.

El método de la densidad de fuerzas establece una relación matricial del tipo:

$$[P]*(X,Y,Z)=(Fx,Fy,Fz) \quad \text{(a.1)}$$

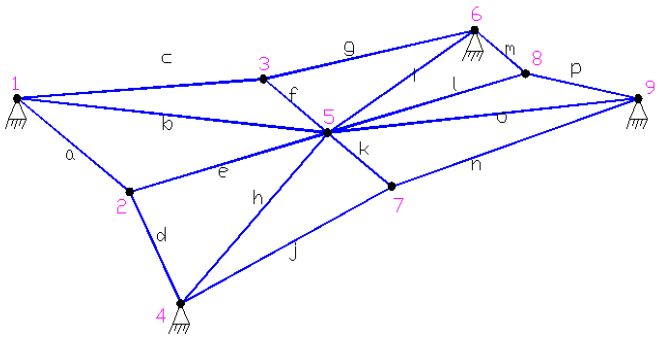
Donde [M] como veremos es una matriz obtenida al imponer el equilibrio “nudo a nudo” conforme los vínculos que las barras establecen entre los propios nudos. Conocidas unas fuerzas (Fx,Fy,Fz) la solución de este sistema permitiría concluir una forma (X,Y,Z) de equilibrio para la posición de los nudos. Aunque no es el objeto de la presente tesis la exposición detallada del método de la densidad de fuerza (véase Schek) necesariamente se debe repasar en profundidad su formulación y propiedades por ser base indispensable de la metodología propuesta.

Así la matriz [M] proviene del producto de la matriz de relaciones G y de la densidad de fuerzas D de la siguiente forma (donde Gt es G traspuesta):

$$[P]=[Gt]*[D]*[G] \quad \text{(a.2)}$$

Así el primer paso sería establecer la matriz de relaciones “G”, matriz que tiene tantas columnas como nudos y tantas filas como barras o vínculos entre nudos, pues cada

fila de esta matriz representaría una barra y cada columna un nudo. Las relaciones en forma de barra se establecen disponiendo un “1” en la columna que corresponde al nudo origen de dicha barra y un “-1” en la columna que corresponde al nudo extremo. Para nuestro ejemplo ello sería una matriz G con filas de la “A” a la “P” (16 barras) y columnas del “1” al “9” (9 nudos).



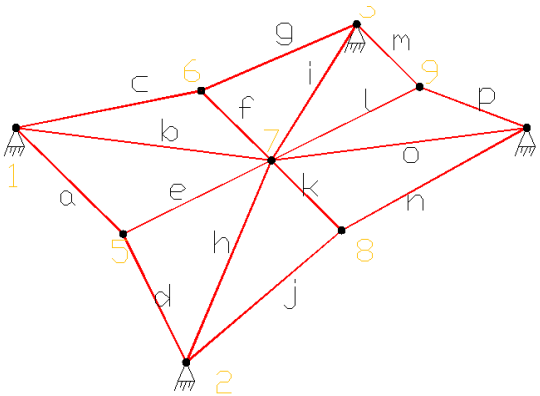
Matriz G

	1	2	3	4	5	6	7	8	9
A	1	-1	0	0	0	0	0	0	0
B	1	0	0	0	-1	0	0	0	0
C	1	0	-1	0	0	0	0	0	0
D	0	1	0	-1	0	0	0	0	0
E	0	1	0	0	-1	0	0	0	0
F	0	0	1	0	-1	0	0	0	0
G	0	0	1	0	0	-1	0	0	0
H	0	0	0	1	-1	0	0	0	0
I	0	0	0	0	1	-1	0	0	0
J	0	0	0	1	0	0	-1	0	0
K	0	0	0	0	1	0	-1	0	0
L	0	0	0	0	1	0	0	-1	0
M	0	0	0	0	0	1	0	-1	0
N	0	0	0	0	0	0	1	0	-1
O	0	0	0	0	1	0	0	0	-1
P	0	0	0	0	0	0	0	1	-1
Suma	3	3	3	3	8	3	3	3	3

Figura 9: Matriz de vínculos [G] para el caso elemental de estudio

Es interesante analizar los valores que aparecen *a título informativo* al pie de la matriz , en la “fila adicional” (valores que no pertenecen a la matriz G y que figuran bajo la fila “P”), pues representan el número total de elementos “no nulos” que hay en cada columna (3,3,3,3,8,3,3,3,3), es decir número de barras que confluyen a cada nudo (recordemos: a cada nudo se le asigna una columna). Para una columna las filas que tienen “1” serían las barras que tienen dicho nudo como origen, y las que tienen “-1” serían las que tendrían dicho nudo como extremo. Por ejemplo analizando el nudo 7 (columna 7) veríamos que las filas “J” y “K” tienen a dicho nudo como extremo a la par que es el origen de la barra “N”, tal y como podríamos constatar en la figura adjunta a la matriz. Naturalmente el valor asignado al pie de la matriz es el 3, pues al nudo 7 confluyen 3 barras.

Un concepto importante que merece volverse a recalcar es que cada columna de G expresa con un “1” las barras que parten de dicho nudo y con un “-1” las barras que llegan a dicho nudo. Si traspusiéramos G (obteniéndose Gt) cada fila representará el equilibrio entre las barras que salen de un nudo (en positivo) y las que lo acometen (en negativo) como veremos más adelante.



Descripción: Paraboloide de 9 nudos y 16 barras

G: Matriz de relaciones

BN	1	2	3	4	5	6	7	8	9
a	1	0	0	0	-1	0	0	0	0
b	1	0	0	0	0	0	-1	0	0
c	1	0	0	0	0	-1	0	0	0
d	0	1	0	0	-1	0	0	0	0
e	0	0	0	0	1	0	-1	0	0
f	0	0	0	0	0	1	-1	0	0
g	0	0	1	0	0	-1	0	0	0
h	0	1	0	0	0	0	-1	0	0
i	0	0	1	0	0	0	-1	0	0
j	0	1	0	0	0	0	0	-1	0
k	0	0	0	0	0	0	1	-1	0
l	0	0	0	0	0	0	1	0	-1
m	0	0	1	0	0	0	0	0	-1
n	0	0	0	1	0	0	0	-1	0
o	0	0	0	1	0	0	-1	0	0
p	0	0	0	1	0	0	0	0	-1
suma	3	3	3	3	3	3	8	3	3

Densidades de fuerza

Den(N/m)
7500
7500
7500
7500
7500
7500
7500
7500
7500
7500
7500
7500
7500
7500
7500
7500

Figura 10: Matriz de vínculos numerando los apoyos en primer lugar según el método

(La anterior matriz "G" está constituida según una numeración de nudos distinta a mostrada en la primera figura, de forma que los nudos que son apoyos se han numerado en primer lugar -del 1 al 4-. La justificación de ello se verá más adelante, y es la razón de que aún siendo la misma figura la matriz G difiera de la anterior)

El segundo paso sería constituir la matriz D de densidad de fuerzas, que es una matriz cuadrada cuya dimensión es el número de barras. En nuestro caso sería de dimensión 16, siendo una matriz *diagonal* cuyos elementos no nulos representan la "Densidad de Fuerza" que corresponde a cada barra.

D: Matriz de densidades Su unidad es Newton por metro (Sistema Internacional)

BN	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
a	7500	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
b	0	7500	0	0	0	0	0	0	0	0	0	0	0	0	0	0
c	0	0	7500	0	0	0	0	0	0	0	0	0	0	0	0	0
d	0	0	0	7500	0	0	0	0	0	0	0	0	0	0	0	0
e	0	0	0	0	7500	0	0	0	0	0	0	0	0	0	0	0
f	0	0	0	0	0	7500	0	0	0	0	0	0	0	0	0	0
g	0	0	0	0	0	0	7500	0	0	0	0	0	0	0	0	0
h	0	0	0	0	0	0	0	7500	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	0	0	7500	0	0	0	0	0	0	0
j	0	0	0	0	0	0	0	0	0	7500	0	0	0	0	0	0
k	0	0	0	0	0	0	0	0	0	0	7500	0	0	0	0	0
l	0	0	0	0	0	0	0	0	0	0	0	7500	0	0	0	0
m	0	0	0	0	0	0	0	0	0	0	0	0	7500	0	0	0
n	0	0	0	0	0	0	0	0	0	0	0	0	0	7500	0	0
o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7500	0
p	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7500

Figura 11: Matriz de Densidades del caso en estudio

La "Densidad de Fuerza" es la principal magnitud física que caracteriza a una barra (según la metodología de la densidad de fuerza) tal que D = F/L (fuerza dividido por longitud = constante). Por tanto dos barras con idéntica Densidad de Fuerza tendrán una relación de esfuerzos conforme a su relación de longitudes. Así como si de "muelles ideales se tratara" alcanzada la posición de equilibrio la solicitación en las barras guardará relación lineal con las longitudes de las piezas.

Volviendo a las expresiones (1) y (2) tenemos:

$$[P]*(X,Y,Z)=(Fx,Fy,Fz)$$

$$[P]=[Gt]*[D]*[G]$$

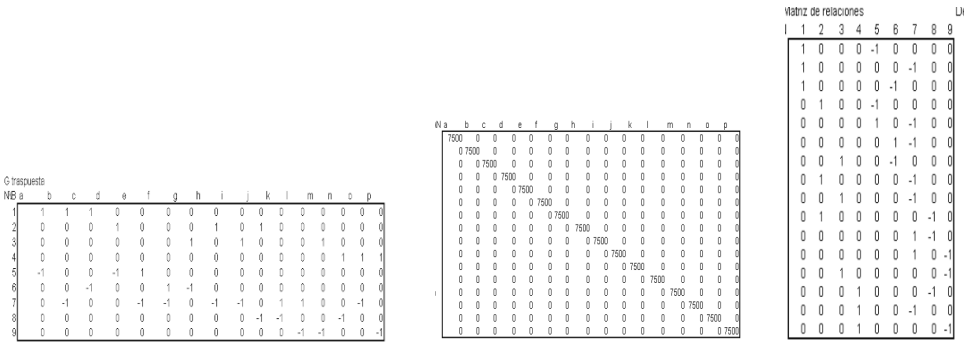


Figura 12: Producto de matrices para la obtención de la matriz [P] del sistema
(Véase las proporciones de las matrices que operan entre si)

Siendo (X,Y,Z) los vectores de coordenadas y (Fx,Fx,Fz) los vectores de esfuerzos sobre los nudos igualmente por proyecciones. Analizando el problema sólo en "X" tendríamos:

$$[P]*(X)=(Fx)$$

$$[Gt]*[D]*[G]*(X)=(Fx)$$

(a.3)

P: Matriz de rigidez equivalente GbxDxG

NN	1	2	3	4	5	6	7	8	9
1	22500	0	0	0	-7500	-7500	-7500	0	0
2	0	22500	0	0	-7500	0	-7500	-7500	0
3	0	0	22500	0	0	-7500	-7500	0	-7500
4	0	0	0	22500	0	0	-7500	-7500	-7500
5	-7500	-7500	0	0	22500	0	-7500	0	0
6	-7500	0	-7500	0	0	22500	-7500	0	0
7	-7500	-7500	-7500	-7500	-7500	-7500	60000	-7500	-7500
8	0	-7500	0	-7500	0	0	-7500	22500	0
9	0	0	-7500	-7500	0	0	-7500	0	22500

X

Vector de posiciones (m)

X	Y	Z
0	5	5
5	0	0
5	10	0
10	5	5
X5	Y5	Z5
X6	Y6	Z6
X7	Y7	Z7
X8	Y8	Z8
X9	Y9	Z9

=

Fuerzas **(Nw)**

Fx	Fy	Fz
Rx1	Ry1	Rz1
Rx2	Ry2	Rz2
Rx3	Ry3	Rz3
Rx4	Ry4	Rz4
0	0	-14700,1
0	0	-14700,1
0	0	-14700,3
0	0	-14700,1
0	0	-14700,1

Los valores se multiplican por 9.8 para pasar a Newton

Figura 13: Ilustración que destaca la parte del sistema que opera con los apoyos.

Analizando la expresion (3) por partes tenemos:

[G]*(X) → El resultado de dicho producto será un vector (de dimensión el número de barras) tal que a cada fila le corresponderá el incremento de coordenadas en proyección X de cada barra (análogamente Y,Z). Llamémoslo (I_{nc}), es decir:

$$[G]*(X)=(I_{nc})$$

(a.4)

[D]*[G]*(X) = [D]*(I_{nc}) → El resultado de dicho producto será un vector (de dimensión el número de barras) tal que a cada fila le corresponderá el esfuerzo en proyección X asociado a cada barra según la geometría de equilibrio. Llamémoslo (F_{bar}), es decir:

$$[D]*(I_{nc}) = (F_{bar}) \tag{a.5}$$

$[Gt]*[D]*[G]*(X)=[Gt]*(F_{bar})$ → El resultado de dicho producto será un vector (de dimensión el número de nudos), en el que cada fila representará a nivel de nudo el balance entre las fuerzas en proyección X asociado las barras que se inician en dicho nudo (1) y las que llegan al mismo (-1).. Llamémoslo (R_{nud}), es decir:

$$[Gt]*(F_{bar})= (R_{nud}) \tag{a.6}$$

En un sistema en equilibrio la resultante sobre los nudos tendrá que tener correspondencia con las fuerzas externas, por tanto:

$$[Gt]*[D]*[G]*(X)= (R_{nud}) = (Fx) \tag{a.7}$$

Que viene a ser un sistema lineal de ecuaciones dado que G se establece a partir de los vínculos entre barras y D representa las Densidades de fuerza asociadas a cada barra, resultando un sistema con tantas ecuaciones como nudos y donde la incógnita es la posición de estos.

Es interesante apreciar que en una tenso estructura modelada con este método "D" guarda relación con el valor absoluto de las cargas internas de la tenso estructura, es decir para una geometría de equilibrio una mayor densidad de fuerzas implica un mayor pretensado del sistema. Ello se entiende bien supuesto un sistema en que la forma sea resultado únicamente de la adaptación de la red a una geometría del contorno (apoyos), los esfuerzos internos aumentarán de manera proporcional al incremento de la Densidad de Fuerza. Ello es importante desde un punto de vista dinámico, pues equivaldría -haciendo un símil- a aumentar la tensión en la cuerda de una guitarra durante el proceso de afinado. Véase así mismo que la respuesta de una tenso estructura ante una carga externa depende igualmente de la razón de proporcionalidad entre el pretensado del sistema y la acción externa. Si la carga externa es "relativamente pequeña" frente al pretensado la adaptación de forma será menor en comparación a si fuera "relativamente grande". Por tanto deberá haber una proporcionalidad entre la Densidad de fuerza y las acciones del sistema para que su comportamiento sea "congruente".

Igualmente se debe destacar la importancia que tiene el manejo de la Densidad de Fuerza de forma *particularizada* (actuando sobre barras determinadas) en la búsqueda de formas, pues su ajuste es la metodología habitual para adaptar la figura resultante a las necesidades del diseñador -en la medida que lo permita el equilibrio-. Partiendo de un sistema en equilibrio, el aumentar la densidad de fuerza para una barra determinada tendrá la consecuencia habitual de generar un acortamiento de la misma (a costa de alargar otras) en la figura de equilibrio.
Por último destacar la linealidad del sistema, lo que implica por ejemplo en "X" que:

$$[P]*(Xa+Xb)=Fx,a+Fx,b$$
$$[P]*(X_{forma}+X_{oscilación}) = Fx,_{equilibrio}+Fx,_{inercia} \quad \text{(a.8)}$$

Es decir: es aplicable el principio de superposición, si bien como veremos ello deberá manejarse con cuidado pues no siempre su aplicación será posible de manera directa a lo largo del método. El uso de esta propiedad facilitará el análisis de la oscilación como estado "complementario" al estado de equilibrio, dado que se podrán disociar las fuerzas de inercia de las fuerzas de equilibrio, siempre que se esté dentro de unos límites, ello matemáticamente se expondrá y verificará con un ejemplo numérico.

Reducción y resolución del sistema orientado al análisis dinámico. Condensación de apoyos.

Como se expresó con anterioridad el método de la densidad de fuerza establece una relación lineal entre la posición de equilibrio de los nudos y las fuerzas actuantes según:

$$[P]*(X,Y,Z)=(Fx,Fy,Fz) \quad \text{(a.1)}$$

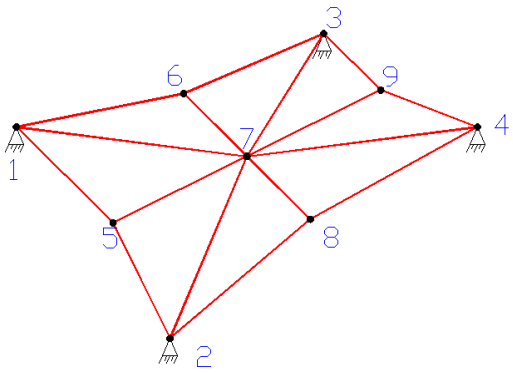
Expresión resoluble directamente conociendo las cargas sobre los nudos (Fx,Fy,Fz) o bien conociendo la forma de equilibrio (X,Y,Z): conocer lo uno permite deducir lo otro.

Desgraciadamente lo que se conoce habitualmente como dato de partida es exclusivamente la posición de los nudos que son apoyos así como las cargas que actúan sobre los nudos que no son apoyos (sin embargo no se conocen las

reacciones), por lo que la resolución del sistema anterior se complica notablemente al haber incógnitas “enredadas” a ambos lados de la igualdad. Para la resolución de esta circunstancia “condensaremos los apoyos”.

$$[P]*(X,Y,Z)=[G_t]*[D]*[G]*(X,Y,Z)=(F_x, F_y, F_z)$$

La “condensación de apoyos” persigue reordenar el sistema , pudiéndose gracias a ello deducir directamente la forma de equilibrio. Ello facilitará además el análisis dinámico de la parte móvil del sistema pues los apoyos “se desligan” del resto de nudos del sistema mediante la introducción de fuerzas sobre aquellos nudos con los que los apoyos están inmediatamente vinculados, es decir, se introducen las fuerzas equivalentes a las que se transmitirían desde los apoyos al resto de la estructura. Ello es posible al ser matemáticamente deducibles dichas cantidades como a continuación se procederá a exponer, siendo la clave el seguir un adecuado ordenamiento del sistema para que el cálculo sea sistemático. El “truco” para la readaptación del sistema comienza ordenando los apoyos en la matriz de vínculos G **siempre los primeros** (asignándoles las primeras columnas).



Zona de Apoyos

B\N	1	2	3	4	5	6	7	8	9
a	1	0	0	0	-1	0	0	0	0
b	1	0	0	0	0	0	-1	0	0
c	1	0	0	0	0	-1	0	0	0
d	0	1	0	0	-1	0	0	0	0
e	0	0	0	0	1	0	-1	0	0
f	0	0	0	0	0	1	-1	0	0
g	0	0	1	0	0	-1	0	0	0
h	0	1	0	0	0	0	-1	0	0
i	0	0	1	0	0	0	-1	0	0
j	0	1	0	0	0	0	0	-1	0
k	0	0	0	0	0	0	1	-1	0
l	0	0	0	0	0	0	1	0	-1
m	0	0	1	0	0	0	0	0	-1
n	0	0	0	1	0	0	0	-1	0
o	0	0	0	1	0	0	-1	0	0
p	0	0	0	1	0	0	0	0	-1

Figura 14: Esquema de reordenación de apoyos

Para el ejemplo que venimos tratando se traduciría en las siguientes matrices [G] y [D]:

(Se han cambiado las densidades de fuerza y cargas para operarse con valores más asequibles)

G: Matriz de relaciones

B\N	1	2	3	4	5	6	7	8	9
a	1	0	0	0	-1	0	0	0	0
b	1	0	0	0	0	0	-1	0	0
c	1	0	0	0	0	-1	0	0	0
d	0	1	0	0	-1	0	0	0	0
e	0	0	0	0	1	0	-1	0	0
f	0	0	0	0	0	1	-1	0	0
g	0	0	1	0	0	-1	0	0	0
h	0	1	0	0	0	0	-1	0	0
i	0	0	1	0	0	0	-1	0	0
j	0	1	0	0	0	0	0	-1	0
k	0	0	0	0	0	0	1	-1	0
l	0	0	0	0	0	0	1	0	-1
m	0	0	1	0	0	0	0	0	-1
n	0	0	0	1	0	0	0	-1	0
o	0	0	0	1	0	0	-1	0	0
p	0	0	0	1	0	0	0	0	-1

D: Matriz de densidades Su unidad es Newton por metro (Sistema Internacional)

B\N	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
a	3,9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
b	0	3,9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
c	0	0	3,9	0	0	0	0	0	0	0	0	0	0	0	0	0
d	0	0	0	1,5	0	0	0	0	0	0	0	0	0	0	0	0
e	0	0	0	0	1,5	0	0	0	0	0	0	0	0	0	0	0
f	0	0	0	0	0	1,5	0	0	0	0	0	0	0	0	0	0
g	0	0	0	0	0	0	1,5	0	0	0	0	0	0	0	0	0
h	0	0	0	0	0	0	0	1,5	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	0	0	1,5	0	0	0	0	0	0	0
j	0	0	0	0	0	0	0	0	0	1,5	0	0	0	0	0	0
k	0	0	0	0	0	0	0	0	0	0	1,5	0	0	0	0	0
l	0	0	0	0	0	0	0	0	0	0	0	1,5	0	0	0	0
m	0	0	0	0	0	0	0	0	0	0	0	0	1,5	0	0	0
n	0	0	0	0	0	0	0	0	0	0	0	0	0	3,9	0	0
o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3,9	0
p	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3,9

Figura 15: Matrices [G] y [D] del ejemplo ordenadas numerando los apoyos al principio

De este orden resulta que:

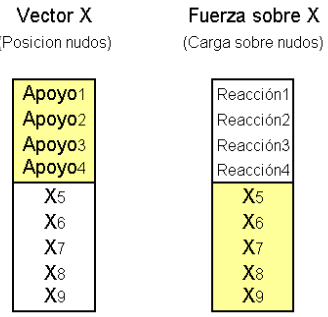


Figura 16: Esquema que muestra las partes de los vectores que son conocidas

Se destacan en amarillo: una por se la posición de los apoyos otra por ser las cargas actuantes

Analizando el producto de una fila de [P] por el vector de posiciones (X) vemos que ambas partes del producto tienen partes constantes –Destacadas en amarillo- que son directamente operables.

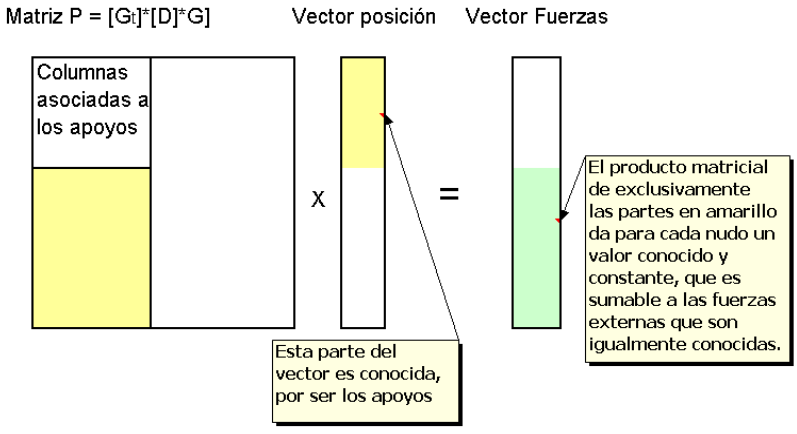


Figura 17 Condensación de apoyos, estructuración de las partes costantes del sistema

Partes de que representamos a continuación sombreadamente en el sistema:

P: Matriz de rigidez equivalente GtxDxG

NNN	1	2	3	4	5	6	7	8	9
1	11,7	0	0	0	-3,9	-3,9	-3,9	0	0
2	0	4,5	0	0	-1,5	0	-1,5	-1,5	0
3	0	0	4,5	0	0	-1,5	-1,5	0	-1,5
4	0	0	0	11,7	0	0	-3,9	-3,9	-3,9
5	-3,9	-1,5	0	0	6,9	0	-1,5	0	0
6	-3,9	0	-1,5	0	0	6,9	-1,5	0	0
7	-3,9	-1,5	-1,5	-3,9	-1,5	-1,5	16,8	-1,5	-1,5
8	0	-1,5	0	-3,9	0	0	-1,5	6,9	0
9	0	0	-1,5	-3,9	0	0	-1,5	0	6,9

X

Vector de posiciones (m)

X	Y	Z
0	1,45	1,3
1,3	0	0
1,3	2,9	0
2,6	1,45	1,3
X5	Y5	Z5
X6	Y6	Z6
X7	Y7	Z7
X8	Y8	Z8
X9	Y9	Z9

=

Fuerzas **(Nw)**

Fx	Fy	Fz
Rx1	Ry1	Rz1
Rx2	Ry2	Rz2
Rx3	Ry3	Rz3
Rx4	Ry4	Rz4
0	0	-1,01
0	0	-1,01
0	0	-1,25
0	0	-1,01
0	0	-1,01

Los valores se multiplican por 9.8 para pasar a Newton

Figura 18: Representación de las partes constantes del sistema del ejemplo desarrollado.

La clave es hacer las operaciones de dichas "partes constantes" externamente al sistema:

Se multiplican las partes sombreadas para obtener los valores que modifican la carga

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
5	-3,9	-1,5	0	0	0	0	0	0	0
6	-3,9	0	-1,5	0	0	0	0	0	0
7	-3,9	-1,5	-1,5	-3,9	0	0	0	0	0
8	0	-1,5	0	-3,9	0	0	0	0	0
9	0	0	-1,5	-3,9	0	0	0	0	0

*

X	Y	Z
0	1,45	1,3
1,3	0	0
1,3	2,9	0
2,6	1,45	1,3
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0

=

Qxm	Qym	Qzm
0	0	0
0	0	0
0	0	0
0	0	0
-1,95	-5,655	-5,07
-1,95	-10,005	-5,07
-14,04	-15,66	-10,14
-12,09	-5,655	-5,07
-12,09	-10,005	-5,07

Figura 19: Operación de condensación de apoyos del ejemplo

Obtenidas las "fuerzas de condensación" dichos valores se compondrán con las fuerzas externas actuantes sobre los nudos que no son apoyos (fuerzas denominadas Qxm, Qym, Qzm en la imagen), reformulándose el sistema. Para ello se tendrá que replantear la matriz [P] trasformándose en [P_{mod}]. Véase que de ello resultará "un sistema mixto" en el que quedan agrupadas todas las incógnitas (reacciones "combinadas" sobre los apoyos y posición de nudos) al mismo lado de la ecuación:

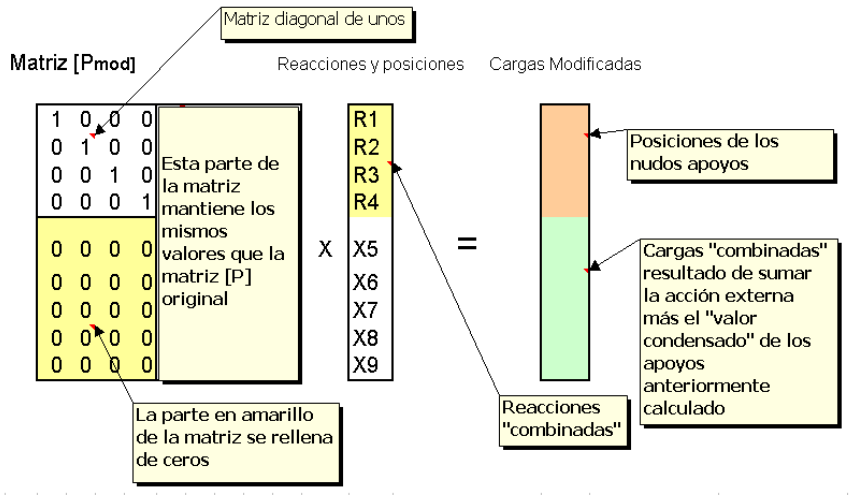


Figura 20: Esquema de reordenación de las matrices y vectores durante el proceso de condesación de apoyos

Es importante apreciar que las reacciones "combinadas" **no son** directamente las reacciones sobre los apoyos: Las reacciones se deducen una vez conocidas las posiciones de los nudos del sistema original [P]*(X)=(Fx)

El sistema así "reformulado" para la resolución directa de la geometría de equilibrio quedará planteado de la siguiente manera para nuestro ejemplo:

P*: Matriz corregida, sistema REORDENADO

NN	1	2	3	4	5	6	7	8	9
1	1	0	0	0	-3,9	-3,9	-3,9	0	0
2	0	1	0	0	-1,5	0	-1,5	-1,5	0
3	0	0	1	0	0	-1,5	-1,5	0	-1,5
4	0	0	0	1	0	0	-3,9	-3,9	-3,9
5	0	0	0	0	6,9	0	-1,5	0	0
6	0	0	0	0	0	6,9	-1,5	0	0
7	0	0	0	0	-1,5	-1,5	16,8	-1,5	-1,5
8	0	0	0	0	0	0	-1,5	6,9	0
9	0	0	0	0	0	0	-1,5	0	6,9

X

X	Y	Z
Rx1	Ry1	Rz1
Rx2	Ry2	Rz2
Rx3	Ry3	Rz3
Rx4	Ry4	Rz4
X5	Y5	Z5
X6	Y6	Z6
X7	Y7	Z7
X8	Y8	Z8
X9	Y9	Z9

=

F*: Matriz de posición de nudos y cargas modificada

X	Y	Z
0	1,45	1,3
1,3	0	0
1,3	2,9	0
2,6	1,45	1,3
1,95	5,655	4,063
1,95	10,01	4,063
14,04	15,66	8,893
12,09	5,655	4,063
12,09	10,01	4,063

Figura 21: Reordenación del sistema del ejemplo

$$[P_{mod}]*(R_x,X)=(F_x+Q_{m,x})$$

$$[P_{mod}]*(R_y,Y)=(F_y+Q_{m,y})$$

$$[P_{mod}]*(R_z,Z)=(F_z+Q_{m,z})$$

Separadamente en X,Y,Z siendo Rx,Ry,Rz las "reacciones combinadas" (b.1)

Véase como al ser ceros la parte destacada de la [Pmod]*(R,X)=(Fx+Qm,x)matriz [P_{mod}], la influencia de las reacciones sobre la parte sombreada en azul del sistema (correspondiente a los nudos móviles, que no son apoyos) se desvincula de las reacciones al operarse el producto matricial. Es esencial apreciar esto, pues esta reordenación del sistema es clave y nos permite visualizar y reducir directamente la parte que rige el comportamiento dinámico del sistema. Véase como las columnas iniciales correspondientes a los apoyos son vectores "canónicos" respecto al resto –ortogonales y de módulo unidad-, por lo que al ser [P_{mod}] invertible (es un sistema resoluble) necesariamente la parte a continuación destacada contendrá una "base" vectorial:

NN	1	2	3	4	5	6	7	8	9
1	1	0	0	0	-3,9	-3,9	-3,9	0	0
2	0	1	0	0	-1,5	0	-1,5	-1,5	0
3	0	0	1	0	0	-1,5	-1,5	0	-1,5
4	0	0	0	1	0	0	-3,9	-3,9	-3,9
5	0	0	0	0	6,9	0	-1,5	0	0
6	0	0	0	0	0	6,9	-1,5	0	0
7	0	0	0	0	-1,5	-1,5	16,8	-1,5	-1,5
8	0	0	0	0	0	0	-1,5	6,9	0
9	0	0	0	0	0	0	-1,5	0	6,9

Figura 22: Parte dinámico del la matriz [Pmod] del ejemplo

Al ser una "base" linealmente independiente dicha parte de la matriz [P_{mod}] contendrá toda la información necesaria para la caracterización de la parte "móvil" del sistema, a continuación destacada:

P*: Matriz corregida, sistema REORDENADO

NN	1	2	3	4	5	6	7	8	9
1	1	0	0	0	-3,9	-3,9	-3,9	0	0
2	0	1	0	0	-1,5	0	-1,5	-1,5	0
3	0	0	1	0	0	-1,5	-1,5	0	-1,5
4	0	0	0	1	0	0	-3,9	-3,9	-3,9
5	0	0	0	0	6,9	0	-1,5	0	0
6	0	0	0	0	0	6,9	-1,5	0	0
7	0	0	0	0	-1,5	-1,5	16,8	-1,5	-1,5
8	0	0	0	0	0	0	-1,5	6,9	0
9	0	0	0	0	0	0	-1,5	0	6,9

X

X	Y	Z
Rx1	Ry1	Rz1
Rx2	Ry2	Rz2
Rx3	Ry3	Rz3
Rx4	Ry4	Rz4
X5	Y5	Z5
X6	Y6	Z6
X7	Y7	Z7
X8	Y8	Z8
X9	Y9	Z9

=

F*: Matriz de posición de nudos y cargas modificada

X	Y	Z
0	1,45	1,3
1,3	0	0
1,3	2,9	0
2,6	1,45	1,3
1,95	5,655	4,063
1,95	10,01	4,063
14,04	15,66	8,893
12,09	5,655	4,063
12,09	10,01	4,063

Figura 23: Planteamiento del sistema dinámico reducido del ejemplo

Al ser un sistema lineal podremos aplicar el principio de superposición, lo cual lo haremos sobre la parte reducida del sistema por estar desvinculada del resto de la ecuación (gracias a la reordenación y por contener una base L.I.).

$$[P_{mod,red}]*(Xa+Xb)=Fx,a+Fx,b$$

$$[P_{mod,red}]*(X_{forma}+X_{oscilación}) = Fx,_{equilibrio}+Fx,_{inercia}$$

$$[P_{mod,red}]*(X_{oscilación}) = Fx,_{inercia}$$

NOTA: Fx,equilibrio contendrá las fuerzas de condensación (b.2)

Concluyéndose así el sistema reducido a su parte dinámica, que será idéntica en X,Y,Z:

6,9	0	-1,5	0	0	X5	=	Finer,x5
0	6,9	-1,5	0	0	X6		Finer,x6
-1,5	-1,5	16,8	-2	-2	X7		Finer,x7
0	0	-1,5	7	0	X8		Finer,x8
0	0	-1,5	0	7	X9		Finer,x9

Figura 24: Sistema dinámico reducido del ejemplo

$$[P_{din}]*(X_m,Y_m,Z_m)=(F_{ix},F_{iy},F_{iz})$$

(b.3)

Destaquemos que los apoyos en cualquier caso nunca tendrían fuerzas de inercia por ser inmóviles por definición, por lo que las fuerzas de inercia existirán solamente en los nudos móviles, lo cual es un argumento agregado a los anteriormente expuestos de cara a reducir el sistema.

Véase que el mismo “tensor de rigidez” gobierna el comportamiento en las tres direcciones, ello es esencial tenerlo en cuenta a la hora de analizar la composición de movimiento pues se trata de una rigidez “única” que deberá compartirse entre las tres direcciones. Ello habrá de ser coherente con que todo movimiento “natural” (modo propio de oscilación) deberá estar sincronizado en X,Y,Z. Estos dos conceptos son esenciales a la hora de entender la dinámica del sistema como más detalladamente se analizará.

Hay que destacar el sentido físico de las “fuerzas de condensación” (Qxm,Qym,Qzm). Véase que el sistema general es idéntico en X,Y,Z salvo por las cargas externas actuantes y salvo por las fuerzas de condensación anteriormente deducidas. Para un sistema con un determinado grado de pretensado estas fuerzas de condensación representan la rigidez geométrica que la posición de los apoyos aporta para cada dirección X,Y,Z, siendo un *valor constante* que se agrega a las cargas externas. La importancia de la geometría de los apoyos frente a las acciones externas dependerá del “peso relativo” de unos valores frente a los otros de forma diferencia en X,Y,Z.

Dimensionado y establecimiento de la Matriz de rigidez dinámica.

En el apartado anterior hemos tratado el cómo resolver el sistema general:

$$[P]*(X,Y,Z)=(Fx,Fy,Fz) \tag{a.1}$$

Gracias a su transformación mediante el reordenamiento de incógnitas y el condensando de apoyos. Así tras ello tenemos el sistema separadamente en X,Y,Z:

$$[P_{mod}]*(R_x,X)=(F_x+Q_{m,x})$$
$$[P_{mod}]*(R_y,Y)=(F_y+Q_{m,y})$$
$$[P_{mod}]*(R_z,Z)=(F_z+Q_{m,z}) \tag{b.1}$$

Deduciéndose así la forma de equilibrio de un sistema del que conocemos como se relacionan los nudos entre si, las cargas que actuantes sobre los mismos y las

densidades teóricas de fuerza que se han asignado a la hora de caracterizar los vínculos. El proyectista ajustando las densidades de fuerza habrá logrado establecer una determinada forma de equilibrio con un determinado grado de pretensado: La forma está siempre asociada a su estado tensional, por lo que cabe cierta libertad de diseño al poderse ajustar detalladamente la densidad de fuerza teórica de cada uno de los vínculos.

Se expuso igualmente la parte reducida dentro del sistema general que contiene las relaciones necesarias para caracterizar el comportamiento dinámico de los nudos móviles:

$$[P_{din}]^*(X_m, Y_m, Z_m)=(F_{ix}, F_{iy}, F_{iz}) \quad \text{(b.3)}$$

Hasta este punto los materiales empleados son “ideales” en tanto y cuanto presentan sólo una elongabilidad sin restricciones según la definición de densidad de fuerza, sin que se atisbe en el sistema la realidad de los materiales de la construcción (secciones, límites elásticos, etc…). El análisis dinámico requerirá pues de la caracterización de los materiales que ciertamente constituyen la estructura, de forma que se manifiesten las propiedades mecánicas de los vínculos que conectan ciertamente los nudos entre si tal y como son en realidad y bajo los fenómenos físicos que las condicionan. El objeto del presente apartado es la obtención e introducción de la rigidez real de la estructura de forma que dinámicamente quede caracterizada en el sistema reducido tal y como es.

Como hemos visto el sistema general (a.1) se ha planteado imponiendo un estricto equilibrio vectorial nudo a nudo. De su resolución se obtiene una *geometría* según la cual los vínculos así posicionados equilibran las cargas actuantes sobre los nudos desarrollando unas solicitaciones determinadas conocidas. Por tanto si disponemos una serie de barras tales que al aplicarles las deducidas solicitaciones éstas alcancen la teórica longitud de equilibrio, la configuración que resulte estará igualmente en equilibrio.

Por tanto para una barra o vínculo determinado lo que tendremos que conocer tras la resolución de sistema -del que obtenemos la forma- es que longitud resulta para la misma en su posición de equilibrio, que solicitación presenta en dicho estado y qué propiedades mecánicas (límite elástico y módulo de elasticidad) dotan al material que

conformará en realidad dicha barra o vínculo. Conocidos estos datos los pasos a seguir serían:

1) Resuelto el sistema en X,Y,Z obtendremos las longitudes de las barras componiendo vectorialmente las componentes X,Y,Z según la forma de equilibrio. Conocidas las longitudes y haciendo uso de la Densidad de Fuerza obtendremos las solicitaciones sobre las barras.

$$[G]^*(X,Y,Z)=(L_x,L_y,L_z)$$

Para cada barra "i" la longitud de la misma así como la fuerza será:

$$Li=\sqrt{L_x^2+L^2y+L_z}$$
$$F_i=D_i\times L_i$$

Lo que matricialmente se expresará como:

$$(F)=[D]^*(L)$$

(Rd.1)

2) Dimensionaremos las barras para la solicitación anteriormente obtenida, de tal modo que no se supere el límite elástico conforme a un determinado coeficiente de seguridad y material, obteniéndose una sección para cada barra

$$\Omega=\frac{F\times C_s}{L_{iE}}$$

C_S=Coeficiente de seguridad

L_{iE}= Límite elástico

(Rd.2)

3) Establecida dicha sección y conocida la longitud que habrá de tener la barra "i" en la forma de equilibrio calcularemos el "acortamiento inicial" ($A_{in,i}$) que hemos de dar a la barra para que al entrar ésta en carga adquiera la longitud precisa en la forma de equilibrio. La barra "i" con un módulo de elasticidad (Ei) habrá de acortarse:

$$A_{in,i} = \frac{F_i \times L_i}{\Omega_i \times E_i}$$

(Rd.3)

4) Conocida la sección, longitud y módulo de elasticidad del material deducimos la Densidad de Forma correspondería a dicha barra, es decir su elasticidad lineal equivalente. La llamaremos Densidad Dinámica de Forma (DDF).

$$L_{0,i} = L_i - A_{in,i}$$

$$\Delta F_i = \frac{\Omega_i \times E_i}{L_{0,i}} \times \Delta L_i$$

$$DDF_i = \frac{\Omega_i \times E_i}{L_{0,i}}$$

(Rd.4)

5) Calcularemos el módulo de elasticidad tangente al aplicar la fórmula de Ernst **(*)** a la barra “i”. Obtendremos su densidad dinámica de fuerza Corregida (DDFr,i)

$$DDF_{r,i} = DDF_i/(1+R_i)$$

$$R = \frac{q^2.d^2.E.A}{(12T^3)}$$

Siendo q el peso lineal de la barra, d su proyección horizontal, A su sección transversal, E el módulo de elasticidad aparente de la barra, y T la carga de trabajo de la barra

(Rd.5)

6) Reformulamos el sistema utilizando para caracterizar los vínculos las Densidades Dinámicas de Forma Corregidas (DDFr) de todas las barras, que se habrán obtenido según el procedimiento descrito y obtendremos el sistema de cálculo:

$$[Pdin]_{DDF} = [Gt]*[DDFr]*[G]$$

(Rd.6)

Véase que en las estructuras reales el “acortamiento inicial” (A_i) será una magnitud pequeña, que será congruente con la deformación de la barra dentro de su rango elástico (propio del material constituyente). Igualmente serán pequeñas las variaciones de longitud de las barras durante las oscilaciones que estudiemos, pues la amplitud del movimiento será presumiblemente menor que la que pudiera llevar los materiales a su plastificación o a las barras a su decompresión -como más adelante veremos cuando estudiemos el movimiento-. Por tanto en nuestro ámbito de estudio, en el que los movimientos están limitados respecto de la posición de equilibrio, es una hipótesis razonable el considerar reducidas las variaciones de longitud de las barras en relación al tamaño de éstas, por lo que la Densidad Dinámica de Forma (DDFr) se comportará como si los materiales fueran ideales y la variación de tensión en la barra no suponga una variación del módulo de elasticidad tangente (así se mantendrá la proporcionalidad entre incremento de desplazamiento e incremento de fuerza), es decir cuando se reformule el sistema se podrán usar dichos valores en el sistema como si de Densidades de Fuerza “Teóricas” se trataran.

Así, como hemos dicho reformulamos nuevamente el sistema:

$$[Pdin]_{DDFr}*(X,Y,Z)=[G_t]*[DDF_r]*[G]*(X,Y,Z)=(F_x, F_y, F_z)$$

(Rd.6)

Plasmándose de esta forma los parámetros reales de los vínculos que conforman la estructura coherentemente con una geometría acorde al pretensado de diseño.

El sistema que se utilizará para estudiar el comportamiento dinámico será la parte reducida de la anterior expresión:

$$[P_{din}]_{DDFr}*(X_m,Y_m,Z_m)=(F_{ix},F_{iy},F_{iz})$$

(Rd.7)

Con una $[P_{din}]_{DDF}$ en las que las Densidades de Fuerza utilizadas en su confección serían obviamente las que se han calculado y denominado como Densidades Dinámicas de Fuerza (DDF). Las Densidades Dinámicas de Forma por tanto *estarán asociadas a un estado tensional* (pues se corresponden con un cierto pretensado y acortamiento inicial propios de la forma), a una determinada sección (obtenida al

dimensionar para el esfuerzo de equilibrio alcanzado) y a un material (cuyos módulo de elasticidad y límite elástico han sido utilizadas en la deducción de parámetro).

(*) Las barras bajo la acción de la gravedad dejan de ser rectilíneas, ello se da en mayor proporción en la medida que la horizontalidad y el peso sean relevantes reapecto de la tracción actuante. Cuando una barra en ésta circunstancia se somete es axialmente traccionada tendrá una elongación distinta a la que tendrían si fuera rectilínea ideal. La elasticidad equivalente se denomina "tangente" y para un tirante se modeliza según la fórmula de Ernst anteriormente expuesta. Cuando es conocida la carga inicial, la geometría y su sección se podrá deducir la elasticidad corregida, lo cual es especialmente interesante para el análisis de estructuras de cierta entidad en las que haya tirantes largos y pesados. Su implementación es sencilla cuando la corrección se limita al módulo tangente, sin embargo el uso del módulo de elasticidad secante –cuya aproximación es aún mejor- sería más complejo de programar dado que requiere introducir la variación de tensión por la propia corrección. Ese sería un procedimiento implícito de cálculo excesivamente costoso para una mejora de precisión que se alcanza de forma razonable con el módulo de elasticidad tangente.

Planteamiento del problema de autovalores. Matriz de Masas.

En el anterior apartado hemos logrado obtener una expresión matricial que caracteriza la rigidez "en el sentido tradicional" para una tenso estructura en equilibrio y dimensionada, pues se relaciona el desplazamiento de los nudos ante una acción que tienda a desplazar la forma respecto de su posición de equilibrio inicial teniéndose en cuenta las propiedades reales de las barras.

Véase mediante un ejemplo las coordenadas de equilibrio de un sistema ante diferentes cargas y densidades de fuerza. Se aprecia que la relación entre fuerza y desplazamiento (rigidez) es lineal, al verificarse la proporcionalidad simple entre incremento de cargas e incremento de desplazamientos, independientemente de la carga de partida.

Densidad de fuerza 5000 Nw/m, Carga Partida "Z" -1000 Nw

Qz=-1000 Nw

X	Y	Z
0	5	5
5	0	0
5	10	0
10	5	5
3,3333	3,333	1,618
3,3333	6,667	1,618
5	5	1,814
6,6667	3,333	1,618
6,6667	6,667	1,618

Qz=-1200 Nw

X	Y	Z	Inc -200	Inc/200
0	5	5		
5	0	0		
5	10	0		
10	5	5		
3,3333	3,333	1,44	0,1764	0,00088
3,3333	6,667	1,44	0,1764	0,00088
5	5	1,68	0,1372	0,00069
6,6667	3,333	1,44	0,1764	0,00088
6,6667	6,667	1,44	0,1764	0,00088

Qz=-1300 Nw

X	Y	Z	Inc -300	Inc/300
0	5	5		
5	0	0		
5	10	0		
10	5	5		
3,3333333	3,333333	1,35339	0,2646	0,00088
3,3333333	6,666667	1,35339	0,2646	0,00088
5	5	1,60819	0,2058	0,00069
6,6666667	3,333333	1,35339	0,2646	0,00088
6,6666667	6,666667	1,35339	0,2646	0,00088

Qz= -1500 Nw

X	Y	Z	Inc -500	Inc/500
0	5	5		
5	0	0		
5	10	0		
10	5	5		
3,33	3,33	1,18	0,441	0,00088
3,33	6,67	1,18	0,441	0,00088
5	5	1,47	0,343	0,00069
6,67	3,33	1,18	0,441	0,00088
6,67	6,67	1,18	0,441	0,00088

Tabla 1 : Sistema planteadocon densidad de Fuerza 5000 N/m, Carga "Z" -1000N

Densidad de Fuerza 5000 Nw/m, Carga de Partida "Z" -3000 Nw

Qz=-3000 Nw		
Posicion nudos		
X	Y	Z
0	5	5
5	0	0
5	10	0
10	5	5
3,3333	3,333	-0,146
3,3333	6,667	-0,146
5	5	0,442
6,6667	3,333	-0,146
6,6667	6,667	-0,146

Qz=-3200			Inc 200	Inc/200
Posicion nudos				
X	Y	Z		
0	5	5		
5	0	0		
5	10	0		
10	5	5		
3,3333	3,3333	-0,32	0,1764	0,00088
3,3333	6,6667	-0,32	0,1764	0,00088
5	5	0,3	0,1372	0,00069
6,6667	3,3333	-0,32	0,1764	0,00088
6,6667	6,6667	-0,32	0,1764	0,00088

Qz=-3300			Inc 300	Inc/300
Posicion nudos				
X	Y	Z		
0	5	5		
5	0	0		
5	10	0		
10	5	5		
3,3333	3,3333	-0,41	0,2646	0,00088
3,3333	6,6667	-0,41	0,2646	0,00088
5	5	0,236	0,2058	0,00069
6,6667	3,3333	-0,41	0,2646	0,00088
6,6667	6,6667	-0,41	0,2646	0,00088

Qz=-3500			Inc 500	Inc/500
Posicion nudos				
X	Y	Z		
0	5	5		
5	0	0		
5	10	0		
10	5	5		
3,33	3,33	-0,59	0,441	0,00088
3,33	6,67	-0,59	0,441	0,00088
5	5	0,1	0,343	0,00069
6,67	3,33	-0,59	0,441	0,00088
6,67	6,67	-0,59	0,441	0,00088

Tabla 2: Sistema planteadocon densidad de Fuerza 5000 N/m, Carga "Z" -3000N

Densidad de Fuerza 7500 Nw/m, Carga de Partida "z" -3000 Nw

Qz=-3000 Nw		
Posicion nudos		
X	Y	Z
0	5	5
5	0	0
5	10	0
10	5	5
3,3333	3,333	0,736
3,3333	6,667	0,736
5	5	1,128
6,6667	3,333	0,736
6,6667	6,667	0,736

Qz=-3200			Inc 200	Inc/200
Posicion nudos				
X	Y	Z		
0	5	5		
5	0	0		
5	10	0		
10	5	5		
3,3333	3,3333	0,62	0,1176	0,00059
3,3333	6,6667	0,62	0,1176	0,00059
5	5	1,04	0,0915	0,00046
6,6667	3,3333	0,62	0,1176	0,00059
6,6667	6,6667	0,62	0,1176	0,00059

Qz=-3300			Inc 300	Inc/300
Posicion nudos				
X	Y	Z		
0	5	5		
5	0	0		
5	10	0		
10	5	5		
3,3333	3,3333	0,56	0,1764	0,00059
3,3333	6,6667	0,56	0,1764	0,00059
5	5	0,991	0,1372	0,00046
6,6667	3,3333	0,56	0,1764	0,00059
6,6667	6,6667	0,56	0,1764	0,00059

Qz=-3500			Inc 500	Inc/500
Posicion nudos				
X	Y	Z		
0	5	5		
5	0	0		
5	10	0		
10	5	5		
3,33	3,33	0,44	0,294	0,00059
3,33	6,67	0,44	0,294	0,00059
5	5	0,9	0,2287	0,00046
6,67	3,33	0,44	0,294	0,00059
6,67	6,67	0,44	0,294	0,00059

Tabla 3 : Sistema planteadocon densidad de Fuerza 75000 N/m, Carga "Z" -3000N

Para este sistema:

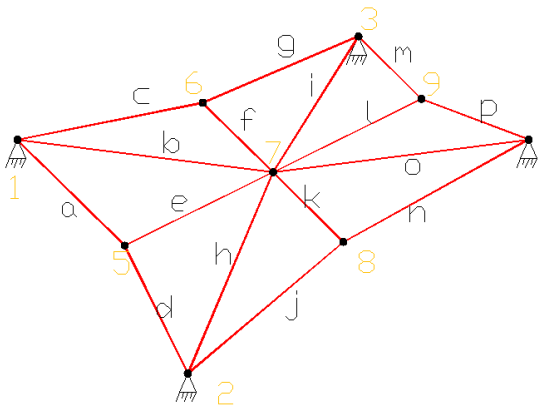


Figura 25: Sistema planteado. Análisis de linealidad

Véase que:

- Para idéntica densidad de fuerza es constante el incremento de movimiento "unitario" (incremento de movimiento/incremento de carga), independientemente de la carga de partida.
- Si aumentamos la densidad de fuerza el incremento de movimiento "unitario" se reduce, si bien es constante para dicha densidad de fuerza.
- La geometría debida a la carga de partida no afecta al incremento de desplazamiento, afectando sólo el pretensado (densidad de fuerza) de partida.

Estas conclusiones aún siendo lógicas al ser todo el proceso de cálculo lineal no son obvias, y se han querido destacar pues tienen trascendental importancia pues nos permiten:

a) Aplicar el principio de superposición una vez se han establecido las Densidades Dinámicas de Fuerza de las barras. El análisis de las fuerzas de inercia por tanto será complementario (superponible) con el estado que define la forma de equilibrio, lo que facilita el analizar de forma diferenciada ambas partes del proceso.
b) Muestra la independencia de la rigidez respecto del estado de cargas inicial, dependiendo la respuesta pues sólo del grado de pretensado del sistema y de las propiedades de los vínculos.

Merece la pena insistir en que el proceso de cálculo de las Densidades Dinámicas de Fuerza (DDF) contiene implícito el pretensado del sistema, pues dicha magnitud depende de la geometría resultante y del proceso de dimensionado en base a las cargas de equilibrio.

Igualmente recordemos que dadas unas Densidades de Fuerza la singular geometría de los apoyos se traducía en cargas equivalentes por la expuesta condensación de apoyos, es decir equivaldría a un estado inicial de cargas, no influyendo por tanto la geometría de los apoyos en la rigidez dinámica del sistema que será idéntica en X,Y,Z. (como ha de ser para que el movimiento sea sincronizado en las tres direcciones) en este nivel de desarrollo del programa *.

(* Podría darse el caso de apoyos que restrinjan sólo una o dos direcciones, caso analizable sobre el mismo fundamento expuesto, pero que se deja para posteriores líneas de desarrollo del programa y método, pues aún siendo idéntica la matemática necesaria se alteraría la configuración de las matrices y por ende la formalización de las expresiones matriciales que en este momento son idénticas para X,Y,Z. Esto se expondrá más detenidamente en las líneas de desarrollo del programa)

Tenemos pues una matriz [K] de rigidez “tradicional” que nos permitirá calcular las fuerzas elásticas del sistema al desviarse éste de su posición de equilibrio, por lo que llegados a este punto podríamos formular el análisis dinámico clásico para múltiples grados de libertad si asignáramos masa a los nudos del sistema, dado que las Fuerzas de inercia (Fi) y Fuerzas elásticas (Fe) se expresan según:

$$(F_{iner})=[M](Ac)=-W2*[M](X)$$
$$(F_{elas})=[K](X)$$

(am.3)

En un sistema discretizado en base a nudos y barras la masa provendrá por un lado de las cargas gravitatorias aplicadas sobre los nudos (cuyo peso se dividirá por la aceleración de la gravedad) y por otro del peso propio, que será un valor dependiente de la geometría del sistema (longitud de las barras), de la sección de las mismas y de la densidad de los materiales. La masa total resultante se aplicará en los nudos por ser estos los puntos de aplicación de las fuerzas (de inercia en este caso), resultando de su composición una matriz diagonal [M] al ser la masa de cada nudo independiente del resto. A cada nudo corresponderá una masa total que será la suma de la masa directamente aplicada más la mitad de la masa de cada barra que llega a dicho nudo

Para ello el proceso es el siguiente:

1) Conocida la forma resultante $[F]_{x,y,z}$ se calcularán las proyecciones “L” en X,Y,Z de las barras haciendo uso de la matriz de relaciones [G]:

$$[G]^* [F]_{x,y,z} = [L]_{x,y,z}$$

(Av.1)

G: Matriz de relaciones

BN	1	2	3	4	5	6	7	8	9
a	1	0	0	0	-1	0	0	0	0
b	1	0	0	0	0	0	-1	0	0
c	1	0	0	0	0	-1	0	0	0
d	0	1	0	0	-1	0	0	0	0
e	0	0	0	0	1	0	-1	0	0
f	0	0	0	0	0	1	-1	0	0
g	0	0	1	0	0	-1	0	0	0
h	0	1	0	0	0	0	-1	0	0
i	0	0	1	0	0	0	-1	0	0
j	0	1	0	0	0	0	0	-1	0
k	0	0	0	0	0	0	1	-1	0
l	0	0	0	0	0	0	1	0	-1
m	0	0	1	0	0	0	0	0	-1
n	0	0	0	1	0	0	0	-1	0
o	0	0	0	1	0	0	-1	0	0
p	0	0	0	1	0	0	0	0	-1

X

Posicion nudos		
X	Y	Z
0	5	5
5	0	0
5	10	0
10	5	5
3,33	3,33	1,62
3,33	6,67	1,62
5	5	1,81
6,67	3,33	1,62
6,67	6,67	1,62

=

Lx	Ly	Lz	Long
-3,3	1,67	3,382	5,033
-5	0	3,186	5,929
-3,3	-1,7	3,382	5,033
1,67	-3,3	-1,62	4,063
-1,7	-1,7	-0,2	2,365
-1,7	1,67	-0,2	2,365
1,67	3,33	-1,62	4,063
0	-5	-1,81	5,319
0	5	-1,81	5,319
-1,7	-3,3	-1,62	4,063
-1,7	1,67	0,196	2,365
-1,7	-1,7	0,196	2,365
-1,7	3,33	-1,62	4,063
3,33	1,67	3,382	5,033
5	0	3,186	5,929
3,33	-1,7	3,382	5,033

Figura 26: Cálculo de las longitudes de las barras en posición de equilibrio

2) Conocidas las proyecciones de las barras se calculará la longitud y masa para cada una de éstas:

$$\text{Longitud} = \text{módulo} = \text{Raiz}\,(Lx*Lx + Ly*Ly + Lz*Lz)$$
$$\text{Masa} = \text{Densidad} * \text{Longitud} * \text{Sección} * \text{Densidad}$$

(Av.2)

Si dispusiéramos el producto de la "Densidad"X"Sección" en la diagonal de una matriz cuadrada de dimensión el "número de barras" [DS] y pusiéramos las longitudes según se han obtenido ordenadamente en un vector (L) la operación anterior se expresaría de la siguiente forma:

$$[DS]*(L)$$

(Av.3)

3) La matriz [Gt] (traspuesta de la matriz G de vínculos) tiene la representatividad de manifestar agrupadamente por filas los vínculos que acometen a una barra a la hora de plantear un equilibrio de fuerzas (véase formulación expresada en am.6) . En este caso lo que deseamos no es establecer un equilibrio de fuerzas sino manifestar la suma positiva de la mitad de la masa de las barras que acometen a un nudo en estudio, a efectos de determinar la masa agregada que le corresponde. Por tanto lo que habría que hacer es:

- Crear una matriz equivalente a la Gt pero con todos sus elementos positivos (convirtiendo en "1" todo valor que fuera "-1", llamemos a esa matriz "positivada" [+Gt]
- Pos multiplicar dicha matriz por ½ del paso anterior.

1/2*[DS]*(L)

[+Gt]

Nudos	Barras a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p			(M)
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0		0,013	
2	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0		0,015	0,02
3	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0		0,013	0,01
4	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1		0,005	0,01
5	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	X	0,002	0,02
6	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0		0,002	= 0,01
7	0	1	0	0	1	1	0	1	1	0	1	1	0	0	1	0		0,005	0,01
8	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0		0,010	0,02
9	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1		0,010	0,01
																		0,005	0,01
																		0,002	0,03
																		0,002	0,01
																		0,005	0,01
																		0,013	
																		0,015	
																		0,013	

Las masas están en Kg bajo la acción de la gravedad, por tanto para hacerlas masas c

(Av.5)

Figura 27: Planteamiento del sistema para la obtención de la matriz de masas asociada a las barras

4) El resultado de esa operación será un vector (M) del cual ya se podrá obtener la pretendida matriz de masas [M]. Para ello por un lado tomamos las masa de las barras del sistema sobre los nudos – (M) según se ha descrito en el paso anterior- y a ese valor añadimos las cargas gravitatorias en "Z" (Z-) que estén aplicadas directamente sobre el nudo divididas por 9.8 . Obtendríamos ya el vector de masas totales sobre los nudos (Mt), a partir del cual disponiendo sus elementos en la diagonal de una matriz cuadrada -de dimensión el número de nudos- obtendríamos la anhelada matriz [M]

$$(Mt)=1/2*[+Gt]*[DS]*(L)+(Z-)/9.8= (M)+(Z-)/9.8$$
$$(Mt) \rightarrow [M]$$

(Av.6)

Obtenida la matriz de masas [M] y la matriz de rigidez dinámica $[Pdin]_{DDF}$ ya tendremos por fin los elementos necesarios para plantear el equilibrio en el tiempo según la Dinámica Clásica de los sistemas estructurales (véase formulaciones dc.1 en adelante). Hemos de hacer notar que el proceso anteriormente descrito para la obtención de las masas es el proceso general que determina la masa de todos lo nudos. Sin embargo tendremos que quedarnos sólo con la parte dinámica del sistema, es decir hemos de tomar la parte que corresponde SÓLO con nudos móviles, (coherentemente con$[Pdin]_{DDF}$). Para ello eliminamos simplemente los nudos que son apoyos.

Retomando la dinámica clásica de los sistemas estructurales y haciendo uso de las expresiones halladas, si planteamos el equilibrio en el tiempo entre las fuerzas de inercia y las fuerzas elásticas tenemos:

$$F_{iner}+F_{elas}=-W2*[M](X)+ [Pdin]_{DDF} *(X)=0$$
$$(-W2[M]+ [Pdin]_{DDF})(X)=0$$
$$(-W2+[M-1]* [Pdin]_{DDF})(X)=0$$

(según am4)

Lo que conduce a la siguiente igualdad matricial:

$$[M-1]*[Pdin]_{DDF*}(X)=W2*(X)$$

(según am.5)

Los vectores (X) que verifiquen la anterior ecuación serán los que permitan satisfacer la igualdad (am.4), por lo que el cuadrado de la velocidad angular del sistema (W2) coincidirá con los autovalores de la matriz $[M-1]^* [Pdin]_{DDF}$ siendo sus autovectores (X) las correspondientes formas modales. La resolución de dicha expresión constituye *el planteamiento del problema de autovalores*.

$$[FM] = \text{Autovectores de } ([M-1]^* [Pdin]_{DDF})$$
$$(Fr) = \text{Autovalores de } ([M-1]^* [Pdin]_{DDF})$$

(Av.7)

Cada forma modal representará un vector cuyas coordenadas están en una escala "neutra" (habrá que "proporcionarlo" como veremos en el análisis del movimiento), valores que se agregarán a las coordenadas de equilibrio (X,Y,Z) de la forma estructural de partida, siendo pues valores incrementales conforme al principio de superposición anteriormente descrito. Habrá tantas formas modales como grados de libertad (se trata de un sistema Linealmente Idenpendiente), si bien las frecuencias de oscilación SI podrán ser coincidentes entre formas modales en algunos casos. El resultado de la resolución del problema de autovalores será pues:

- Una Matriz de cuadrada de dimensión el número de nudos móviles, en la que cada columna es una forma de oscilación: [FM]
- Un vector de dimensión el número de nudos móviles con las frecuencias asociadas a cada forma de oscilación (Fr)

(Av.8)

Expuesto esto véase un hecho trascendental: la inercia está asociada a una masa que es única, si bien el movimiento puede ser en X,Y,Z, (importante: en relación a la dimensión de la estructura) por lo que podrá haber fuerzas de inercia en las tres direcciones respecto de la rigidez. La rigidez que equilibra al sistema es única pero debe lograr el equilibrio entre las tres direcciones (pues las barras son las mismas para el movimiento en X, en Y o en Z). Si el movimiento y la rigidez de la estructura coincidieran en una misma dirección toda la rigidez estaría "focalizada" a equilibrar dicha fuerza unidireccionalmente, pero si por el contrario la dirección del movimiento "contrasta" con la rigidez tridimensional de la estructura ésta deberá repartirse de forma proporcionada a cada dirección. Como la masa y la rigidez son tensores lineales en ambos casos el factor de proporcionalidad que atienda a esta circunstancia podría aplicarse a cualquiera de los dos términos, ya sea aumentando la masa en su conjunto

(si se analiza agrupadamente el movimiento), ya sea disminuyendo la rigidez disponible (si se analiza el movimiento en cada dirección). Entendemos matemática y físicamente más sencillo y visual el aplicar dicho factor *a la masa* con un **factor de masas** *"Fm".* Así un movimiento que fuera unidimensional en relación a la estructura tendría que calcularse con un Fm= 1 (por ejemplo el movimiento en su eje de varias masas enlazadas linealmente entre si), un movimiento bidireccional (en relación a la estructura) de idéntica amplitud en cada dirección tendría un Fm= 2 (por ejemplo el movimiento en su plano de una malla plana) y por ultimo el idéntico caso en las direcciones X,Y,Z tendría que calcularse con un Fm= 3. (El caso generalizado de movimientos de amplitudes diferentes en X,Y,Z se analizará más adelante cuando se trate en detalle el factor de masas).

Así el problema generalizado a X,Y,Z se plantearía agrupadamente introduciendo el factor de masas "Fm" según:

$$(1/Fm)*[M-1]*[Pdin]_{DDF*}(X)=W2*(X) \quad \text{(Av.9)}$$

Véase que esas "formas naturales" (modales) expresadas por el Autovector (X) son formas múltiples de la matriz $[M-1]\,[Pdin]_{DDF}$, es decir con un símil "gráfico" son "ondas/formas estacionarias" implícitas a la matriz (que expresa la relación entre rigidez y masa) que encajan armónicamente en dicha relación entre masa y rigidez. Las formas modales NO dependen del factor de masas pero las frecuencias fundamentales SI dependen de él. En cualquier caso el Factor de Masas se tratará más detenidamente cuando se analice la composición tridimensional del movimiento.

Por último es muy importante destacar un hecho que atañe directamente a los límites de validez del método: la relación anteriormente mostrada se basa en un equilibrio nodo a nodo establecido sobre compatibilidades elásticas y geométricas. Es decir, los movimientos amplios en los que se da un equilibrio tipo "pendular" (donde la gravedad es una fuerza estabilizadora) o aquellos equilibrios "predominantemente geométrico" (en los que poco interviene la elasticidad de los componentes, siendo la variación de tensión en las barras escasa, como el análisis básico de una masa oscilando en una cuerda a tensión constante) no son en ocasiones adecuadamente interpretados o incluso modelados con esta metodología. Por tanto las situaciones dinámicas coherentes con la metodología propuesta serán preferentemente aquellas formas ANTICLÁSTICAS (o muy poco sinclásticas) en las que predomine la respuesta según la compatibilidad descrita entre los elementos dentro de un rango de amplitud de

movimientos coherente con las propiedades de las barras y su grado de pretensado. Más adelante se tratarán los límites de validez del método con mayor profundidad.

Planteamiento del problema energético. Verificación.

Para una tenso estructura en equilibrio y conocido un modo propio de oscilación podríamos obtener la energía potencial asociada a ese movimiento, sumando para todas las barras su energía potencial elástica cuando la amplitud de oscilación sea máxima (por tanto velocidad nula, energía cinética nula y aceleración máxima).

En un sistema conservativo la energía potencial se habrá de mantener constante en todo momento, por lo que cuando la amplitud de oscilación sea nula (por tanto velocidad máxima, energía potencial nula y aceleración nula) la energía cinética máxima habrá de coincidir con la energía potencial.

Según vimos en el apartado de fundamentos tenemos:

$$Ec_{(max)} = Ep_{(max)}$$
$$0.5*M*W*W*U_{(max)}*U_{(max)} = 0.5*Kb*U_{(max)}*U_{(max)}$$
$$W*W = [Kb*U_{(max)}*U_{(max)}]/[M*U_{(max)}*U_{(max)}]$$

(pe.8)

Entrando de lleno tenemos que la energía potencial elástica de una barra “Ep,n“ se define según:

$$E_{p,n} = 1/2*K*(\text{inc}L_{,n})^2$$
$$E_{p,n} = 1/2*DDF_{,n}*(\text{inc}L_{,n})^2$$

(ve.1)

Donde K es la rigidez de la barra según la ley de Hooke, cuyo valor aplicado a nuestro caso será la Densidad Dinámica de Fueza (DDF).

Véase que la verificación pretendida concierne a un modo propio de oscilación, por ello se podrá afirmar que el movimiento en toda la estructura estará sincronizado, dado

que los modos propios son por definición las figuras básicas de oscilación a partir de las cuales se compone toda posible vibración de la estructura. Así el máximo de amplitud establecido para cada nudo por el autovector (X_i) se alcanzará simultáneamente en todos los puntos móviles.

Según vimos en la ecuación (Av.1) establecida una forma (F) el incremento de coordenadas asociado a cada barra es calculable matricialmente según:

$$[G]^* (F) = (\text{inc}L)$$

Lo cual se aplicará usando la forma de oscilación definida por el autovector (X_i) en su punto de máxima amplitud:

$$[G]^* (X_i) = (\text{inc}L)_i \qquad \text{(ve.2)}$$

Obteniéndose de este modo los incrementos de longitud para cada barra.

Véase que la rigidez del sistema se constituye por barras que **son únicas** y que aportan axialmente la rigidez X,Y,Z por igual, por lo que el planteamiento de la energía potencial habrá de ser adireccional. La energía potencial elástica para el total de la estructura cuando oscila según la forma canónica “i” será:

$$[G]^* (X_i) = (\text{inc}L)_i$$
$$(L2)_i=(\text{inc}L^* \text{inc}L)_i$$
$$E_{p,i}=1/2^*[DDF]^*(L2)_i$$

(L2) es un vector de idéntica dimensión que (incL) que parte de éste con sus componentes al cuadrado (ve.3)

Por otra parte la energía cinética del sistema para un nudo “j” será:

$$E_{c,j} = ½^*M_j^*V_j^*V_j \qquad \text{(ve.4)}$$

Donde la velocidad V_j es de la derivada en el tiempo del vector de posición supuesta la oscilación sinusoidal. Como sólo nos interesa su máximo sabemos que dicho valor es W^*X_i, donde W es la velocidad angular y (X_i) son las coordenadas máximas del vector

de posición, es decir, las coordenadas del autovector asociado al modo de vibración "i". Por tanto para el nudo "j" del modo de vibración "i" tenemos:

$$V_{i,j} = W_i * X_{i,j}$$

$$V_{i,j} * V_{i,j} = W_i * W_i * X_{i,j} * X_{i,j}$$

$$(V_i 2) = W_i 2 * (X_i 2) \text{ Generalizado a vector}$$

$$E_{c,i} = ½ [M] * (V_i 2) = ½ W_i 2 * [M] * (X_i 2)$$

(ve,5)

En la anterior relación aún no se considera un tema importante que habrá de introducirse: en un movimiento en el espacio la masa y la rigidez son únicas, pero éstas atienden a razones físicas distintas. Las barras aportan una rigidez que es única para las tres direcciones del espacio, pero por el contrario las fuerzas de inercia se movilizan en X,Y,Z. Atendiendo a esto si analizamos la energía cinética en cada dirección tendríamos que aplicar en una masa en cada eje. Aunque la composición de movimiento será objeto de posterior estudio, si la oscilación tuviera idéntica amplitud en las tres direcciones la masa estaría triplicada en relación a la rigidez del sistema:

$$E_{p,i} = 1/2 * [DDF] * (L2)_i$$

$$E_{c,i} = \mathbf{3} * ½ [M] * (V_i 2) = \mathbf{3} * ½ W_i 2 * [M] * (X_i 2)$$

$$\mathbf{3} * W_i 2 * [M] * (X_i 2) = [DDF] * (L2)_i$$

$$W_i 2 = (1/\mathbf{3}) * ([DDF] * (L2)_i) / ([M] * (X_i 2))$$

(ve,6)

La expresión anterior generalizándose para movimientos que pudieran tener diferente dimensionalidad de oscilación respecto a X,Y,Z será:

$$W_i 2 = (1/\mathbf{FM}) * ([DDF] * (L2)_i) / ([M] * (X_i 2))$$

(ve,7)

Donde "*Fm*" será el factor de masas que se desarrollará con mayor profundidad a posteriori.

De esta forma establecido un Factor de Masas para el autovector en estudio deduciremos la velocidad angular de oscilación al cuadrado "$W_i 2$" , valor que habrá de coincidir con el autovalor asociado a ese autovector (que se habrá deducido del análisis de modal haciendo uso del mismo Factor de Masas). Si esta condición se satisface la verificación será a priori correcta.

La verificación energética mostrada permite una programación informática sencilla por lo que es el método de verificación integrado en el software asociado a la Tesis (la verificación computacional se hace por defecto con factor de masas 1). No obstante los primeros planteamientos de verificación que se formularon se basaron en el teorema de Rayleigh Ritz, dado que es una verificación "paso a paso", sin simplificación ni suposición alguna (no aplica principio de superposición, trabaja con longitudes de barras completas….) fundamentada en que en todo momento (y por tanto en parte de un ciclo o en un ciclo completo de oscilación) la suma del trabajo de las fuerzas internas y externas habrá de ser cero:

- Trabajo fuerzas externas:

$$\text{Fuerzas externas de inercia= } F_{ext} = \text{Masa*Aceleración}$$

$$\text{Aceleración} = A_{c(t)} = -W*W*U_{(t)}$$

$$W_{ext} = F_{ext(t)}*\text{Desplazamiento}_{(t)} = -M*(W*W*U_{(t)})*U_{(t)}$$

M=Masa; W= Velocidad angular de oscilación; U(t)= desplazamiento

(pe.3)

- Trabajo de las fuerzas internas:

$$Fb_{(t)} = Kb* \text{incL}_{(t)}$$

$$\text{Wint} = \text{Fuerza}_{(t)}*\text{Desplazamiento}_{(t)} = Kb* \text{incL} * \text{incL}$$

(pe.4)

- Para una oscilación ("t" entre 0 y T) el balance de ambos trabajos deberán ser nulo:

$$\text{Integral}[W_{ext}+W_{int}]=0$$

$$\text{Integral}[M*(W*W*U_{(t)})*U_{(t)}]dt = \text{Integral}[Kb* \text{incL} * \text{incL}]dt$$

$$W*W = \text{Int}[Kb* \text{incL} * \text{incL}]dt / \text{Int}[M*(W*W*U_{(t)})*U_{(t)}]dt$$

(pe.5)

Así para la verificación según Rayleigh Ritz se analiza un ciclo de oscilación, que se discretiza en pasos infinitesimales para cada uno de los cuales se calcula el trabajo externo de las fuerzas actuantes y el interno de la elasticidad del sistema. De la relación entre ambos se obtiene la velocidad angular de rotación al cuadrado, que habrá de coincidir con el valor calculado tanto por el método "simplificado" como por el "análisis modal". A continuación se muestra el estudio para la estructura que venimos analizando:

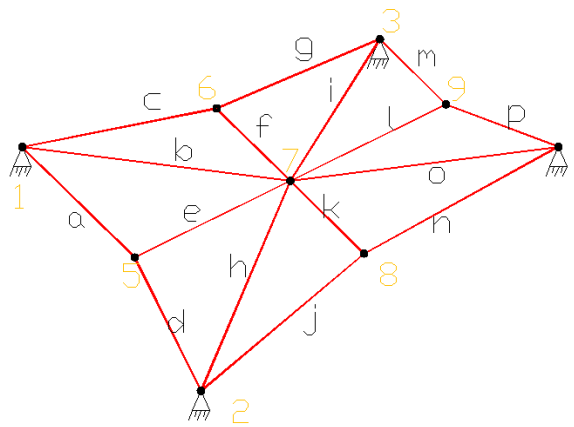


Figura 28: Sistema para el análisis de frecuencias por método energético

a) Resolución del problema de autovalores (Densidad de fuerza 7500, Masas -1500 NW, FM=1):

Análisis modal del sistema

Rigidez de los nudos móviles=relación entre fuerza y desplazamiento

Véase que G sólo depende de la densidad de fuerza, por lo que dicha densidad de fuerza deberá ser acorde al sistema

K:Relación entre fuerza y desplazamiento

NW	5	6	7	8	9
5	22500	0	-7500	0	0
6	0	22500	-7500	0	0
7	-7500	-7500	60000	-7500	-7500
8	0	0	-7500	22500	0
9	0	0	-7500	0	22500

Posición inicial

	X	Y	Z
	3,33	3,33	1,62
	3,33	6,67	1,62
	5,00	5,00	1,81
	6,67	3,33	1,62
	6,67	6,67	1,62

M:Masas de inercia (peso dividido por gravedad)

Unidad: Kg*S^2/m Fm= 1

NW	5	6	7	8	9
5	153,1	0	0	0	0
6	0	153,06	0	0	0
7	0	0	153,1	0	0
8	0	0	0	153,1	0
9	0	0	0	0	153,1

Verificación de modos

1	Fz de inercia	Frza elástica	Cociente
M*Inc	w2*M*inc	K*Inc	% error
-72,21624	-8133,0	-8133,2	0,002
-72,21624	-8133,0	-8133,2	0,002
-50,66574	-5706,0	-5706,3	0,006
-72,21624	-8133,0	-8133,2	0,002
-72,21624	-8133,0	-8133,2	0,002

2	Fz de inercia	Frza elástica	Cociente
M*Inc	w2*M*inc	K*Inc	% error
9,88613	1453,3	1453,253	0,001
75,40606	11084,7	11084,63	0,001
2,44E-14	0,0	0,0075	-100,000
-126,3329	-18570,9	-18570,83	0,001
41,04055	6033,0	6032,925	0,001

M-1: Inversa de masas móviles

NW	5	6	7	8	9
5	0,007	0	0	0	0
6	0	0,0065	0	0	0
7	0	0	0,007	0	0
8	0	0	0	0,007	0
9	0	0	0	0	0,007

M-1xK: Matriz a diagonalizar

NW	5	6	7	8	9	Autovalores				
						112,62	147	147	147	426,38
5	147	0	-49	0	0	-4,72E-01	6,46E-02	6,80E-01	5,32E-01	1,66E-01
6	0	147	-49	0	0	-4,72E-01	4,93E-01	-6,47E-01	2,97E-01	1,66E-01
7	-49	-49	392	-49	-49	-3,31E-01	1,59E-16	3,80E-17	7,57E-17	-9,44E-01
8	0	0	-49	147	0	-4,72E-01	-8,25E-01	-2,59E-01	-3,80E-02	1,66E-01
9	0	0	-49	0	147	-4,72E-01	2,68E-01	2,27E-01	-7,92E-01	1,66E-01
				W=		10,61	12,12	12,12	12,12	20,65 Rad/s
				T=		0,59	0,52	0,52	0,52	0,30 Sec
				Frec=		1,69	1,93	1,93	1,93	3,29 Hz

Tabla 4: Determinación de frecuencias por análisis modal

W=10.61 rad/s

b) Resolución simplificada del sistema (análisis del modo 1):

Planteamiento simplificado

Energía cinética "Fm" = 1

Nudo	"Xi"	"Xi2"	[M]*(Xi2)	FM*½*[M]*(Xi2)
1	0	0	0	0,00
2	0	0	0	0,00
3	0	0	0	0,00
4	0	0	0	0,00
5	-4,72E-01	0,222604676	34,07234565	17,04
6	-4,72E-01	0,222604676	34,07234565	17,04
7	-3,31E-01	0,10956762	16,770866	8,39
8	-4,72E-01	0,222604676	34,07234565	17,04
9	-4,72E-01	0,222604676	34,07234565	17,04
			Energia cinética/W2=	76,55

Energía potencial elástica

Barra	[G]* (Xi)= (incL)i	(L2)=(incL* incL)i	Epi=1/2*[DDF]*(L2)i
1	4,72E-01	1,11E-01	834,7675354
2	3,31E-01	5,48E-02	410,8785754
3	0,47181	1,11E-01	834,7675354
4	0,47181	1,11E-01	834,7675354
5	-0,1408	9,91E-03	74,3424
6	-0,1408	9,91E-03	74,3424
7	0,47181	1,11E-01	834,7675354
8	0,33101	5,48E-02	410,8785754
9	0,33101	5,48E-02	410,8785754
10	0,47181	1,11E-01	834,7675354
11	0,1408	9,91E-03	74,3424
12	0,1408	9,91E-03	74,3424
13	0,47181	1,11E-01	834,7675354
14	0,47181	1,11E-01	834,7675354
15	0,33101	5,48E-02	410,8785754
16	0,47181	1,11E-01	834,7675354
	Energía Potencial elástica:		8619,024185

W2=	112,59
W=	10,61

Tabla 5: Análisis energético de frecuencias simplificado (Hamilton)

W=10.61 rad/s

c) Resolución de sistema paso a paso aplicando una discretización basada en Rayleigh Ritz. El ciclo se discretiza en 1000 divisiones

Análisis de Modo 1

Coordenadas Equilibrio

Coordenadas punto equilbrio

	X	Y	Z
1	0,00	5,00	5,00
2	5,00	0,00	0,00
3	5,00	10,00	0,00
4	10,00	5,00	5,00
5	3,33	3,33	1,62
6	3,33	6,67	1,62
7	5,00	5,00	1,81
8	6,67	3,33	1,62
9	6,67	6,67	1,62

Empezar n (poner "s")

Divisiones 1000

N 1 División Naux 2

W 10,61 Rad/seg

T 0,59 Segundos

Fase X 0 Grados respecto a Z

Fase Y 0 Grados respecto a Z

Oscilador "Xi"	X	Y	Z	X	Y	Z
	Paso N			Paso N-1		
-0,47181	-0,003	-0,003	-0,003	0,000	0,000	0,000
-0,47181	-0,003	-0,003	-0,003	0,000	0,000	0,000
-0,33101	-0,002	-0,002	-0,002	0,000	0,000	0,000
-0,47181	-0,003	-0,003	-0,003	0,000	0,000	0,000
-0,47181	-0,003	-0,003	-0,003	0,000	0,000	0,000

Tabla 6: Análisis energético de frecuencias según Rayleigh Ritz. Paso 1

Análisis paso a paso del trabajo interno del sistema, obteniéndose un trabajo absoluto acumulado al final del ciclo de 34.600 Julios = 4*8.640J:

Análisis del trabajo interno asociado al paso N

N: 1000

	Lon o	X	Y	Z	Long N	Inc L(N)	Inc Fuerza	X	Y	Z	Long N-1	Inc Paso	W(Fxd)
a	5,033	-3,33	1,67	3,38	5,03	0,000	0,00	-3,34	1,66	3,38	5,03	0,0010	1,59E-06
b	5,929	-5,00	0,00	3,19	5,93	0,000	0,00	-5,00	0,00	3,18	5,93	-0,0006	6,34E-07
c	5,033	-3,33	-1,67	3,38	5,03	0,000	0,00	-3,34	-1,67	3,38	5,03	-0,0010	1,42E-06
d	4,063	1,67	-3,33	-1,62	4,06	0,000	0,00	1,66	-3,34	-1,62	4,07	-0,0024	8,97E-06
e	2,365	-1,67	-1,67	-0,20	2,37	0,000	0,00	-1,67	-1,67	-0,20	2,36	0,0013	2,72E-06
f	2,365	-1,67	1,67	-0,20	2,37	0,000	0,00	-1,67	1,67	-0,20	2,37	0,0001	8,32E-09
g	4,063	1,67	3,33	-1,62	4,06	0,000	0,00	1,66	3,33	-1,62	4,06	0,0025	9,50E-06
h	5,319	0,00	-5,00	-1,81	5,32	0,000	0,00	0,00	-5,00	-1,82	5,32	-0,0027	1,11E-05
i	5,319	0,00	5,00	-1,81	5,32	0,000	0,00	0,00	5,00	-1,82	5,32	0,0012	2,42E-06
j	4,063	-1,67	-3,33	-1,62	4,06	0,000	0,01	-1,67	-3,34	-1,62	4,07	-0,0048	3,64E-05
k	2,365	-1,67	1,67	0,20	2,37	0,000	0,00	-1,67	1,67	0,20	2,37	0,0001	8,32E-09
l	2,365	-1,67	-1,67	0,20	2,37	0,000	0,00	-1,67	-1,67	0,20	2,37	-0,0012	2,15E-06
m	4,063	-1,67	3,33	-1,62	4,06	0,000	0,00	-1,67	3,33	-1,62	4,06	0,0000	1,90E-09
n	5,033	3,33	1,67	3,38	5,03	0,000	-0,01	3,33	1,66	3,38	5,03	0,0049	3,80E-05
o	5,929	5,00	0,00	3,19	5,93	0,000	0,00	5,00	0,00	3,18	5,93	0,0029	1,29E-05
p	5,033	3,33	-1,67	3,38	5,03	0,000	0,00	3,33	-1,67	3,38	5,03	0,0030	1,38E-05

W	1,42E-04
Acum=	3,46E+04
Poten	8,64E+03

Waux	34569,43211

Tabla 7: Análisis energético de frecuencias según Rayleigh Ritz. Paso 2

Análisis paso a paso del trabajo de las fuerzas externas del sistema:

lisis del trabajo externo (fuerzas de inercia)

1000

	Ax	Ay	Az	A abs	Inc X	Inc Y	Inc Z	Inc abs	Masa	Tx/W	Ty/W	Tz/W	Tabs/W
5	0,000	0,000	0,000	0,000	-0,003	-0,003	-0,003	0,0051346	153,062	2,79843E-07	2,7984E-07	2,7984E-07	8,3953E-07
6	0,000	0,000	0,000	0,000	-0,003	-0,003	-0,003	0,0051346	153,062	2,79843E-07	2,7984E-07	2,7984E-07	8,3953E-07
7	0,000	0,000	0,000	0,000	-0,002	-0,002	-0,002	0,0036023	153,064	1,37742E-07	1,3774E-07	1,3774E-07	4,1323E-07
8	0,000	0,000	0,000	0,000	-0,003	-0,003	-0,003	0,0051346	153,062	2,79843E-07	2,7984E-07	2,7984E-07	8,3953E-07
9	0,000	0,000	0,000	0,000	-0,003	-0,003	-0,003	0,0051346	153,062	2,79843E-07	2,7984E-07	2,7984E-07	8,3953E-07
									suma	1,25711E-06	1,2571E-06	1,2571E-06	3,7713E-06
									Waux	306,1205075	306,120507	306,120507	918,361522
									Acum	306,1205087	306,120509	306,120509	918,361526
									Cociente	1,13E+02	1,13E+02	1,13E+02	3,76E+01
									Factor	1,00E+00	1,00E+00	1,00E+00	3,34E-01
									W	10,62673646	10,6267365	10,6267365	6,13534916

Tabla 8: Análisis energético de frecuencias según Rayleigh Ritz. Paso 3

Resultando Ec/W2=306.12=4*76.53.

W=10.62 rad/s

Podemos apreciar pues que la correlación entre los métodos de cálculo es formidable.

En la anterior tabla se ha analizado el trabajo por componentes (separado X,Y,Z) pero además el movimiento combinado de idéntica amplitud en X,Y,Z. En ese caso la frecuencia que resulta es W= 6.1353 Rad/s

Véase que la relación entre ambas frecuencias 10.62/6.1353=1.732, valor que coincide exactamente con raíz de 3, es decir la relación entre los cuadrados de las frecuencias entre el movimiento unidireccional y el tridimensional de idéntica amplitud es 3, como ya precedía el Factor de Masas.

El análisis según Rayleigh Ritz muestra la aplicación directa de los principios físicos de la conservación de la energía sin simplificaciones de ningún tipo, trabajándose sobre las longitudes totales de las barras y sin aplicarse principio alguno de superposición, lo que avala los pasos dados a lo largo de todo el proceso.

Obtención de frecuencias y modos propios. Composición del fenómeno por direcciones.

Del análisis modal planteado sobre las matrices de masas y rigidez dinámica que caracterizan las tenso estructuras se obtendrán los autovalores y autovectores, estableciéndose las formas fundamentales de oscilación. A partir de dichas figuras se realizará la verificación energética de los resultados (de manera simplificada o según Rayleigh Ritz) restando para la completa caracterización dinámica del sistema la composición tridimensional del movimiento.

El inicio de una oscilación requiere de una acción, que podrá ser puntual, transitoria o permanente (según su duración); periódica o aleatoria (según su pauta de excitación); y que podrá ser aplicada según una dirección constante o variable… el rango pues de frentes de excitación será infinito, como lo será igualmente la adaptada respuesta de la estructura en cada caso. Obviándose el amortiguamiento toda vibración libre de una estructura a partir de una situación de partida será combinación de unas formas básicas (canónicas) de oscilación llamados modos propios, de tal modo que el movimiento total se compondrá como suma de dichas formas básicas, oscilando cada uno de los modos en torno a la posición de equilibrio según su frecuencia natural y conforme la amplitud establecida por las condiciones de partida. Toda oscilación forzada tendrá generalmente una respuesta acompasada con la acción causante, desfasada de la misma un cierto ángulo de fase y con una amplitud que será objeto de estudio, pues si la fuente de oscilación tiende a acompasarse con algún modo propio (ya sea por su pauta característica o a causa de la interacción con la propia estructura, como es típico de los fenómenos aeroelásticos) entonces la respuesta podrá amplificarse. Es el conocido fenómeno de la resonancia.

Las formas modales de oscilación de las tenso estructuras según la metodología de análisis propuesta presentan una singularidad respecto de las estructuras convencionales: su rigidez a efectos dinámicos debe plantearse de forma común para X,Y,Z pues en cada barra un único mecanismo axil atiende la respuesta en los tres ejes por igual. Por ello si la fuente de excitación tiende a generar una respuesta alineada con la rigidez de la estructura (problema unidimensional) la frecuencia fundamental tenderá a ser máxima (pues habrá una mínima masa inercial asociada a la rigidez); y por el contrario si el movimiento tiende a ser tridimensional respecto de la rigidez de la estructura la frecuencia fundamental tenderá a ser mínima (pues será máxima la masa inercial asociada a la rigidez común). Por tanto cada modo propio de oscilación tendrá asociado un rango de frecuencias que dependerá de la "dimensionalidad" del movimiento en relación a la estructura y de la fuente de excitación (en el caso de que la excitación sea forzada). El parámetro ajusta las frecuencias a dichas circunstancias es el factor de masas "Fm".

Por tanto a la hora de asignar un valor al Factor de Masas habrá que atender a la forma de excitación y la dimensionalidad del movimiento en relación a la estructura.

Forma de excitación: tradicionalmente el comportamiento dinámico de un oscilador se asemeja a una función sinusoidal según una amplitud "A", una velocidad angular "W" y una fase "fi"

$$U_{(t)} = A*sen(W*t+ fi)$$

Derivando respecto del tiempo tendríamos respectivamente la velocidad V(t) y la aceleración A(t):

$$V_{e(t)} = -A*W*cos(W*t+fi)$$
$$A_{c(t)} = -A*W2*sen(W*t+fi)$$

Por ser sinusoides oscilarán del +/- 1 pasando por el cero, por lo que el máximo en valor absoluto será:

$$U_{(max)} = A$$
$$V_{e(max)} = A*W$$
$$A_{c(max)} = A*W2$$

Por tanto para una frecuencia dada la fuerza de inercia será proporcional a la amplitud del movimiento.

Un movimiento podría tener amplitudes diferentes en X,Y,Z siempre que el movimiento en cada dirección esté acompasado a una frecuencia dada. Como las fuerzas de inercia "baten" la rigidez del sistema proporcionalmente a la amplitud en cada dirección, la "masa dinámica" asociada a dicho modo de vibración tendrá que ajustarse a ello para que el movimiento sea síncrono, lo que significa que la masa dinámica tendría que corregirse según:

$$Fm = (3*\text{Máxima Amplitud}_{x,y,z})/ (A_x+A_y+A_z) \quad \text{(co.1)}$$

La tenso estructura por su naturaleza (rigidez única) puede adaptar su frecuencia fundamental dentro de un rango, ajustando el Factor de Masas entre 1 y 3, por lo que a efectos de estudiar la sensibilidad ante una excitación forzada debe entenderse que la resonancia puede darse *en un rango de frecuencias* .Así las frecuencias de resonancia para un modo propio podrán estar en una relación de 1.73 entre su máximo y su mínimo (el máximo se calcula utilizando un factor de masas 1)

Dimensionalidad relativa del movimiento: Ante una oscilación libre si el movimiento fuera unidimensional y estuviera alineado con la rigidez del sistema el factor de masas sería 1, pues toda la rigidez asume la inercia como si de un muelle se tratara; pero por el contrario si fuera la configuración de la estructura tridimensional respecto de la forma libre del movimiento el factor de masas sería 3 dado que se movilizaría la rigidez única para las 3 direcciones de inercia con equidad (pues el problema se plantea igual en X,Y,Z). Obviamente la tenso estructura plana atendería al movimiento plano con un factor de masas 2.

Como vemos el factor de masas deberá atender tanto la forma de excitación como la dimensionalidad de la estructura. Lo importante es el concepto:

Las frecuencias fundamentales de las tenso estructuras no son valores concretos: la singularidad de su rigidez define un rango de frecuencias para cada modo propio.

.

Análisis del movimiento y límites de validez.

Llegados a este punto el comportamiento dinámico de una tenso estructura queda caracterizado dentro de unos ciertos límites que conviene analizar:

Amplitud máxima del movimiento y oscilación de la rigidez: La oscilación compatibiliza los elementos entre si, equilibrándose las fuerzas de inercia gracias al ajuste de la posición de los nudos en asociación con la variación de la elongación de las barras. Al ser un movimiento cíclico las barras se alargarán y encogerán respecto de su punto de partida, proceso que normalmente se desarrollará dentro de un rango limitado por dos tipos de situaciones "extremas": la decompresión de las barras como límite de la pérdida de tensión y la plastificación de las mismas como extremo superior de carga. Entre ambas situaciones se producirá una variación del módulo de elasticidad tangente/secante al variar la tensión. Estas no linealidades no se modelizan y deberán por tanto interpretarse adecuadamente los resultados a tenor de ello.

- Decompresión de los elementos: Cuando la estructura se pone en carga adquiriendo su forma de equilibrio las barras experimentan un cierto alargamiento inicial. Como vimos dicha magnitud es conceptualmente muy importante, pues la longitud de cálculo de las barras será la que se deduzca de la geometría de equilibrio menos dicho alargamiento inicial. Ello garantizará que cuando las barras entren en carga la forma sea la prevista según vimos en (Rd.3).

$$A_{in,i} = \frac{F_i \times L_i}{\Omega_i \times E_i}$$

Al producirse la oscilación las barras se alargan y acortan. Si el acortamiento fruto de la oscilación supera el alargamiento inicial -de puesta en carga- la barra entraría en compresión, lo cual sería inconsistente con el modelo de cálculo (que no admite un cambio súbito del signo del esfuerzo de una barra) e inconsistente con la física, pues la esbeltez de los elementos impediría que el elemento entrara en compresión: el elemento quedaría pues en "decompresión". Así superado el "acortamiento" que

admite la barra –al perderse su elongación de equilibrio- se produciría un súbito cambio en la rigidez del sistema que el modelo no es capaz de interpretar. Por tanto la amplitud máxima del movimiento en un ciclo teórico, supuesto éste simétrico respecto de su posición de equilibrio, tendrá que respetar que no se supere el alargamiento inicial de las barras.

El software realiza este análisis sugiriendo una amplitud máxima de oscilación ajustada a esta circunstancia. Se determina además la energía potencial elástica asociada al movimiento “ideal”.

- Plastificación de los elementos: Para obtener la Densidad Dinámica de Fuerza se ha de dimensionar las barras asignándoles una sección y un material. Cuando la estructura se pone en carga la barra experimenta su alargamiento inicial adquiriendo cierta solicitación que distará en cierta medida del agotamiento. Cuando la estructura oscila las barras se alargan, aumentando proporcionalmente la solicitación. Si el alargamiento supera el recorrido elástico admisible esta plastificaría en el mejor de los casos o rompería en el peor de los mismos. En el supuesto de alcanzarse la plastificación la barra ésta sería “incapaz” de proporcionar mayor respuesta en una situación real, situación que el modelo propuesto no es capaz de interpretar con exactitud, pues la Densidad Dinámica de Fuerza supone un comportamiento elástico lineal ideal.

Por tanto la amplitud máxima modelable “con ortodoxia” requeriría que las barras estuvieran dimensionadas para que en la situación de equilibrio inicial estuvieran al 50% de su capacidad real, es decir, optimizadas al 100% con un coeficiente de seguridad de 2. En cualquier caso siempre conviene tener muy clara la amplitud máxima del movimiento a efectos de evitarse decompresiones y plastificaciones.

- Variación no lineal de rigidez: Entre los dos extremos anteriormente expuestos se producirá una variación de la rigidez del sistema fruto de la modificación no lineal del módulo de elasticidad tangente/secante durante la carrera de tensiónes que experimentan las barras al oscilar el sistema. Véase que cuando se parte de valores de tensión altos R tenderá a ser una cifra baja que además será prácticamente insensible a los aumentos de tensión, sin embargo R tenderá a ser más sensible a las reducciones de tensión

$$R = \frac{q^2.d^2.E.A}{(12T^3)}$$

Ello el programa no lo modeliza: como se ha expuesto el modelo introduce la fórmula de Ernst bajo la hipótesis de que las barras estarán suficientemente tensas (módulo de elasticidad tangente) y que la carrera de tensiones durante la oscilación será baja de forma que dicho valor permanezca más o menos cosntante. Por tanto introducida dicha corrección a efectos de modelizar la rigidez real del sistema se supondrá que la elasticidad de las barras se mantiene constante, sin que su analice su variación durante el ciclo. Esto es una hipótesis relativamente correcta al inicio del proceso de la oscilación, cuando la variación de tensión no supone una importante modificación del módulo de elasticidad tangente.

Rigidez Dinámica apropiada respecto al comportamiento simulado. En el apartado concerniente al "*Dimensionado y Establecimiento de la Matriz de Rigidez Dinámica*" se expresaba claramente esta idea: el método propuesto se basa en la compatibilidad elástica y geométrica de las barras, que se orientan y elongan equilibrando una determinada acción según define el Método de la Densidad de Fuerza. Por tanto un movimiento amplio proclive a un equilibrio "pendular" (como una bolsa) en el que la gravedad es una fuerza estabilizadora significativa, o un equilibrio "exclusivamente geométrico" en el que no intervenga la elasticidad ni la variación de tensión, (como el análisis básico de una masa oscilando en una cuerda a tensión constante), en el que no varíe por tanto la energía potencial, tenderán a no ser adecuadamente modeladas con el método propuesto. Por tanto las situaciones dinámicas coherentes con la metodología sugerida son preferentemente aquellas formas ANTICLÁSTICAS (o poco sinclásticas) en las que predomina la respuesta según la compatibilidad entre los elementos dentro de un rango de movimientos adecuado a las propiedades de las barras y su grado de pretensado.

Véase que los métodos energéticos de validación se basan en la energía potencial elástica del sistema, por lo que cuando un equilibrio tienda a lograrse antes por la adaptación de la forma que por la elongabilidad de las barras se abrirá una brecha entre el análisis modal y su verificación energética. Véase que la adaptación de la forma no asociada a una elongación elástica (por ejemplo en una estructura por así decirlo muy "laxa") es antes un "freno" que un comportamiento dotado de un claro "mecanismo de recuperación", como lo es el ajuste elástico.

El programa está dotado de un mecanismo de verificación que compara la frecuencia obtenida por análisis modal respecto a la obtenida por la verificación energética simplificada (ambos casos usando un Fm=1 –unidimensional-). La frontera de error se ha establecido subjetivamente en el 5% si bien ciertamente es un tema que debe quedar bajo interpretación de analista, quien tendrá que interpretar el resultado en base a la forma modal presentada considerando las circunstancias anteriormente descritas. Aquellas formas en las que la compatibilidad geométrica guarde poca relación con la compatibilidad elástica tenderán a manifestar velocidades angulares de oscilación dispares.

Cálculos complementarios asociados al Software

El programa informático incorpora algunos procesos de cálculo que van más allá de la propuesta de método hasta este punto descrita:

- Determinación de los Autovalores y Autovectores de una matriz [M]: Como se sabe los autovalores “v” y autovectores (aV) de una matriz [M] serán aquellos que satisfagan la relación:

$$[M]*(aV)= v*(aV) \rightarrow$$
$$\rightarrow ([M]-[I]*v)*(aV)=0$$

[I] es la matriz unidad, donde todo sus elementos son nulos salvo la diagonal, que serán 1

  Lo que implica que el determinante de ([M]-[I]*v) deberá ser nulo. La imposición de nulidad de determinante conduce a la resolución de un polinomio característico de grano “n” siendo n el rango de la matriz. La resolución de un polinomio de gado cinco o superior ha de resolverse iterativamente por métodos numéricos, problema que afronta el Sofware. El algoritmo seguido para la obtención de autovalores y autovectores se describe en la parte de la Tesis correspondiente al Programa.

- Ajuste iterativo de la forma en función del peso de las barras y de los elementos de discretización superficial: La forma de la membrana es aquella configuración de equilibrio que balancea las fuerzas de pretensado y las

acciones externas. Entre dichas acciones externas está la gravedad, que supone una acción importante cuando las barras de un material pesado tienen secciones significativas o el peso de la superficie es relevante. Asignado un material y una sección el programa calcula el "peso real" de las barras y superficies, comparándose con el valor de peso propio asignado en los nudos como masa de cálculo. Si el peso es superior o inferior recalcula la estructura con dicho "peso real", variando la longitud y por tanto también el propio "peso real", repitiéndose el proceso iterativamente hasta que coincidan los valores de masa de cálculo y masa real.

- Discretización superficial por elementos finitos: Aunque algunas superficies son propiamente mallas constituidas por cables de sección establecida, muchos modelos pretenden simular membranas cuya superficie es continua. En la metodología propuesta las superficies se simulan mediante barras si bien éstas habrán de tener unas propiedades congruentes con la fracción de superficie que tributa sobre ellas. Entre otros fines se crean los elementos finitos para ello, entendiendo los EF como ámbitos inscritos en un polígono convexo de barras de tal modo que dicha área y sus propiedades asociadas podrían transferirse a las barras y nudos circundantes mediante una serie de reglas. Así las masas, cargas y propiedades mecánicas superficiales quedarán plasmados en atributos asociados a elementos lineales y puntos, lo que análogamente implicará para algunos parámetros un proceso iterativo de cálculo (asignación de masas, cargas superficiales y opcionalmente rigideces). Esta modelización de la rigidez superficial no es objeto de esta tesis (se ha realizado para la masa) aunque el programa tiene el motor de discretización superficial desarrollado.

Líneas de desarrollo

Incorporable al modelo sería:

- Condensación Geométrica: metodología que permitiría la deducción de formas preestablecidas dentro de unos límites (para que resulte un sistema resoluble). Para ello se fijarían las coordenadas de una serie de puntos –que se

establecerían como condiciones de partida-, para los cuales se impondría la ausencia de reacciones fijadas sus coordenadas. Para ello se tendría que asumir la libertad de densidad de fuerza para una serie de barras. El sistema planteado debería "reformularse" con la metodología expuesta para la condensación de apoyos, pero en este caso siendo variables las densidades de fuerza necesarias para ajustar la forma a los criterios dados con ausencia de reacciones en los puntos establecidos. Habría que plantear la mecánica matricial de las operaciones y una pauta que estableciera las variables en base a las coacciones.

- Modelo con condiciones de contorno diferenciadas en X,Y,Z. La resolución del sistema aunque respondería a la misma metodología que el método ahora expuesto tendría que formularse de forma diferenciada en X, Y, Z atendiendo a las particularizadas condiciones de contorno de cada dirección. Actualmente se resuelve de forma agrupada en las tres direcciones, por lo que el sistema tendría que resolverse de forma separada para cada eje. Si varían las condiciones de contorno en una dirección variarán las frecuencias asociadas, por lo que la sincronización de la forma de oscilación conjunta se basará en un ajuste de los posibles parámetros para que resulte una frecuencia coincidente en las 3 direcciones. Ello implicaría deducir las posibles amplitudes y masas tributarias asociadas a cada dirección del movimiento, lo que conduciría a un rango de soluciones posibles. Por tanto además de ampliarse el planteamiento del problema se tendría que realizar un profundo estudio de los factores implicados, como el Factor de Masas.

- Análisis de los límites de amplitud mediante el ajuste de densidades una vez se superan ciertos rangos de amplitud del movimiento: Actualmente todo el proceso de cálculo es lineal, lo que sólo permite estudiar movimientos de una amplitud muy acotada. Permitiría abordar análisis más complejos el introducir la plastificación de los elementos una vez se excede su elasticidad (lo que implicaría que pasado cierto punto la barra trabajase a carga constante), o el considerar densidad prácticamente cero en los elementos cuando la amplitud del ciclo los llevara hacia la compresión (si se pusiera cero se darían singularidades matemáticas). También sería interesante el introducir la posibilidad que un elemento pueda trabajar a compresión estando previamente

a tracción o viceversa (lo que implicaría la deducción de una Densidad Dinámica de Fuerza de signo contraria a la original de la barra para esa parte del movimiento). La introducción de estas "discontinuidades" dentro de un ciclo de oscilación implicará la alteración de la dinámica del movimiento al producirse los cambios de rigidez, lo que implicaría alteraciones en la frecuencia y forma del ciclo (estos dejarían de ser formas simétricas, etc…). Este tema ha sido anteriormente comentado cuando se trataron las limitaciones del método.

- Aunque el modelo actualmente está orientado al estudio de tenso estructuras de barras de propiedades constantes sería interesante desarrollar una Densidad de Forma "*evolutiva*" que estuviera asociada a las propiedades del elemento finito. El modelo actualmente planteado discretiza utilizando barras que tienen una sección establecida, si bien una mejora significativa sería el introducir la posibilidad de asignar rigidez a las barras en relación a las superficies tributarias y propiedades geometricas de los elementos que ficticiamente quedan acotados sobre la superficie (entendidos estos como un conjunto de nudos, barras y superficie). Ello implicaría un proceso recursivo de cálculo de la rigidez (ahora lo realiza con la masa), pues deberá haber una rutina que obligue a que las propiedades de las barras converjan con los parámetros de las superficies que simulan **(actualmente las bases de dicho planteamiento está en desarrollo).** Análogamente se asignarían a los nudos las acciones del modelo que tuvieran carácter superficial.

- Análisis detallado del Factor de Masas atendiendo a un posible "desfase" por direcciones: Se debería estudiar la influencia del "desfase" en las amplitudes sincronizadas "X,Y,Z" (análisis de posibles modos de vibración según formas *cónicas*). Véase que un desfase (por ejemplo 90º) de las amplitudes en las diferentes direcciones generaría una solicitación sobre la rigidez única del sistema dado que las fuerzas derivada de la inercia no actuarían a la vez

- Desarrollo de los espectros de respuesta no elásticos que traten específicamente la no linealidad de este tipo de estructuras. La aplicación práctica de los resultados obtenidos requiere de un profundo entendimiento del fenómeno a la par que una meditada prudencia a la hora de aplicar las conclusiones del análisis dadas las simplificaciones descritas. Tener unos espectros experimentales de respuesta que correlacionen las frecuencias

fundamentales halladas elásticamente con las características de las diferentes fuentes de excitación (sismo, uso humano, etc..) ayudaría mucho al uso general y a la normalización de la técnica. Para ello se requeriría de un concienzudo proceso de ensayo de sistemas de una complejidad moderada de forma que se pudieran realizar las correlaciones de respuestas, tal y como se ha hecho con las estructuras convencionales.

Software.

Introducción

Esta Tesis, cuyo objeto es la propuesta de un método de análisis dinámico para las tenso estructuras, desarrolló en su primera parte los Principios y la Metodología constituyentes del proceso de cálculo, estableciéndose su base teórica y las pautas numéricas necesarias para resolver analíticamente un caso que se pudiera plantear. Es objeto de esta segunda parte la expresión de todo ello en forma de un Software que esté orientado a la resolución de los casos prácticos más habituales. Como hemos tratado los casos de análisis se deberán ajustar al campo de validez del método, que es suficientemente amplio como para afrontar los casos más comunes abordados por la técnica, en los que la rigidez elástica del sistema es la fuerza estabilizadora dominante cuando el sistema adapta su geometría para equilibrar las fuerzas de inercia.

Llegados a este punto de la exposición se ha propuesto una cadena de cálculos expresable matricialmente, no obstante dicha cadena por si sola no basta para constituir un software puesto que además del proceso descrito habrá que tratar:

- Los interfaces que habiliten tanto la introducción de datos y parámetros como la exposición de resultados en formato de texto.
- El motor gráfico que permita la exposición visual de figuras y resultados.
- La gestión de datos de forma que los valores introducidos se ajusten a la mecánica de cálculo propuesta
- La elaboración de archivos que permitan guardar los casos de estudio.
- Las rutinas complementarias de cálculo que apoyan el método, como los algoritmos de operación de matrices (inversa, transpuesta, suma, resta, producto, etc…), la resolución numérica del problema de autovalores o los procesos recursivos de ajustes de masas y superficies.

Así pues de la adecuada conjugación de todos estos aspectos con la metodología propuesta surgirá el programa, labor que habrá de hacerse con perspectiva para que el programa resulte amigable e intuitivo al usuario.

Esta segunda parte expondrá la concepción del software, describiéndose el significado físico de las variables intervinientes y la mecánica operacional del programa.

Al final la Tesis (anexo III) se incluye un breve manual de usuario del programa. Una lectura del mismo simultánea con el manejo del software familiarizará al usuario lo suficiente como para que éste adquiera soltura en el uso del mismo por si solo.

Planteamiento del Programa

A nivel de usuario el programa se concibe en torno a “ventanas” jerarquizadas en dos niveles. El primer nivel lo constituyen las ventanas esenciales para el manejo del programa que son:

- Control
- Gráficos

El segundo nivel de ventanas es más específico. Lo constituyen los “Navegadores”, lo cuales gestionan la interfaz con el usuario por contenidos organizados en torno a los siguientes ámbitos:

- Nudos.
- Barras.
- Cálculo.
- Frecuencias.
- Elementos finitos.

En el primer ámbito se definen las características esenciales del proyecto, como son sus magnitudes generales, el tipo de ordenamiento para la retícula base de partida, las propiedades que se tomarán como denominador común para los elementos del proyecto… así como la gestión general del programa como por ejemplo el manejo de archivos, la interfaz gráfica e incluso la llamada a los navegadores; mientras que en el segundo nivel es donde se determinan las singularidades de la estructura: configuración de apoyos, cargas, propiedades específicas de las barras, rutinas particulares de cálculo…

Las operaciones que realizadas por el programa así como el manejo de sus principales parámetros se expondrá a medida que se desarrollen los citados dos niveles si bien el planteamiento de manejo es el que sigue:

El programa modeliza partiendo de una malla de partida, la cual se ajustará a las dimensiones de "contorno" del problema conforme al tipo de "ordenamiento de vínculos" que el usuario estime entre las mallas disponibles "cuadradas, triangulares o cuadradas con diagonales". Supongamos un caso cuya planta será "inscribible" en un contorno de 100 x 200 que deseamos discretizar con elementos triangulares partiendo de una densidad de mallado de un elemento "origen" de 12 x 12.

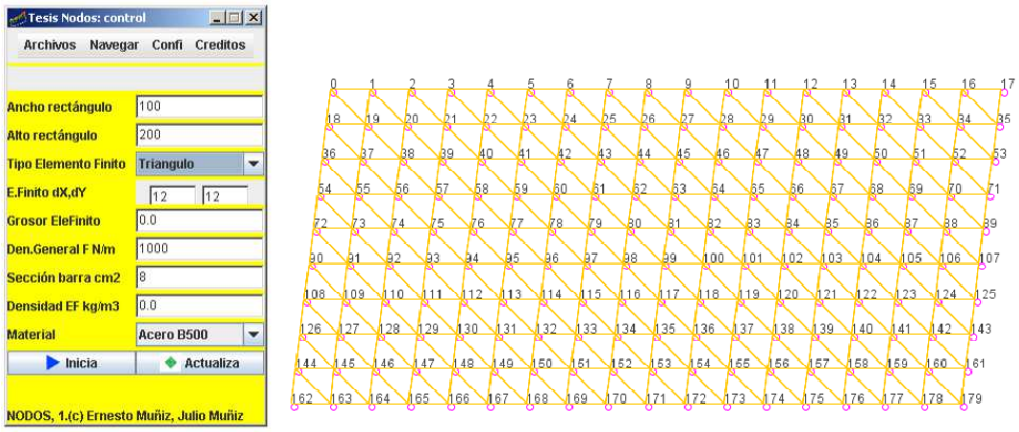


Figura 29: Malla de partida de un caso práctico (Software)

(al ser "no múltiplos" elemento y malla el programa ajusta la dimensión de contorno respetando tamaño de elemento)

Definidas las condiciones de partida (estableciéndose una densidad de Fuerza, sección y material "tipo" para los elementos de forma general) procederíamos a definir las condiciones de contorno geométrico de la malla, ubicando los apoyos donde corresponda (el usuario podrá ubicar arbitrariamente los apoyos en X,Y,Z).

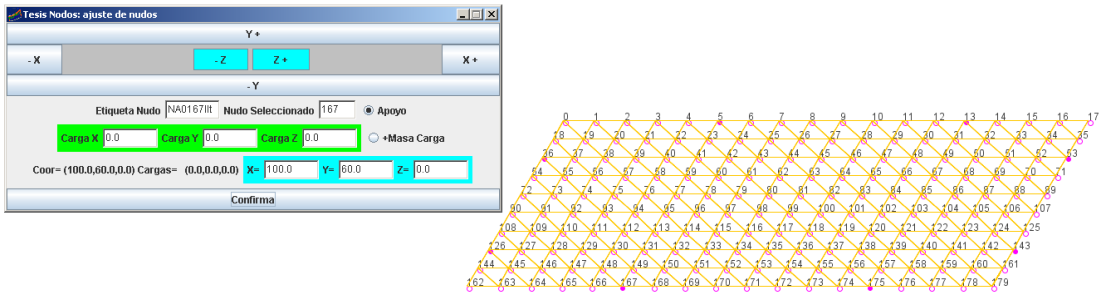


Figura 30: Ajuste de apoyos (Software)

Tras lo cual añadiríamos los nudos y vínculos externos:

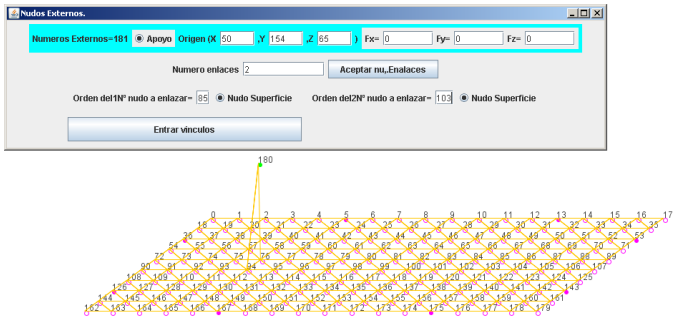


Figura 31: Introducción de víclulo externo (Software)

Así como las cargas aplicadas sobre los nudos:

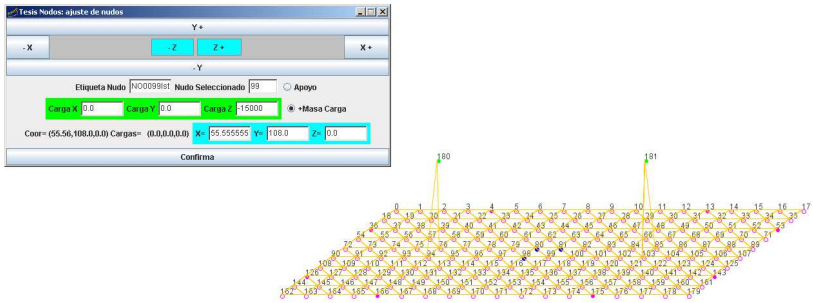


Figura 32: Ajuste de cargas sobre nudos (Software)

Pudiéndose proceder a la determinación de la forma inicial de equilibrio:

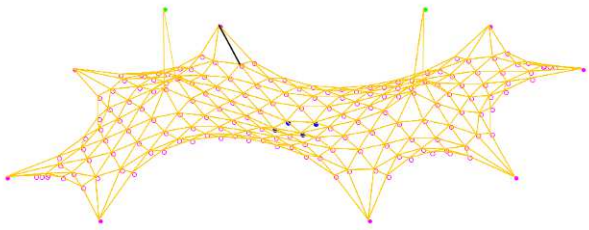


Figura 33: Imagen de forma de equilibrio (Software)

La forma de equilibrio realmente puede obtenerse en cualquier momento del proceso.

Tras esto se procedería a ajustar la forma a los “requerimientos de diseño” mediante el ajuste de las densidades de fuerza de aquellas barras que el usuario seleccione. De esta manera se moldeará la figura dentro de los límites establecidos por el equilibrio. Véase como se ha elevado uno de los vértices interiores de nuestro ejemplo:

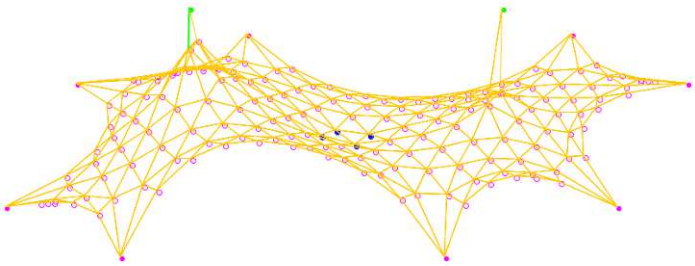


Figura 34: Ajuste de formas actuando sobre las densidades de las barras (Software)

Es importante destacar más allá de la “búsqueda de forma” el importante sentido físico de la densidad de fuerza, dado que establecerá el grado de pretensado de partida del sistema y por tanto su sensibilidad “relativa” a las acciones externas: una densidad de fuerza mayor implica un pretensado, una tensión de trabajo y por tanto un dimensionamiento mayor.

Una vez establecida una forma de equilibrio a nuestro interés podremos analizar el dimensionamiento de la estructura (ratio de agotamiento de las barras), de forma que ajustemos como usuario las secciones y materiales a los requisitos de proyecto, lo cual debe hacerse (como se ha tratado con anterioridad) entendiendo los límites de modelado.

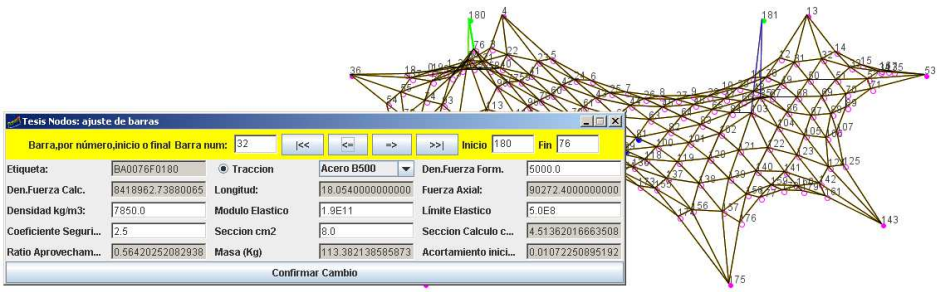
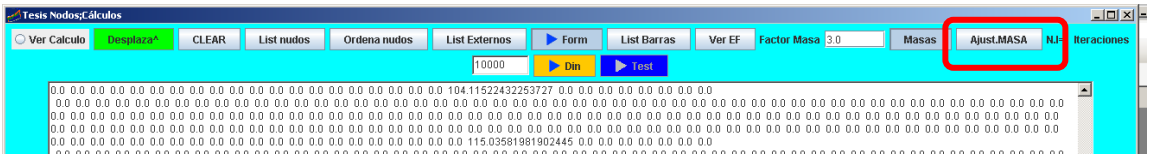


Figura 35: Dimensionado de barras sobre la malla (Software)

Ajustado ya el modelo se procederá a su “ajuste de masas”, de forma que la masa del sistema sea convergente con su geometría, quedando ya físicamente ajustado el modelo. Las propiedades de barras, elementos finitos, masas en nudos, etc… estarán pues ya listas para su análisis dinámico.



En este momento se procederá a realizar el Cálculo Dinámico (Din) del Sistema así como su Verificación (Test).

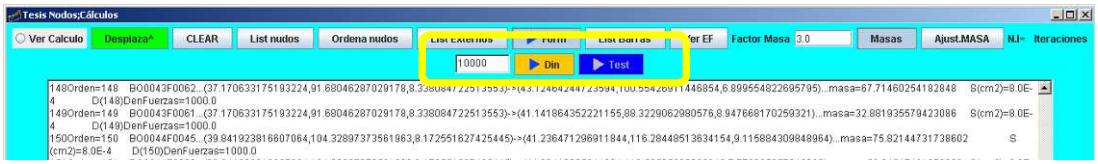


Figura 36: Realización del cálculo dinámico (Software)

Tras dicho proceso de cálculo se accederá al análisis Dinámico de los resultados, obteniéndose entre otros resultados las frecuencias fundamentales, los modos propios, así como el Factor de Masas. El análisis del Factor de Masas es extremadamente conceptual, habiéndose dispuesto un campo en el que el usuario podrá disponer un valor entre 0 y 1 para dicho factor en cada uno de los ejes X,Y,Z.

El programa representará las formas modales de oscilación asociadamente al factor de masas en cada eje.

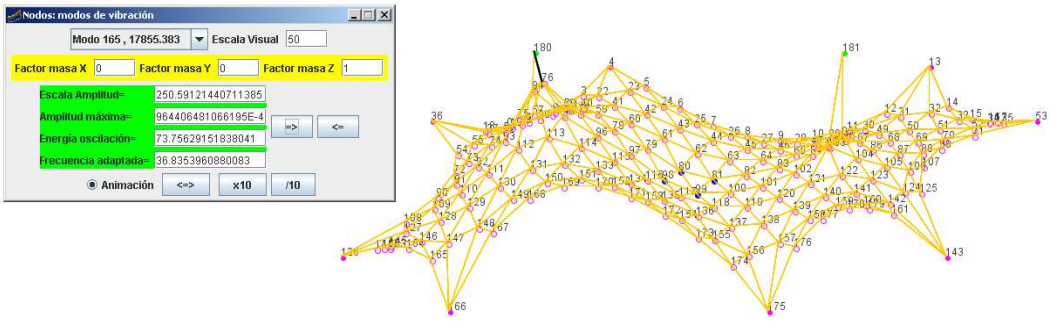


Figura 37: Resultados del cálculo dinámico (Software)

Este proceso será detallado más adelante. Por el tiempo requerido de cálculo se recomienda que la familiarización con el programa se realice sobre modelos sencillos.

Bases y arquitectura del software

A nivel de programación se opta por el Java (El entorno utilizado es el Netbeans 8 de Oracle, que es un software libre) al ser éste un lenguaje actual, multiplataforma (lo que permite compilar el código para diferentes sistemas operativos), dotado de una gran potencia de cálculo (gestiona la memoria con “punteros”) y eminentemente orientado al objeto, lo cual como se intentará exponer es una gran virtud para este tipo de casos.

La orientación al objeto es una “filosofía” de programación según la cual el código no es una simple secuencia lineal de comandos, por el contrario es una relación de objetos programables, los cuales estarán dotados de propiedades cualitativas/cuantitativas (se les asigna atributos/valores específicos) y funcionales (pueden contener algoritmos propios y llamadas a otros algoritmos). Así desde una secuencia de programación “principal” se llama a los objetos (programados a conveniencia) de forma que se establece una serie de relaciones cruzadas entre las partes del programa según esta filosofía.

El Java se articula en torno a clases, dotadas éstas de atributos y funciones.

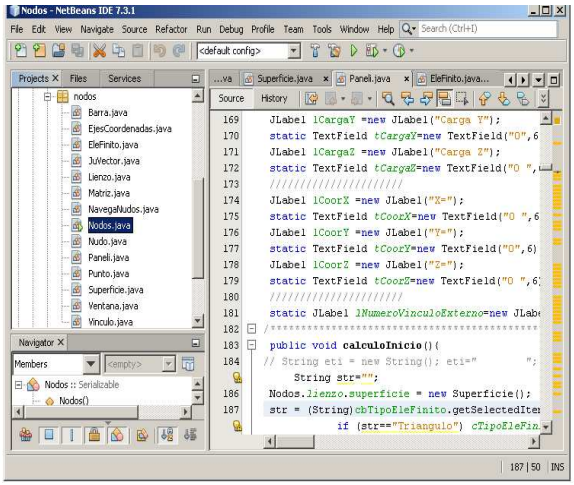


Figura 38: imagen las clases constituyentes del programa.

En nuestro programa la clase principal es la clase Nodos (destacada) , pues desde ella se inicializa el funcionamiento general al activarse las relaciones con las otras clases del programa. Sin entrar a excesivo detalle se procurará mostrar dicho comienzo y la arquitectura de relaciones del Java.

Las Clases “generales” tienen una organización en torno a cuatro etapas:

- Primero se declaran los atributos que constituyen dicha Clase.
- Segundo se define “el constructor” de dicha Clase (con el mismo nombre que la clase), donde se inicializan las variables y se hacen las pertinentes llamadas.
- Tercero se declaran los métodos de asignación/obtención sobre los atributos (get/set), es decir se pone nombre a la introducción y obtención de valores sobre las variables (se definen las llamadas de introducción y obtención de valores)
- Cuarto se programan las funciones que operan en la Clase.

La Clase “principal” es singular respecto de las otras clases pues tiene una “quinta etapa” en la que se realiza la llamada “Principal” (main) a un “constructor” (que puede estar en la misma clase o no), por donde se inicia el proceso. (véase la línea 101, en la que se realiza llamada al constructor en la propia clase). El programa localiza dónde se encuentra el “main” y comienza en dicho punto.

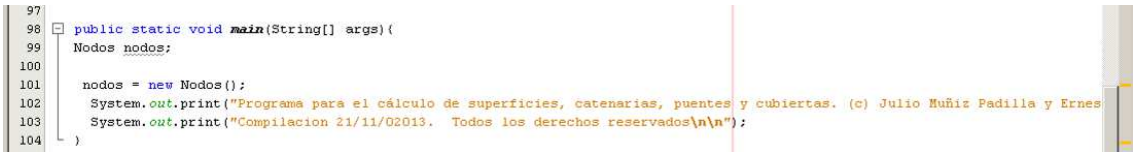


Figura 39: Detalle del código mostrando la parte “main”

Véase el caso expuesto: El inicio se produce cuando en la parte “main” crea el objeto “nodos” al llamar al *constructor* “Nodos ()”.

Vemos como en dicho punto el constructor “Nodos()” contiene llamadas a las Clases “Lienzo()”, “Paneli ()” y “Ventana” con las que trabajará anidadamente dentro de las operaciones que realiza el objeto “nodos” :

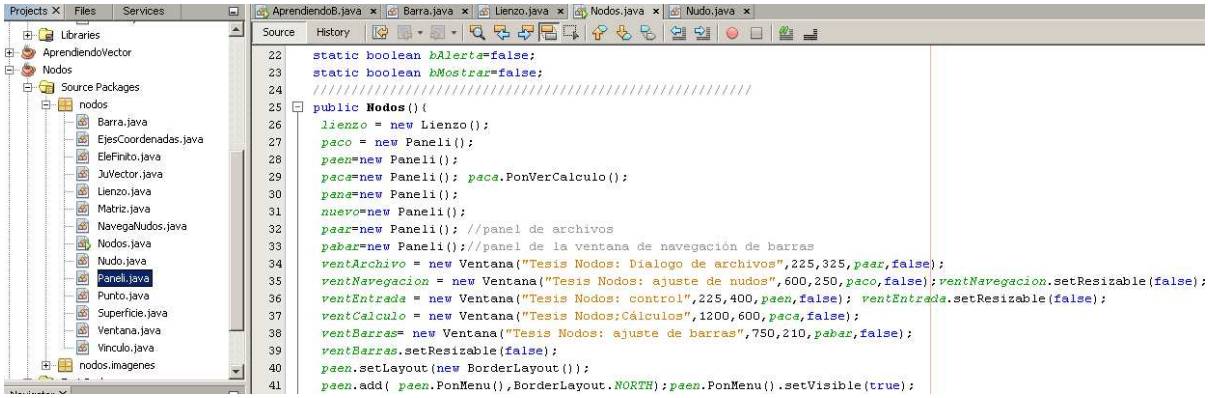


Figura 40: Detalle de código realizandose llamadas de unas clases a otras

Al realizarse dichas llamadas se crea el objeto “lienzo” de la Clase “Lienzo” (véase el matiz de las mayúsculas), los objetos “paco, paen, paca, pana, nuevo, paar, pabar” de la Clase “Paneli” así como los objetos “ventArchivo, ventNavegación, ventEntrada, ventCalculo, ventBarras” de la Clase Ventana.
Cada un objeto es un ente independiente del resto, con su nombre y atributos propios. Véase que no todas las llamadas son iguales: Unas son genéricas mientras que otras -por ejemplo las llamadas a la Clase “Ventana”- contienen información que atribuye propiedades específicas al objeto desde su misma llamada.

De esta forma se va estableciendo un “tejido” de procedimientos convenientemente entrelazados entre si, de forma que el funcionamiento y la información está distribuida en objetos de propiedades singulares.

Para un programa de estructuras ello es sumamente interesante, pues permite crear objetos con sentido “estructural” que contengan todas las características que sean necesarias. Clases singulares en esta línea son:

- Los Nudos
- Las Barras
- Los Elementos Finitos.

Así cada vez que se llama a la pertinente “Clase” se crea un Objeto “Nudo, Barra, o Elemento Finito” (véase en la siguiente imagen las clases destacadas)

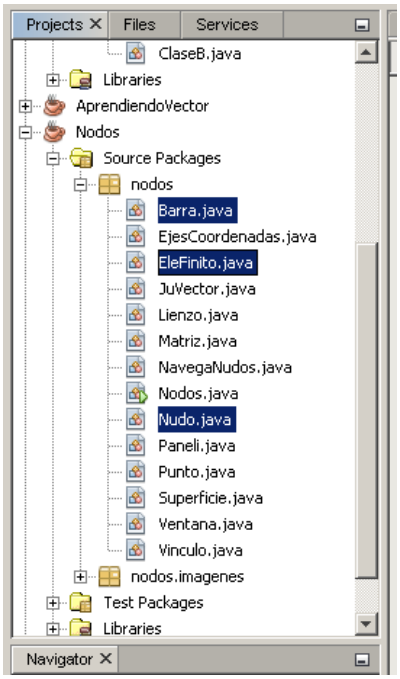


Figura 41: Clases Nudo, Barra y Elefinito destacadas

Creándose una barra, nudo o elemento finito de la estructura. Véase a continuación las propiedades de una barra "tipo" de la estructura:

```
15    String etiqueta;
16    private int orden;
17    private int indice;
18    private int ordenMatrizDensidad;
19    public Nudo origen,fin;
20    private double masa=0; //la hacempos private para que solo se pueda acced
21    private double seccion= 1e-4;//en metros cuadrados
22    double seccionC=1e-4;//  fuerzaAxial*coeficienteSeguridad/Limiteelastico
23    double densiEspecifica=5000; //*por defecto la del acero.
24    private double longitud=1;
25    double limElas=1550e6;//*
26    double modElas=  550e6;//*
27    double enerElas=0.1;
28    char tipo; //*
29     double densiFuerzaF=1000;  //densidad de fuerza para las formas
30    double densiFuerzaC=1000;
31    double tension=100;
32    double fuerzAxial=1;
33    double ratioAgota=1;
34    double acortaInicial=0;
35    double coeficienteSeguridad=1;
36    // g generica, b borde, x externo
37    boolean traccion=true;  // raccion false es compresion.
38    ////////////////////////////////////////////////
```

Figura 42: Atributos clase barra

Estas son:

Etiqueta / Orden / Índice / Orden en la Matriz Densidad / Nudo Origen y Fin / Masa / Sección / Densidad / Longitud / Límite Elástico / Módulo de Elasticidad / Energía Elástica / Densidad de Fuerza / Densidad Dinámica de Fuerza / Tensión / Fuerza Axial / Ratio de Agotamiento / Acortamiento Inicial / Coeficiente de seguridad / Compresión o tracción.

Atributos propios de cada *barra*. Aunque todas estas propiedades comúnmente se inicializan con algún valor algunas de éstas serán asignadas de forma directa: Etiqueta, Sección, Densidad, Límite Elástico, Módulo de Elasticidad, Densidad de Fuerza, su Coeficiente de Seguridad o si es una Barra a Tracción o Compresión; mientras que otros atributos adquirirán su valor como fruto de cálculos o procesos desarrollados durante la ejecución del programa: Orden, Índice, Orden en la Matriz de Densidad, Nudo Origen y Fin, Masa, Longitud, Energía Elástica, Densidad Dinámica de Fuerza, Tensión, Fuerza Axial, Ratio de Agotamiento.

Análogamente las clases Nudos y Elemento Finito tienen atributos coherentes con las necesidades de cálculo y operación asociadas a dichos objetos (Nudos: Índice, Orden, Masa, Fuerzas Externas…. Elemento Finito: Etiqueta, Orden, tipo de malla, nudos constituyentes, barras constituyentes, espesor, …)

Bien sea porque el sistema realiza operaciones o porque el usuario introduce cambios en el modelo, la lógica de relaciones habrá de concebirse de forma tal que se actualicen los atributos de todos los objetos de forma coherente a su contexto. La complejidad de la programación reside en buena parte en las implicaciones derivadas de aplicar la lógica de la programación orientada al objeto a la mecánica de cálculo expuesta en la primera parte de la tesis. A modo de ejemplo mostramos algunos aspectos singulares de interrelación:

- Orden de las Barras: El programa asigna una numeración (Índice) a barras y nudos automáticamente a partir de la malla de modelado de partida. A medida que el modelo se “expande” (incorporando nuevas barras y nudos) el sistema continua atribuyendo numeración de forma correlativa. Dicha numeración será la que el usuario vea en pantalla y sobre la cual se actúe cuando éste quiera modificar los atributos que estén a su alcance (carga en nudos, si el nudo es apoyo… o si la barra tiene tal sección o cual densidad de fuerza…). Sin embargo –como se ha visto al exponerse la condensación de apoyos- el proceso de cálculo EXIGE que las matrices que se operen estén ordenadas de

forma *particular,* de forma que los apoyos sean numerados (ordenados) los primeros. Así, aunque un nudo tenga un determinado Índice en pantalla tendrá un Orden determinado deducido a partir de las condiciones de apoyo del conjunto de la estructura, orden que tendrá que ser adaptable a que el usuario pueda modificar las condiciones de apoyo en el momento que lo estime conveniente. Esto exige unos algoritmos de asignación de orden y la adaptación de TODAS las matrices de cálculo en base a ello. Ello se gestiona sobre la base de que el objeto Nudo tiene un "Índice" (que se ve en pantalla) y un "Orden" (con el que se calcula)

- Orden de las Barras: Como se ha expuesto las barras son las relaciones entre nudos según se ha visto en el proceso de generación de la matriz "G". Por ende si los nudos se reordenan las barras deberán "re ajustarse" a dicho orden, renumerándose de forma SUBORDINADA al orden de los nudos. Así las matrices que contienen la información de las barras (Matriz de Densidades de Fuerza, etc….) tendrán que configurarse subordinadamente a la coyuntura de los nudos, lo cual es posible gracias a unos algoritmos que asignan orden preservando los índices.
- Asignación de elementos finitos: La superficie continua es interpretada a través de un entramado discreto de barras dispuestas a modo de malla. Por tanto para una ajustada interpretación de las propiedades mecánicas del objeto modelado los ámbitos de superficie inscritos en polígonos convexos de barras deberán poder ser "transportados" a las propiedades de las barras y nudos de forma biunívoca. Como en los casos anteriores la identidad de los elementos finitos deberá ser acorde a la dinámica anteriormente expuesta de barras y nudos (lo cual es posible gracias a que el objeto tiene además de un índice un orden independiente), al igual que sus propiedades deberán guardar una relación de "convergencia" con las propiedades de las barras mediante un proceso iterativo de cálculo.

Complementariamente se han tenido que generar rutinas complementarias de cálculo como:

- Rutinas de generación de relaciones: cuando se crea la malla de partida en función al tipo que se elija (rectangular, rectangular con diagonales o triangular) debe existir un procedimiento que vincule los nudos unos con otros, generándose los pertinentes objetos y dotándoles a estos de los específicos atributos sistemáticamente con el orden establecido.

- Rutinas de generación de matrices: Cuando se desea buscar la forma de equilibrio deberán generarse las matrices [G], [D], etc… para ello el sistema en primer lugar deberá ejecutar las rutinas de ordenamiento de nudos (de forma que los apoyos se ordenen los primeros) para entonces empezar a componer de forma sistemática la matriz [G], [D], etc… en base a las propiedades de los objetos y el orden establecido según una lógica programada.
- Operaciones Matriciales: Se han de crear los procedimientos que permitan generar Matrices y sobre estas trasponer, sumar, restar, multiplicar, invertir….
- Resolución de problema de Autovalores y Autovectores. El método de cálculo requiere la resolución de este problema. Para ello se genera una rutina de cálculo (a continuación mostrada):

```
//Algoritmo para obtener el polinomio característico
//cada elemento de la matriz son elementos del array de dos
//dimensiones x[][]
//se usa la función complementaria traza, que calcula la suma de los //valores de la
diagonal de una matriz.

double[] polCaracteristico(){
Matriz Munidad=new Matriz(n);
//matriz unidad con todos los elementos de la diagonal igual a 1.00
for(int i=0; i<n; i++){
Munidad.x[i][i]=1.0;
}
double[] p=new double[n+1];
double[] s=new double[n+1];
for(int i=1; i<=n; i++){
Munidad=this.producto(Munidad, this);
s[i]=Munidad.traza();
}
p[0]=1.0;
p[1]=-s[1];
for(int i=2; i<=n; i++){
p[i]=-s[i]/i;
for(int j=1; j<i; j++){
p[i]-=s[i-j]*p[j]/i;
}
}
return p;
}
// Vector de valores propios sobre la matriz.
// la salida es un array de una sola dimensión llamado valores.
// Se debe poner un límite a las interacciones, por ejemplo maxIter=1000
//este valor se puede modificar en un menú del programa

Matriz valoresPropios(double[] valores, int maxIter)throws ValoresExcepcion{
final double CERO=1e-8;
//declaro el valor final para que se
//comporte como una constante.
double max, dTolera, sumsq;
double x, y, z, c, s;
int contador=0;
//pongo a 0 el contador que tomara como máximo maxIter;
int i, j, k, l;
Matriz a=(Matriz)clone();
//la clase es una derivada la la clase clonable que permite
//hacer copia pero situada en otro jugar de la memoria,
// es decir matrices con punteros independientes.
//copio los valores de la matriz para compararlos luego

Matriz p=new Matriz(n);
Matriz valPropio=new Matriz(n);
//matriz unidad
for(i=0; i<n; i++){
valPropio.x[i][i]=1.0;
```

```
}
do{
//la condición while se hace al final comparando el contador con el //máximo de
iteraciones.
k=0; l=1;
max=Math.abs(a.x[k][1]);
for(i=0; i<n-1; i++){
for(j=i+1; j<n; j++){
if(Math.abs(a.x[i][j])>maximo){
k=i; l=j;
max=Math.abs(a.x[i][j]);
}
}
}
sumsq=0.0;
for(i=0; i<n; i++){
sumsq+=a.x[i][i]* a.x[i][i];
//le sumo el cuadrado del elemento central
}
dTolera=0.0001*Math.sqrt(sumsq)/n;
if(max<dTolera) break;
//calcula la matriz ortogonal de p
//inicialmente es la matriz unidad
for(i=0; i<n; i++){
for(j=0; j<n; j++){
p.x[i][j]=0.0;
}
}
for(i=0; i<n; i++){
p.x[i][i]=1.0;
}
y=a.x[k][k]-a.x[l][l];
if(Math.abs(y)<CERO){
c=s=Math.sin(Math.PI/4);
//el coseno y el seno es igual para 45º
}else{
x=2*a.x[k][l];
z=Math.sqrt(x*x+y*y);
c=Math.sqrt((z+y)/(2*z));
//la función getSigno devuelve +1 0 -1 según valores positivos o nega.
s=getSigno(x/y)*Math.sqrt((z-y)/(2*z));
}
p.x[k][k]=c;
p.x[l][l]=c;
p.x[k][l]=s;
p.x[l][k]=-s;
a=this.producto(p, this.producto(a, this.traspuesta(p)));
valPropio=this.producto(qvalPropio , this.traspuesta(p));
contador++; // contador=contador +1;
}while(contador<maxIter);

if(contador==maxIter){
throw new ValoresExcepcion("número de iteraciones pequeño");
}
//valores propios
//
double[] valores=new double[n];
for(i=0; i<n; i++){
valores[i]=(double)Math.round(a.x[i][i]*1000)/1000;
}
//vectores propios
return valPropio ;
}
////////////////////////////////función traza/////////////////////////////
//es la suma de todos los elementos de la matriz principal
double traza(){
double salida=0.0;
for(int i=0; i<this.numFilas; i++){
for (int j=0;j<this.numColumnas;j++){ salida+=x[i][j];}}
return salida; }


////////////////////////////////////funcion getSigno//////////////////////
int getSigno(double x){
int salida;if(x>0)salida=1; else salida=-1; return salida;}
```

- Rutinas para la visualización gráfica de la estructura: La visualización de giros, escalas, etc… se logra mediante transformaciones vectoriales que se han de programar.
- Etc (Hay decenas de rutinas específicas asociadas a las barras, análisis de resultados, etc…)

El manejo se expone al final de la presente tesis, en el Anexo III. A continuación se exponen los contenidos de los menús desplegables desde un punto de vista conceptual.

Ventana Control

Es la primera ventana que aparece al iniciarse el programa. Adicionalmente es utilizada cuando se inicia un nuevo estudio, se carga un caso anteriormente estudiado (desplegable archivos) o se llama a los navegadores de barras, nudos, etc… (desplegable navegar) por requerimiento del usuario.

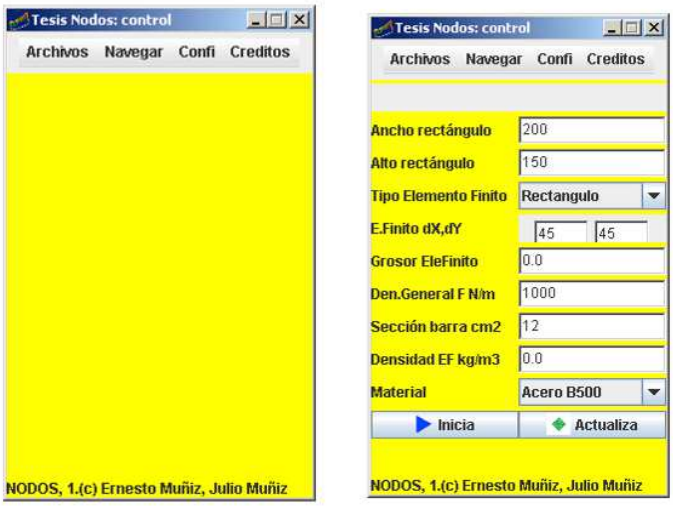


Figura 43: Ventana Control (Software)

La imagen de la izquiera corresponde a cuando se activa la opción de Archivos/Nuevo.

Al iniciarse un caso lo primero que aparece en el desplegable son las propiedades geométricas de la malla de partida: su Ancho (eje X) y Alto (eje Y), así como el tipo de discretización (Tipo Elemento Finito) y tamaño de ésta, referida a cómo es la relación

entre nudos de la malla (patrón rectangular, rectangular con diagonales o triangular - que es un rectángular pero sólo con una diagonal-).

Véase que se trata de una malla de inicio “conceptual” puesto que el usuario podrá ajustar la posición de los nudos de apoyo líbremente, siendo pues lo realmente interesante la forma de relación y la “densidad de vínculos” (frecuencia de nudos) en la “superficie de modelado”, pues el método de la densidad de fuerza ajustará la posición de los nudos a la ubicación que le corresponda por equilibrio.

En el mismo menú vemos:

Grosor de elemento finito: El espesor se utilizará para calcular las propiedades de masa y rigidez de los elementos cuando la superficie sea discretizada (en desarrollo).

Densidad General de Fuerza: Representa la densidad de fuerza que por defecto se asigna a todas las barras de la malla en el momento de creación de la superficie de modelado. Posteriormente el usuario podrá aplicar libremente densidades de fuerza específicas a las distintas barras durante el proceso de búsqueda de la forma. Es importante resaltar el sentido físico de esta magnitud, dada su relación con el grado de pretensado del sistema: a mayor densidad de fuerza mayor tensión interna del conjunto.

Sección Barra: A la hora de “materizalizar” la estructura y obtenerse las Densidades Dinámicas de Fuerza los elementos deben tener una sección de partida. En esta ventana se aplica una sección por defecto que luego el usuario podrá configurar elemento a elemento.

Densidad EF: Al igual que el grosor es un parámetro para el modelizado de la superficie cuando esta sea discretizada en elementos (en desarrollo)

Material: Asigna propiedades por defecto a las barras (módulo de elasticidad, límite elástico, peso específico, etc…) para su dimensionado y obtención de las Densidades Dinámicas de Fuerza. Como en el resto de casos el usuario podrá modificar los parámetros barra a barra.

El resto de apartados de la ventana de control no tienen especial significado conceptual, estando algunas partes del sistema (como expotar DXF, salvar

configuraciones, etc…) en pleno desarrollo. Véase que el Software es un contenido complementario a esta Tesis Doctoral con su propia línea de desarrollo.

Ventana Gráficos

Esta ventana constituye el principal interface para el usuario a efectos del manejo del programa y la interpretación de resultados. Para ello se muestra gráficamente la visualización del modelo de cálculo y se disponen de “botones” de:

- Visualización gráfica: modificación de la persepectiva, ver ejes coordenados, mostrar número y datos de los nudos, etc…
- Añadir externo: implementar nudos y barras adicionales a las de la malla de partida .
- Interpretar resultados como el propio análisis dinámico, las reacciones sobre los apoyos, el grado de aprovechamiento de la estructura, etc…

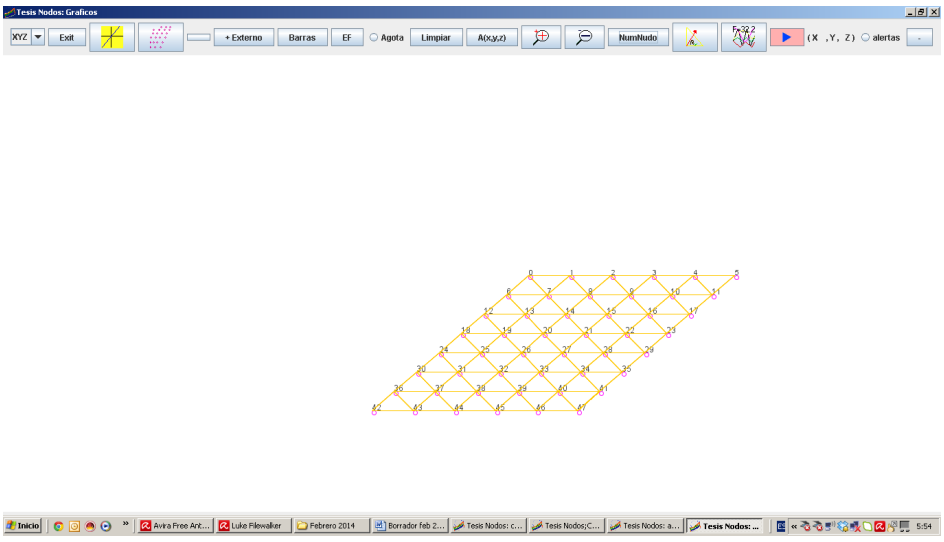


Figura 44: Ventana Gráficos (Software)

Este ámbito tiene un contenido conceptual limitado, si bien es el lugar de acceso para manejo de algunos parámetros importantes de modelado, como todo lo concerniente a la adición de nudos externos:

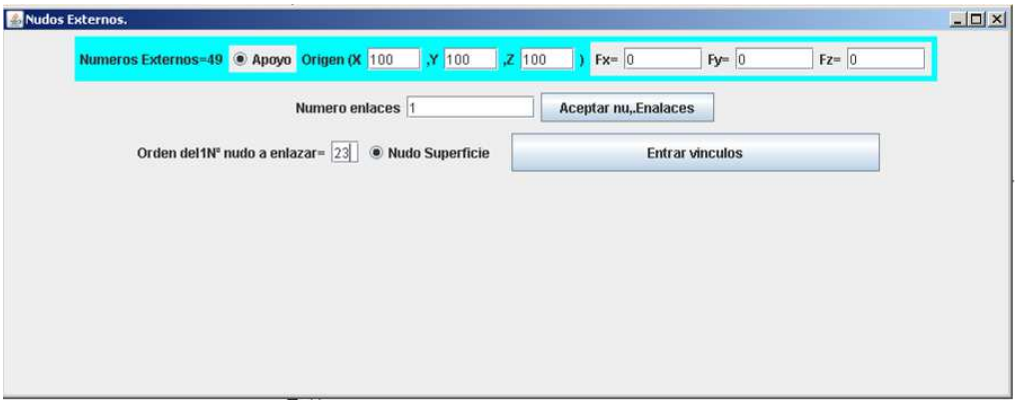


Figura 45: Ventana añadir vículo externo (Software)

Aquí se nos consulta si el nuevo nudo añadido será un apoyo o no, las coordenadas con las que se introduce (lo que tiene sentido sobre todo cuando es apoyo, además de para su primera visualización), si se le aplican cargas y el número de enlaces de éste nudos con otros nudos ya existentes (importante prever que sólo se ha de introducir vínculos que en ese momento sean posibles): de esa forma automáticamente se generan las barras asociadas a los vínculos. Ha de destacarse que los nuevos vínculos y nudos así creados luego serán manejables con los correspondientes navegadores de barras y nudos.

Tiene su interés físico también el visualizar el grado de aprovechamiento de las barras:

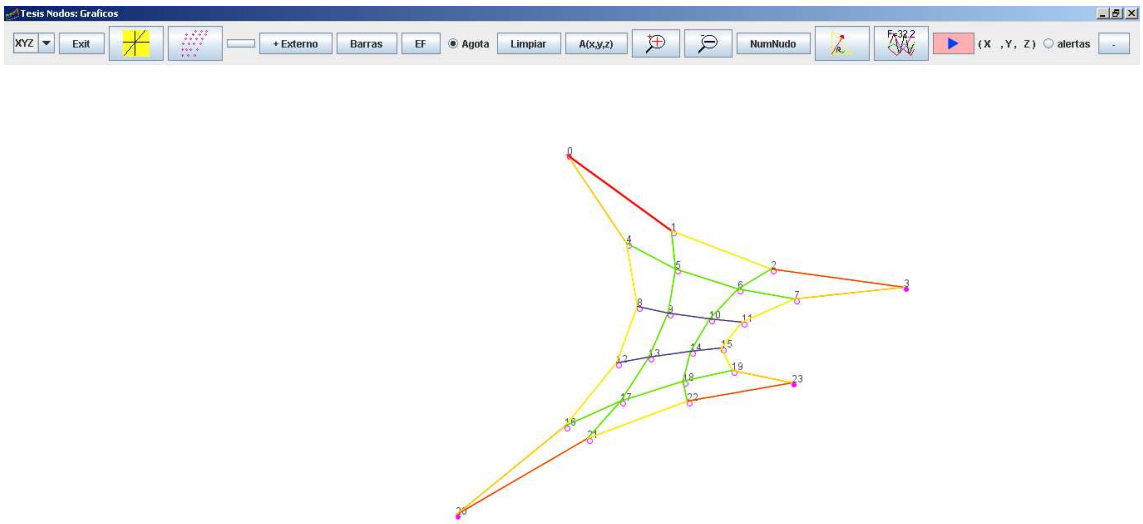


Figura 46: Grado de aprovechamiento de las barras de un sistema (Software)

Deducidos los esfuerzos de equilibrio y para un coeficiente de seguridad establecido se calcula este valor en relación a la sección y material de la barra, mostrándose de forma gráfica según una sucesión de colores:

Negro / Azul/ Verde/ Amarillo/ Naranja / Rojo
Hasta 20% / 40% / 60% / 80% / 100% / Mayor al 100%

Tiene su interés el apreciar que interesa que las estructuras estén en un grado de solicitación elevado, tanto por el aprovechamiento del material como por lo concerniente a evitar la decompresión durante el análisis dinámico. Podría decirse que una situación de buen dimensionado sería intentar lograr que la estructura trabajara en un rango del 80-100% con un coeficiente de seguridad 2 – 2.5. El grado de aprovechamiento elevado permitiría ciertas amplitudes de oscilación sin entrar a decompresión, mientras que el coeficiente de seguridad cubriría la plastificación del material durante la parte del ciclo en la que se produce el mayor estiramiento

Análogamente se muestran las reacciones:

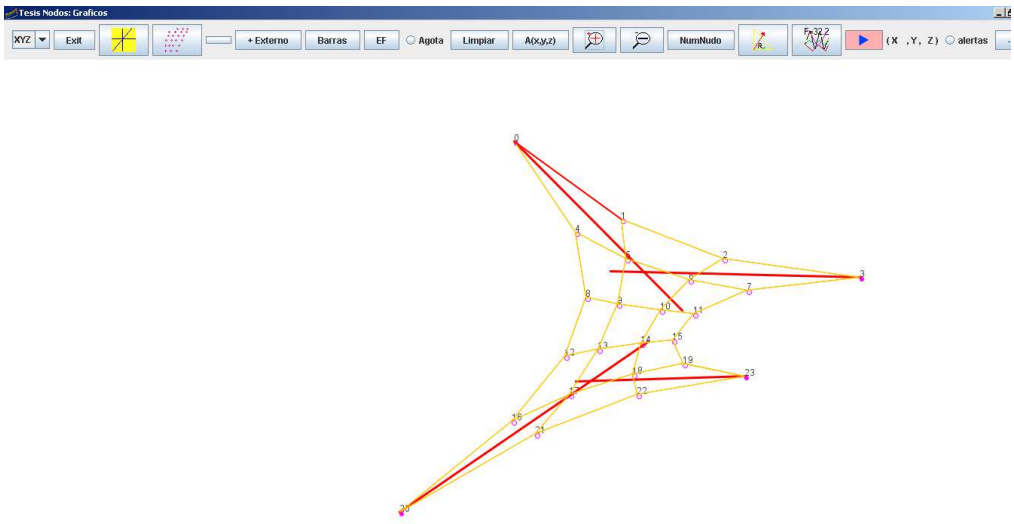


Figura 47: Representación de las reacciones (Software)

Estas se muestran sobre la malla (no sobre los apoyos) dado que representa el equilibrio del sistema.

Navegador de Nudos

Habitualmente este navegador se activa de forma automática al iniciarse el programa, si bien en caso de que el usuario lo cierre o no aparezca se puede activar desde la ventana de control/navegar.

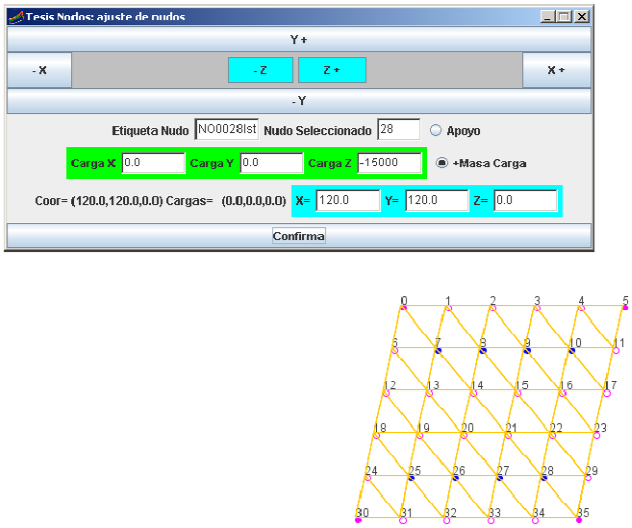


Figura 48: Navegador de nudos (Software)

Desde el mismo se introducen, consultan o modifican las propiedades de cualquier nudo de la estructura.

La parte superior muestra unos pulsadores que permiten modificar la posición del nudo por incrementos sucesivos, lo cual tiene interés básicamente si el nudo es apoyo.

La parte central e inferior de la pantalla contiene información más específica. Así introduciendo la numeración de un nudo podremos seleccionarlo para introducir, consultar o modificar sus características:

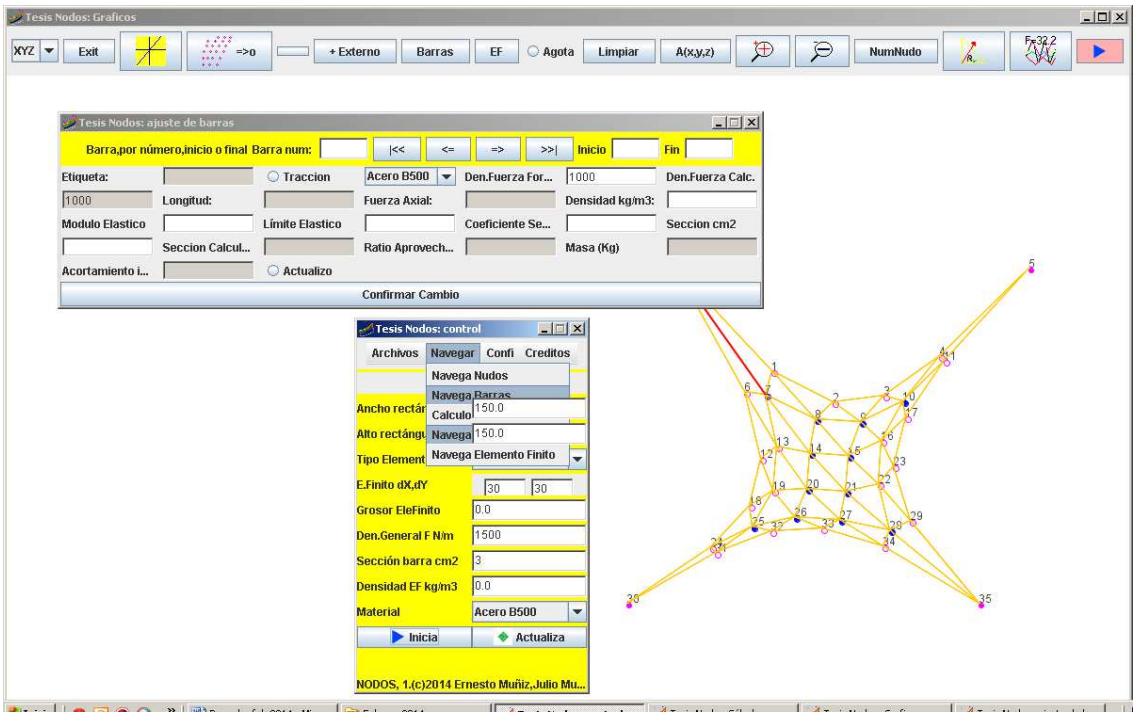
- Si es apoyo o no.
- Las cargas actuantes, y si éstas -en caso de ser en Z (dirección negativa)- son cagas asociadas a masas, lo que debería tenerse en cuenta en el cálculo dinámico (esto es muy importante tenerlo claro en el manejo del programa)

- Las coordenadas del nudo, tanto para conocer la posición del mismo como para modificar sus coordenadas en caso que sea apoyo o interesara visualizar algún cambio.

Para que una modificación sea efectiva siempre habrá “confirmar” pinchando el botón inferior.

Navegador de Barras

La activación de este navegador no es automática dado que para su funcionamiento se requiere que la estructura previamente haya sido calculada al menos una vez, pues muchos atributos de las barras (entre los que está la propia generación de las mismas matricialmente hablando) requieren de un cálculo previo. Por tanto para su activación deberá hacerse un cálculo inicial, debiendo entonces desplegarse desde la ventana de control:



Una vez desplegado seleciconaremos una barra:

Figura 49: Navegador de barras (Software)

Para seleccionar una barra tenemos los botones y pulsadores en el entorno amarillo, mediante los cuales podemos bien introducir su número, avanzar o retrodecer sobre la estructura o por último seleccionar su nudo inicio o fin, lo que conjugadamente con los botones de avance/retroceso permite ubicarnos muy rápidamente sobre la barra que se desee estudiar.

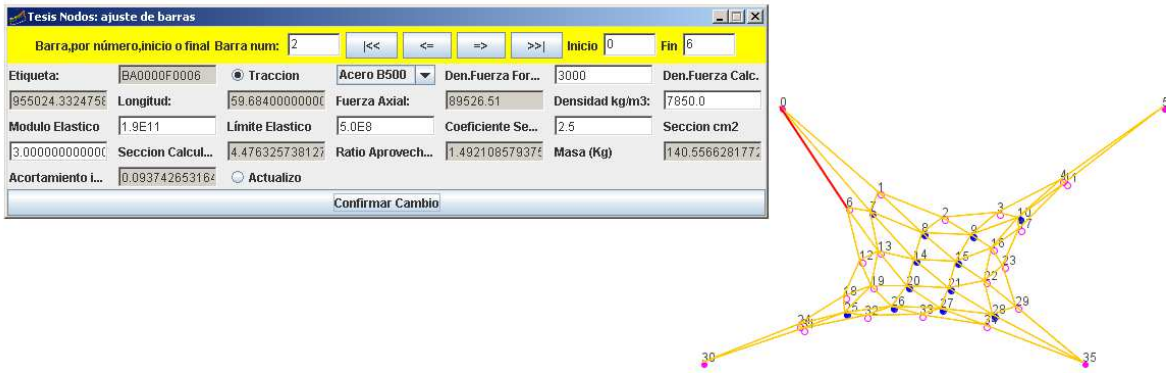
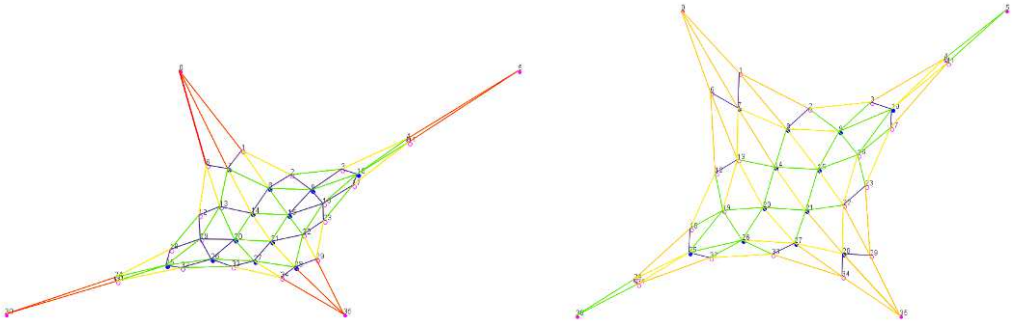


Figura 50: imagen navegando sobre una barra (Software)

El navegador muestra los principales datos de las barra, permitiendo al usuario leer y manipular los mismos: Si la barra es a tracción, su material, la densidad de fuerza, etc…

Modificando la densidad de fuerza de cálculo ajustaremos la forma de la estructura (y por ende la distribución de cargas sobre las barras del sistema). Con el resto de parámetros como la densidad, las propiedades del material (módulo y límite elástico), la sección y el coeficiente de seguridad se calculará la densidad dinámica de fuerza, la masa debida al peso propio así como se estudiará el adecuado dimensionado de la estructura. Véase que ajustando la sección y el coeficiente de seguridad se “afinará” el ajuste de la estructura.



Véase la variación de la estructura ajustando la densidad de fuerza de los apoyos y las secciones de algunas barras

Figura 51: Ajuste de forma actuando sobre las densidades de fuerza de las barras (Software)

Cada vez que se apliquen los cambios habrá que “confirmar cambios”, siendo muy interesante el boton de “Actualizo”. Cuando está activado cada vez que se realiza un cambio se recalcula la estructura, mientras que cuando no está activado guarda los cambios sin recalcular, lo cual facilita que los cambios se puedan hacer agrupadamente de forma rápida.

Ventana de Cálculo

Habitualmente esta ventana se inicia de forma automática, si bien en caso de que el usuario la cierre por error o se requiera su activación siempre podrá activarse desde la ventana de Control/Navegadoras.

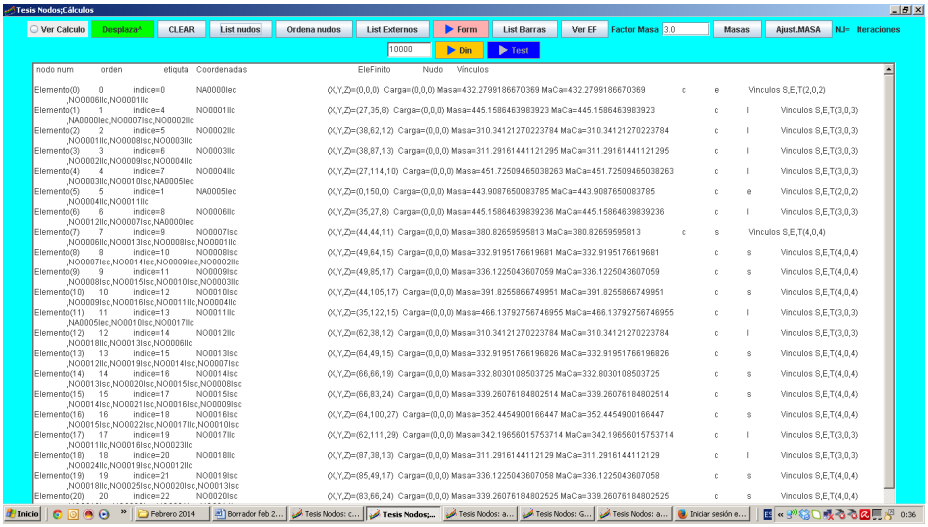


Figura 52: Ventana de cálculo (Software)

Esta ventana es la de mayor contenido conceptual, puesto que muestra todos los procesos del cálculo. La primera de las opciones “ver cálculo” filtra los resultados que se muestran en la ventana de proceso a los más destacados o bien muestra la totalidad de resultados (lo que NO se recomienda). A parte de los botones de “desplazar” y “Clear” (borrar) se muestra una serie de pulsadores que permiten activar partes del proceso de cálculo o muestran algunos resultados específicos.

Así “List Nudos” muestra los nudos según la ordenación “de pantalla”, mostrando su codificación, su numeración “ordenada”, el tipo de nudo que es (si es de borde, de esquina o interior), sus coordenadas, su carga (X,Y,Z) su masa de cálculo y algunas verificaciones de proceso.

“Ordena Nudos” muestra los mismos datos, pero según la ordenación “de cálculo” que será la que encaje con la ordenación de barras y nudos en las matrices. Recordemos que el proceso de cálculo requiere de una ordenación específica según la cual los nudos que sean apoyos se ordenarán los primeros. De esta forma las operaciones matriciales se simplifican cara a facilitar la “condensación de los apoyos”.
“List Externos” muestra de forma diferenciada los nudos que son externos, mostrando su etiqueta, número, orden, número de vínculos y nudos con los que está vinculado.

“Form” activa el cálculo de forma. Durante este proceso se generan una serie de matrices y operaciones según una secuencia:

- Matriz Origen: Es una matriz de relaciones similar a la “G” pero realizada sin que los nudos estén ordenados. Ello permite identificar que las relaciones entre nudos sea la correcta según la numeración visualizada en pantalla (número de barras x número de nudos).
- Matriz G [G]: Es la matriz de relaciones elaborada según el procedimiento de cálculo, es decir, con los apoyos primero. (número de barras x número de nudos).
- Matriz transpuesta de G (trans G) [Gt] (número de nudos x número de barras).
- Listado de barras (a partir de los vínculos) (*). Este listado aparece cuando la forma ya ha sido calculada –por tanto no la primera vez-. Muestra la numeración de barras, su etiqueta, los nudos inicio y final con sus coorenadas, su masa, sección y su densidad de fuerza (para el cálculo de forma).

- Matriz Densidad [D]: Matriz cuadrada (número de barras x número de barras) que muestra en su diagonal la densidad de forma de las barras conforme a la numeración ordenada de éstas.
- Matriz [ME] = trans G x D x G (número de nudos x número de nudos) Esta es la matriz de partida según la metodología de la densidad de fuerza. Los procedimientos de condensación de apoyos y resolución del sistema parten de la manipulación de esta matriz.
- Matriz de coordenadas de los apoyos (numero de nudos x 3). Las primeras filas muestran la posición de los apoyos en X,Y,Z. El resto de elementos es cero.
- Matriz ME modificada con ceros en los nudos que no son apoyos (Fap): es una matriz de cálculo para la condensación de apoyos. (número de nudos x número de nudos)
- Matriz de Fuerzas desde los apoyos [Fuapo]: Parte del proceso de condensación de apoyos, su valor es el resultado del producto entre la matriz [Fap] por la matriz de coordenadas en los apoyos. (numero de nudos x 3)
- Matriz [FUSiste]: Matriz de "pseudo-fuerzas" para la resolución del sistema condensado. Dicha matriz es resultad de la combinación de las coordenadas en los apoyos en las filas que corresponden a los apoyos y de resta de las cargas externas menos [Fuapo] (número de nudos x 3)
- Matriz de resolución del sistema [MESiste]: Proviene de la matriz [ME] pero modificando tantas columas como número de apoyos tenga el sistema, de forma que en dicha parte de la matriz que serán todo ceros salvo los valores de la diagonal que serán unos. (número de nudos x número de nudos)

Msiste	1	2	3	4	5	6	7	8	9
1	1	0	0	0	-1000	-1000	0	0	0
2	0	1	0	0	-1000	0	0	-1000	0
3	0	0	1	0	0	-1000	0	0	-1000
4	0	0	0	1	0	0	0	-1000	-1000
5	0	0	0	0	3000	0	-1000	0	0
6	0	0	0	0	0	3000	-1000	0	0
7	0	0	0	0	-1000	-1000	4000	-1000	-1000
8	0	0	0	0	0	0	-1000	3000	0
9	0	0	0	0	0	0	-1000	0	3000

Figura 53: Esquema de la matriz [MESiste] que equivale a la [Pmod]

- [MESisteInv]: Matriz inversa de la anterior (número de nudos X nudos)
- Matriz de resultados (Las filas que corresponden a los apoyos son unas fuerzas "ficticias" asociadas a los apoyos, y el resto de filas son las

coordenadas de los nudos móviles del sistema). Es resultado del producto [MESisteInv]x[FUSiste] (número de nudos x 3)

- Reacciones en los apoyos: Estas se deducen del producto de [ME]x[Posición de los nudos]
- Matriz de masas asociadas a los nudos: Es una matriz cuadrada (dimensión número de nudos) que en su diagonal contiene la masa asociada a cada nudo. Esta masa proviene por un lado de las masas tributarias de las barras y por otro lado de las masas aplicadas directamente sobre los nudos (fruto de la componente "-Z/9.8" de las cargas externas aplicadas)

El botón "List Barras" lista las barras de forma análoga a la anteriormente descrita.

El botón ver EF (en desarrollo): muestra las propiedades de los elementos finitos, entre las que estarán su numeración y orden, su etiquetado, la superficie y masa asociada, sus parámetros mecánicos (asociados al material y grosor), los nudos y barras que lo constituyen, su perímetro así como la transferencia de masas a los nudos y las propiedades mecánicas que se trasladan a las barras.

El Factor de Masa es un valor que introduce el usuario cuyo significado conceptual se ha tratado durante el desarrollo teórico. Por defecto se muestra un valor de 3 que se recomienda mantener, pues es el que mejor se ajusta para modelar las oscilaciones libres en estructuras tridimensionales. El estudio singular de casos planos o lineales en su propia dimensión invitaría a alterar dicho factor, si bien ello debe hacerse entendiendo adecuadamente el concepto de rigidez espacial asociado a una masa adimensional.

El botón Masas muestra la matriz de masas asociada a los nudos que anteriormente se ha descrito, pues se determina durante el proceso de cálculo general.

El botó Ajust. MASA activa un proceso iterativo de cálculo: En un primer paso se calcula la forma sin considerarse el peso propio de las barras. Una vez conocida la primera forma de equilibrio de la estructura se recalcula el peso propio de las barras con su longitud actualizada, se actualiza el valor de las cargas actuantes y se recalcula la forma, obteniéndose una nueva longitud de barras y por ende una nueva masa. Ese valor se compara con el anterior: si la diferencia de masas está por debajo de un determinado umbral, el proceso se detiene, por el contrario si se supera el umbral se

recalcula la forma con la nueva masa, obteniéndose una nueva longitud y peso para volverse a comparar con el paso anterior. Esto se repetirá hasta que converja la diferencia de masas a un valor por debajo del umbral que se establezca. El proceso será iterativo en la medida que sean mayores las masas por unidad de longitud de barra en relación a que sean bajas las densidades de fuerza de las barras (menor pretensado). Hay un contador que muestra el número de ciclos que tarda en converger el proceso.

El boton “Din” activa el cálculo dinámico, afrontando la resolución del problema de autovalores y autovectores del sistema. El casillero junto al botón muestra el número de iteraciones permitido para la resolución numérica del problema.

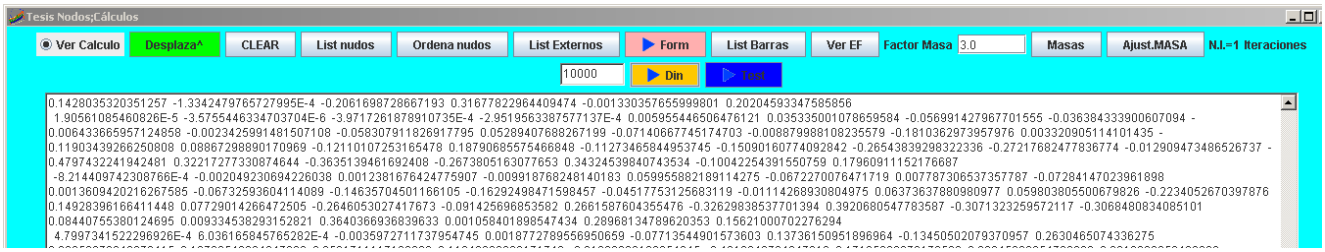


Figura 54: Ventana del proceso de cálculo

Este proceso lleva asociado el cálculo de una serie de matrices que a continuación se exponen:

- Matriz de densidad de fuerzas dinámica [DD]: Representa la rigidez real de las barras que constituyen la estructura una vez ésta ha sido dimensionada (sección) y se conocen las longitudes de los elementos. Ello se expresa con una matriz cuadrada de dimensión número de barras en las que los elementos de la diagonal son las Densidades Dinámicas de Fuerza (DDF) colocadas según el la posición ordenada de barras. (barras x barras)
- Matriz [MDE] = [Gt]*[DD]*[G], siendo la matriz de partida para la obtención de la matriz de rigidez dinamica. (nudos x nudos)
- Matriz de masas reducidas factorizada [MRF]: Matriz de masas a la que se eliminan las filas y columnas correspondientes a los apoyos y que se multiplica por el Factor de Masas. (nudos x nudos)
- Matriz [MDER], igual que la matriz [MDE] pero habiéndose eliminado las filas y columnas que corresponden a los apoyos. (nudos móviles x nudos móviles)

- Matriz inversa de la matriz de masas reducida factorizada [MIRF]. (nudos móviles x nudos móviles)
- Matriz [MA] para el cálculo de los autovalores (nudos móviles x nudos móviles) , que proviene del producto [MIRF]*[MDER]
- Matriz de Valores Autovectores: Es una matriz (nudos móviles x nudos móviles) en la que cada columna tiene los desplazamientos adimensionalesde los nudos asociados a cada forma modal. Las columnas se ordenan por autovalores y las filas por numeración ordenada de nudos.
- Valores propios (VP): Es un vector que muestra los autovalores (número de nudos). Su orden coincide con el orden de las columnas de la matriz de valores Autovectores. Los autovalores muestran el cuadrado de la velocidad angular de oscilación que corresponde a cada forma modal.

El boton “Test” activa la verificación energética. Como en el resto de casos activa un proceso de cálculo que se expresa mediante una sucesión de matrices que procedemos a su sintética descripción:

- Matriz MF de autovectores orlada: Esta matriz añade a las formas modales (matriz de valores autovectores) los desplazamientos de los apoyos (que son cero). Resulta una matriz de dimensión (nudos x nudos móviles)
- Matriz [AL] de alargamiento: aplicando el incremento de coordenadas asociado a la forma modal en X,Y,Z sobre la posición de equilibrio de los nudos calcula los alargamientos que experimentan las barras. Resulta una matriz de dimensión (num barras x nudos móviles)
- Matriz [EAL] de Energía por alargamiento. Calcula para cada barra el valor de 0.5*Alargamiento^2 (num barras x nudos móviles)
- Matriz [EP] de Energía Potencial. Calcula el producto de la Densidad Dinámica de fuerza por las energías de alargamiento [DD]*[EAL], calculando de esta forma las energías potenciales de las barras en cada forma modal (num barras x nudos móviles)
- EPE (p) Energía Potencial Elástica de cada Oscilación. Es un vector de dimensión (nudos móviles) de forma que a cada forma modal le corresponde el valor de energía suma de todas las barras.
- Matriz [EC] de desplazamiento cinético. Es una matriz de dimensión (nudos móviles x nudos móviles) en la que cada columna representa una forma modal de

oscilación, siendo el valor representado en cada casilla 0.5*cuadrado del desplazamiento asociado a la forma modal para ese nudo.

- Matriz [CIN]. Es una matriz de dimensión (nudos móviles x nudos móviles) en la que se realiza el producto de las masas factorizadas por el desplazamiento cinético [MRF]*[EC]
- EPE (c): Es un vector de dimensión el número de nudos móviles, ordenados según los autovalores, en los que cada valor es la suma de cada columna de la matriz [CIN]
- Porcentaje de error de los modos de vibración. El cuadrado de la velocidad angular de oscilación de cada modo de vibración se deduce del cociente entre los valores de EPE(p)/EPE(c). Dichos valores se compararán con los valores propios (VP), calculándose la variación relativa respecto de este último valor. Adicionalmente se calcula la velocidad angular, el periodo y la frecuencia asociada a cada forma de oscilación.

Este navegador es el que muestra todos los procesos numéricos de una forma pormenorizada, si bien la interpretación de los resultados exige un detallado conocimiento de todo el proceso de cálculo. Por ello, aún siendo una información muy valiosa, se recomienda desactivar la opción de ver todo el cálculo.

Ventana de Análisis modal

En este navegador se exponen los resultados del análisis dinámico de una forma “amable”, accediéndose al mismo tanto desde la ventana de Control como desde la ventana de Gráficos.

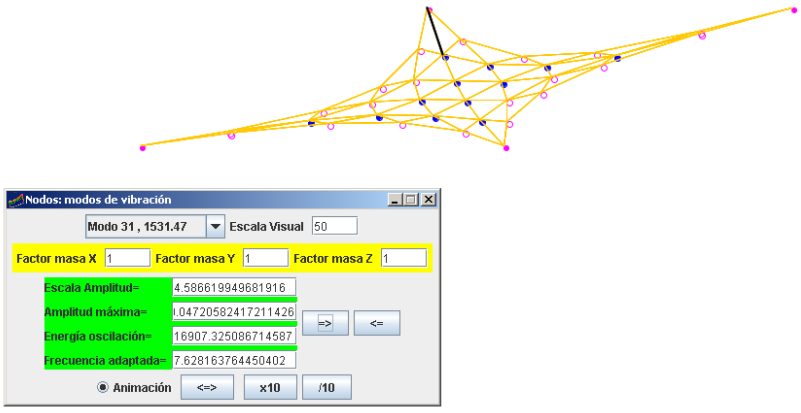


Figura 55: Ventana de análisis dinámico (Software)

Permite seleccionar el modo de vibración que se desea representar, ajustando la escala visual de la distorsión para la fácil interpretación del movimiento. Las flechas que se muestran en la ventana gráfica cambian la visión de la figura, mostrándose sin amplitud de oscilación o con amplitud de oscilación (multiplicado por el factor de escala visual).

Al pie de la imagen muestra la opción de “animación”, acompañado de un botón de doble flecha que activa el proceso y unos factores de x10 ó /10 que atañen a la velocidad de representación del movimiento. Se aconseja que se experimente con los modos menores (velocidades angulares más altas) pues los modos mayores (más lentos) pueden conducir a periodos de representación más prolongados no interrumpibles.

Respecto a los factores de masas estos conciernen al Factor de Masas en cada dirección (que debe ser un valor entre 0 y 1). Como se sabe la modificación de los factores de masas conlleva una variación de frecuencias, recomendándose que se adapte un valor de 1 en cada dirección, salvo que:

a) Estemos ante un problema unidimensional o bidimensional, por lo que se elegirían los factores en la dirección de estudio.
b) Estemos ante una respuesta forzada en una dirección preferente, por lo que en ese caso el factor de masas convendría se ajustara a dicha dirección.
c) Queramos realizar una determinada “visualización”, dado que el movimiento se representa escalado en cada dirección según el factor de masas que le corresponda.

Adicionalmente se muestran una serie de valores:

- Escala de amplitud: Representa la relación entre los valores obtenidos del cálculo y la amplitud posible por los límites de alargamiento de las barras (por decompresión)
- Amplitud máxima (en metros): Representa la amplitud de movimiento dentro del rango de movimientos limitados por la decompresión.
- Energía de oscilación (en Julios): Representa la Energía Potencial asociada al movimiento limitado.
- Frecuencia adaptada: Muestra los valores de velocidad angular corregidos por las posibles variaciones del factor de masas.

Se recomienda que cuando se modifiquen valores se asegure el usuario de que las casillas ciertamente se adaptan. En ocasiones cuando se cambia un parámetro conviene ir a otro modo de oscilación y retornar al modo en análisis para asegurarnos de que los valores se actualizan.

Ventana de elementos finitos

Se trata de una parte del programa en actual desarrollo, si bien se expone por estar creado su navegador y su llamada, pudiendo realizarse tanto desde la ventana de Control como desde la ventana de gráficos del programa.

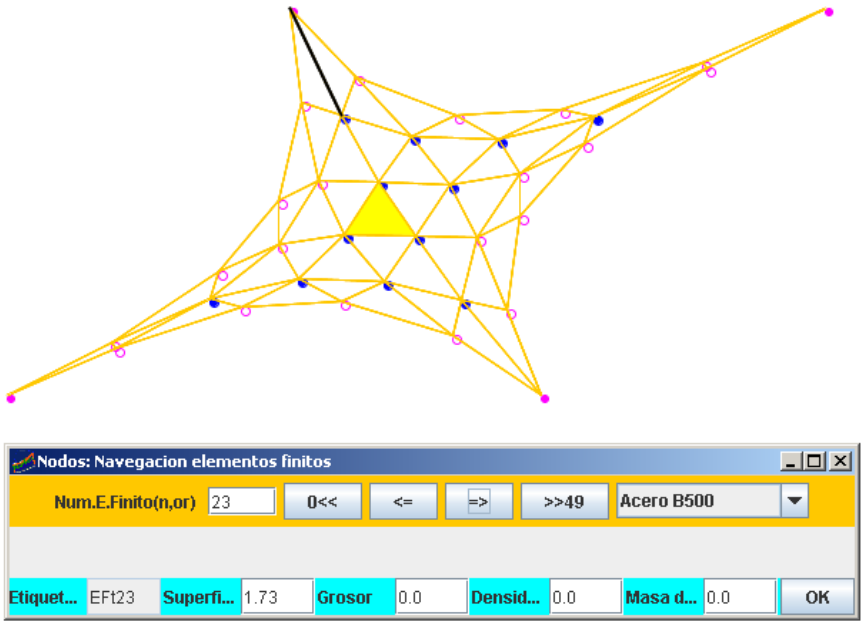


Figura 56: Ventana de elementos finitos (Software)

Como se ha comentado con anterioridad desde este navegador se representarán los principales parámetros de los “elementos finitos”, entendidos estos como un recinto de la superficie acotado por un conjunto de barras y nudos. Su misión es el traslado de

las propiedades superficiales (masa, rigidez, cargas superficiales….) a los nudos y barras que lo delimitan, si bien como se ha comentado en otras partes de la presente tesis este programa no realiza el procedimiento de asimilación de una superficie continua a una discretización de barras. Ello deberá realizarse por cuenta del usuario en caso que fuera de su interés.

Firma

Ernesto Muñiz Martín

Septiembre de 2014.

Anexos.

Anexo I – Código del programa

El programa se ha redactado en Java, haciéndose uso del compilador NetBeans de Oracle. Las clases en torno a las que se organiza el programa son por orden alfabético:

- Barra
- EjesCoordenadas
- EleFinito
- JuVector
- Lienzo
- Matriz
- NavegaNudos
- Nodos
- Nudo
- Paneli
- Punto
- Superficie
- Ventana
- Vinculo

Mostraremos a continuación los códigos de cada una de las clases.

Clase Barra

```
package nodos;
import java.io.Serializable;
import javax.swing.JOptionPane;

/**
 *
 * @author julio MuÃ±iz Padilla y Ernesto MuÃ±iz martÃn
 2014, Islas Canarias.
 */

public class Barra implements Cloneable,Serializable {

 String etiqueta;
 private int orden;
 private int indice;
 private int ordenMatrizDensidad;
 public Nudo origen,fin;
 private double masa=0; //la hacempos private para que solo se pueda
acceder mediante setMasa(doouble m) y getMasa(bool carga)
 private double seccion= 1e-4;//en metros cuadrados
 double seccionC=1e-4;//
fuerzaAxial*coeficienteSeguridad/Limiteelastico
 double densiMasa=5000; //*por defecto la del acero.
 private double longitud=1;
 double limElas=1550e6;//*
 double modElas=  550e6;//*
 double enerElas=0.1;
 char tipo; //*
 double densiFuerzaForma=1000;  //densidad de fuerza para las formas
 //double densiFuerzaC=1000;
 double densiFuerzaDina=1000;
 double tension=100;
 double fuerzAxial=1;
 double ratioAgota=1;
 double acortaInicial=0;
 double coeficienteSeguridad=1;
 boolean traccion=true;  // raccion false es compresion.
 JuVector vector;
 ////////////////////////////////////////////

 void setEtiqueta(){
String sInicio="";
  if(origen!=null && fin!=null){
  if(origen.apoyo ||fin.apoyo) sInicio="BA";else sInicio="BO";
  int Or=this.origen.getOrden(); String
sOr=String.format("%04d",Or).toString();
  int Fi=this.origen.getOrden(); String
sFi=String.format("%04d",Fi).toString();
  if(Or<Fi) { this.etiqueta=String.valueOf(sInicio+sOr+"to"+sFi);}
else{  this.etiqueta=String.valueOf(sInicio+sFi+"to"+sOr); }
   }}

  void setEtiqueta(String eti){ this.etiqueta=eti; }

 void setMaterial(double[] mat){
```

```
      this.densiMasa=mat[0];
      this.limElas=mat[1];
      this.modElas=mat[2];
      this.coeficienteSeguridad=mat[3];
      this.actualizar();
   }

  Barra(){this.origen = new Nudo();
  this.fin=new Nudo(); this.vector=new JuVector(); };

  Barra( Nudo ori, Nudo finis ){
  String sInicio="BO";
  if(ori!=null && finis!=null){
  this.origen = new Nudo(); this.origen=ori;
  this.fin=new Nudo(); this.fin=finis;

  if(origen.apoyo || fin.apoyo) sInicio="BA";
  //  String sOrden =String.format("%04d", pos);
  int Or=this.origen.getOrden(); String
sOr=String.format("%04d",Or).toString();
  int Fi=this.fin.getOrden(); String
sFi=String.format("%04d",Fi).toString();
  if(Or<Fi) {
      this.etiqueta=String.valueOf(sInicio+sOr+"F"+sFi);}
      else{  this.etiqueta=String.valueOf(sInicio+sFi+"F"+sOr); }
  double Dx=this.origen.coordenada.X - this.fin.coordenada.X;
  double Dy=this.origen.coordenada.Y - this.fin.coordenada.Y;
  double Dz=this.origen.coordenada.Z - this.fin.coordenada.Z;
  this.longitud=Math.sqrt(Dx*Dx + Dy*Dy + Dz*Dz);
  this.vector=new JuVector(this.origen.coordenada,this.fin.coordenada);
  }
  this.seccion=Nodos.lienzo.superficie.seccionBarra; // introducimos
cuando creamos la secciÃ³n general

  }

  void setExtremos(Nudo ori, Nudo finis){
  String sInicio="BO";

  this.origen = new Nudo();
  this.fin=new Nudo();
   this.origen=ori;this.fin=finis;

  if(origen.apoyo || fin.apoyo) sInicio="BA";
  int Or=this.origen.getOrden();  String sOr=String.format("%04d",Or);
  int Fi=this.fin.getOrden(); String sFi=String.format("%04d",Fi);

  if(Or>Fi){ this.etiqueta=sInicio+sOr+"F"+sFi;}else{
this.etiqueta=sInicio+sFi+"F"+sOr;}
  double Dx=this.origen.coordenada.X - this.fin.coordenada.X;
  double Dy=this.origen.coordenada.Y - this.fin.coordenada.Y;
  double Dz=this.origen.coordenada.Z - this.fin.coordenada.Z;
  this.longitud=Math.sqrt(Dx*Dx + Dy*Dy + Dz*Dz);
   this.vector=new JuVector(ori.coordenada,finis.coordenada);
  }


  Nudo getOrigen(){return this.origen;}
  Nudo getFin(){return this.fin;}
  int getOrdenOrigen(){return this.origen.getOrden();}
  int getOrdenFin(){return this.fin.getOrden();}
```

```
 double getSeccion(){return this.seccion;};
 double getDensiFuerzaF(){return this.densiFuerzaForma;}
  double getDensiFuerzaC(){ if(this.longitud !=0
)this.densiFuerzaDina= this.modElas*this.seccion/this.longitud;else
JOptionPane.showMessageDialog(null, "DivisiÃ³n por cero");return
this.densiFuerzaDina;}
 void Barra( Nudo ori, Nudo fin, double densi, double sec ){
 this.origen = new Nudo();
 this.fin=new Nudo();
 double Dx=this.origen.coordenada.X - this.fin.coordenada.X;
 double Dy=this.origen.coordenada.Y - this.fin.coordenada.Y;
 double Dz=this.origen.coordenada.Z - this.fin.coordenada.Z;
 this.longitud=Math.sqrt(Dx*Dx + Dy*Dy + Dz*Dz);
 this.densiMasa=densi;
 this.seccion=sec;
 this.masa= this.seccion*this.densiMasa*this.longitud;
 }
 public void setOrden(int or){this.orden=or;}
 public int getOrden(){return this.orden;}
 public void setOrdenMatrizDensidad( int
or){this.ordenMatrizDensidad=or;};
 int getOrdenMatrizDensidad( ){return this.ordenMatrizDensidad;};
 void setDensiEspecifica(double densi){
 this.densiMasa=densi;
 this.masa= this.seccion*this.densiMasa*this.longitud;
 }
 void setSeccion(double sec){
 this.seccion=sec;
 }

 void setTipo(char tip){
 this.tipo=tip;
 }

 void setTraccion(boolean trac){   //solo acepta true false;
 }

 void setLimElastico(double lim){
 this.limElas=lim;
  }

 void setDensiFuerza(double denfu){
 this.densiFuerzaForma=denfu;
 this.actualizar();
 }

 public void setOrdenIndice(int inOrd){
        this.indice   =inOrd;
    }


 void setModElastico(double elas){
 this.modElas=elas;
  }
 void setEneElastica(double ener){
 this.enerElas=ener;
 }
 void setParametrosMaterial(double den, double lE, double mE, double
coeSeg){
 this.coeficienteSeguridad=coeSeg;
 this.limElas=lE;
```

```
 this.modElas=mE;
 this.densiMasa=den;
 if(this.limElas !=0)
this.seccionC=this.getFuerzAxial()*this.coeficienteSeguridad/this.limE
las;
 if
(this.seccionC<this.seccion){if(Nodos.bAlerta)JOptionPane.showMessageD
ialog(  null, "Alerta:SecciÃ³n de la barra insuficiente. No obstante,
seguimos el cÃ¡lculo por el cual el ratio de agotamiento serÃ¡ mayor
que 1");}
 if(this.getLongitud() !=0 )this.densiFuerzaDina=
this.modElas*this.seccion/this.getLongitud();else
JOptionPane.showMessageDialog(null, "DivisiÃ³n por cero");
 if (this.seccion != 0){
     this.ratioAgota=this.seccionC/this.seccion;
     this.masa=this.seccion*this.getLongitud()*this.densiMasa ;


 }else JOptionPane.showMessageDialog(null, "DivisiÃ³n por cero");
 if(this.seccion*this.modElas !=0){
this.acortaInicial=this.getFuerzAxial()*this.getLongitud()/(this.secci
on*this.modElas);} else JOptionPane.showMessageDialog(null, "DivisiÃ³n
por cero");

 //acortamientoInicial
 }

 double getFuerzAxial(){
 this.fuerzAxial=this.densiFuerzaForma *this.getLongitud();
 return this.fuerzAxial;
 }

 double getLongitud(){
 return this.longitud;
 }
 double getMasa(){
 return this.masa;
 }
 public void setIndice(int index){this.indice=index;}
 public int getIndice(){return this.indice;}
 double getTension(){
 return this.tension;
 }

 public String toPrint() {
 String salida="";
 if (this.etiqueta!=null) {
  salida= this.getEtiqueta()+"...(" +this.origen.coordenada.X + ","
+this.origen. coordenada.Y + "," + this.origen.coordenada.Z + ")"+
     "->(" +this.fin.coordenada.X + "," +this.fin. coordenada.Y + ","
+ this.fin.coordenada.Z + ")...masa="+this.getMasa();
}else{ salida="(" +this.origen.coordenada.X + "," +this.origen.
coordenada.Y + "," + this.origen.coordenada.Z + ")"+
     "->(" +this.fin.coordenada.X + "," +this.fin. coordenada.Y + ","
+ this.fin.coordenada.Z + ")  ";}
return salida;
 }

////////////////////////////////////////////////////////////////////
/////
public String getEtiqueta(){return this.etiqueta;}
```

```
////////////////////////////////////////////////////////////////////
////

public static Barra[] generaArrayBarras( Nudo[] nud){
  // Barra[] copia=null;
 //  if(Nodos.lienzo.superficie.barras !=null){  copia=
Nodos.lienzo.superficie.barras.clone(); }

 Nudo[] nd;
 nd=new Nudo[nud.length];for(int n=0;n<nd.length;n++){nd[n]=new
Nudo();}
 nd=Nodos.lienzo.superficie.OrdenarNudos(nud);
 nd=(Nudo[])nud.clone(); // nd=this.OrdenarNudos(nd); //si ordenamos
este array se descolocan todos los nudos
 Barra[] salida;
double numBa=0;
//int numNuTotal= nd.length;
for(int n=0; n<nud.length;n++){ numBa=numBa+nd[n].vinculo.length; }
int numBarras=(int)(numBa/2);
Nodos.lienzo.superficie.setNumBarras(numBarras);
salida =new Barra[Nodos.lienzo.superficie.getNumBarras()];
Nodos.lienzo.superficie.barraTemp= new
Barra[2*Nodos.lienzo.superficie.getNumBarras()];

int b=0; //orden que se le asignarÃ¡ a la barra
for(int n=0;n<nd.length;n++){
   for(int k=0;k<nd[n].vinculo.length;k++){

//  int a=nd[n].getOrden();int f=nd[a].vinculo[k].getOrden(); //genera
barras pero no le gustan a Ernesto
int a=nd[n].getIndice();int f=nd[a].vinculo[k].getIndice();  //no
genera barra

   if (f>a) {Nodos.lienzo.superficie.barraTemp[b]=new
Barra(nd[a],nd[f]);  b++;  }
     }}
// ahora deben transferirse los datos de la barra temporal a la de
salida y asignarle el rden
if (b==numBarras){
 //le asignamos el Ãndice para llamarla
for (int n=0;n<Nodos.lienzo.superficie.getNumBarras();n++){
salida[n]=new Barra( );
salida[n]=Nodos.lienzo.superficie.barraTemp[n];
salida[n].setOrden(n);
salida[n].setIndice(n);
salida[n].setSeccion(Nodos.lienzo.superficie.seccionBarra);
salida[n].setMaterial(Nodos.lienzo.superficie.dMaterial);
}
if(Nodos.bAlerta)JOptionPane.showMessageDialog(  null, "EXITO: Array
de barras creadas");
//si existe una matriz de densidad Antigua, se vuelcan sus datos sobre
el array de barras
if(Nodos.lienzo.superficie.DFF !=null){
 for(int t=0; t<Nodos.lienzo.superficie.DFF.getNumFilas();t++) {
 salida[t].setDensiFuerza(Nodos.lienzo.superficie.DFF.getElemento(t,
t));
 }
 }
else{ }
}
```

```
else{ JOptionPane.showMessageDialog(  null, "ERROR en el cÃ¡lculo del
nÃºmero de barras");
Nodos.lienzo.superficie.barras=null; //destruimos el puntero asignado
por new Barras[]
}

return salida;

}

 public static Matriz matrizDensidad(Barra[] bar, Matriz Mat){
int a=0,b=0, salida=0;
int filas=bar.length; //que coincidira con
lienzo.superficie.gerNumeroBarras();
Matriz Den=new Matriz(filas, filas);
Den.x=new double[Nodos.lienzo.superficie.getNumBarras()][filas];
 for(int i=0;i<filas;i++){ for(int j=0;j<filas;j++){ //el numero de
filas y columnas es el mismo
        Den.setElemento(i, j, 0);  //se le asigna a todos los valores
cero
   }}
 if(filas==Mat.getNumFilas()){  //se hace una comprobaciÃ³n
 for(int i=0;i<filas;++i){ //barrido por el array de barras
     a=bar[i].getOrigen().getOrden(); //se localiza el orden del punto
que origina la barra segun el orden de la matriz de nudos ordenados
     b=bar[i].getFin().getOrden();   //se localiza el orden del
elemento que finaliza la barra segÃºn su posiciÃ³n en el array de
nudos ordenado

     for(int j=0;j<filas;++j){ //barrido por las fila de la  matiz Mat

if (Mat.getElemento(j, a)==1 && Mat.getElemento(j, b)==-1){
    bar[i].setOrdenMatrizDensidad(j);

Den.setElemento(bar[i].getOrdenMatrizDensidad(),bar[i].getOrdenMatrizD
ensidad(), bar[i].getDensiFuerzaF());}
                                 }
                                }
} else{ //en caso que de error se asigna el valor de densidad por
defecto
 JOptionPane.showMessageDialog(  null, "Error en el cÃ¡lculo de la
matriz de Densidades de Fuerza. Se asignara una densidad por defecto
para todas las barras");
     for(int i=0;i<filas;i++){ Den.setElemento(i, i,
Nodos.lienzo.superficie.densiFuerzaBarraForma   );
  }}
 return Den;
}
 public static Matriz asignarMasaNudos(Barra[] bar,boolean bC){
 Matriz Mas;
 Nudo[] temp= Nodos.lienzo.superficie.nudo.clone();
 int filas=Nodos.lienzo.superficie.nudo.length;
 for(int j=0;j<bar.length;j++){ bar[j].actualizar();}

//Nodos.lienzo.superficie.nudo=Nodos.lienzo.superficie.masaBarraToNudo
(bar,Nodos.lienzo.superficie.nudo.clone() );
Nodos.lienzo.superficie.nudo=Nodos.lienzo.superficie.masaBarraToNudo(b
ar,temp );
temp=Nodos.lienzo.superficie.OrdenarNudos(temp);
```

```
 //Ahora construimos la Matriz de masa que es cuadrada y solo con
terminos en la diagonal
 Mas=new Matriz(filas, filas);
 Mas.x=new double[filas][filas];
 for(int i=0;i<filas;i++){ for(int j=0;j<filas;j++){ //el numero de
filas y columnas es el mismo
       Mas.setElemento(i, j, 0);  //se le asigna a todos los valores
cero
   }}
 for(int j=0;j<filas;++j){
     Mas.setElemento(j, j,temp[j].getMasa(bC));
 }
  return Mas;
 }

 void actualizar(){  //recalcula todos los datos en funciÃ³n de los
datos introducidos

 this.longitud=this.vector.modulo;
 this.masa= this.seccion*this.densiMasa*this.longitud;
 this.fuerzAxial=this.densiFuerzaForma *this.longitud;
 if(this.seccion!=0)this.tension=this.fuerzAxial/this.seccion;
 if(this.limElas !=0)
this.seccionC=this.fuerzAxial*this.coeficienteSeguridad/this.limElas;
  if ( Nodos.bAlerta &&
this.seccionC>this.seccion)JOptionPane.showMessageDialog(  null,
"Alerta:SecciÃ³n de la barra insuficiente. No obstante, seguimos el
cÃ¡lculo por el cual el ratio de agotamiento serÃ¡ mayor que 1");
 try{
     this.densiFuerzaDina= this.modElas*this.seccion/this.longitud;
     this.ratioAgota=this.seccionC/this.seccion;

this.acortaInicial=this.fuerzAxial*this.longitud/(this.seccion*this.mo
dElas);
 } catch(Exception err) {JOptionPane.showMessageDialog(null,
"DivisiÃ³n por cero");}

         }}
```

Clase EjesCoordenadas

```
package nodos;

/**
 *
 * @author julio MuÃ±iz Padilla y Ernesto MuÃ±iz MartÃn
 *
 */
import java.awt.Graphics2D;
import java.awt.Graphics;
import java.awt.Point;
import java.awt.geom.GeneralPath;
import java.io.Serializable;

public class EjesCoordenadas implements Serializable  {
  //  GeneralPath paso;
    Graphics g;

    boolean boolEjes = false;
    double Ancho = 10;
    double Alto = 10;
    int Xo = 300;  //posicion en la pantalla
    int Yo = 300; //posicion e n la pantalla del origen
    double escala = 1.0;
    int anguloX = 0;
    int anguloY = 0;
    int anguloZ = 45;
    int cAlto = 10;
    int cAncho = 10;
    int x1 = 0;
    int y1 = 0;

    EjesCoordenadas(){};  //constructor por defecto

    public void setPosicion(int xo, int yo, double esc, int angX, int
angY, int angZ) {
        this.anguloX = angX;this.anguloY=angY;this.anguloZ=angZ;
        this.Xo = xo;
        this.Yo = yo;
        this.escala = esc;
    }

   void  dibujaEjes(Graphics g) {  //  GeneralPath paso = new
GeneralPath();
         if(this.boolEjes){
          Point pixelO=new Point();Point pixelF=new Point(); //pixelO
=pixel asociado al inicio de la linea pixelF al final
            Graphics2D g2 = (Graphics2D) g;
            g2.drawLine(Xo - (int) (Ancho / 2), Yo, Xo + (int) (Ancho
/ 2), Yo);//EjeX
            g2.drawLine(Xo, Yo - (int) (Alto / 2), Xo, Yo + (int)
(Alto / 2));//ejeY
            Punto pOrigen=new Punto(); pOrigen.setPunto(-500,0,0);
            pixelO=pantallaP(pOrigen, anguloX,anguloY,anguloZ);
            Punto pFinal= new Punto(); pFinal.setPunto(500,0,0);
            pixelF=pantallaP(pFinal,anguloX, anguloY,anguloZ);
```

```
            g2.drawLine(pixelO.x, pixelO.y, pixelF.x, pixelF.y);
//EjeZ

            g2.drawString("X",pixelF.x, pixelF.y);

           for (int i = 0; i < 100; i++) {  //dibujar la escala sobre
los ejes
                if (i == 0) { continue; }
                x1 = Xo + (int) (i * this.escala);
                g2.drawLine(x1, Yo - 4, x1, Yo + 4);
                g2.drawString(String.valueOf(Math.abs(i)), x1 - cAncho
/ 2, Yo + cAlto);
            }
            //parte negativa eje X
            for (int i = 0; i < 100; i++) {
                if (i == 0) {
                    continue;
                }
                x1 = Xo - (int) (i * this.escala);
                g2.drawLine(x1, Yo - 4, x1, Yo + 4);
                g2.drawString("-" + String.valueOf(Math.abs(i)), x1 -
cAncho / 2, Yo + cAlto);
            }
            //parte positiva eje Y
            for (int i = 0; i < 100; i++) {
                if (i == 0) {
                    continue;
                }
                y1 = Yo - (int) (i * this.escala);
                g2.drawLine(Xo - 4, y1, Xo + 4, y1);
                g2.drawString(String.valueOf(Math.abs(i)), Xo +
cAncho, y1 + cAlto / 2);
            }
            //parte negativa eje Y
            for (int i = 0; i < 100; i++) {
                if (i == 0) {
                    continue;
                }
                y1 = Yo + (int) (i * this.escala);
                g2.drawLine(Xo - 4, y1, Xo + 4, y1);
                g2.drawString("-" + String.valueOf(Math.abs(i)), Xo +
cAncho, y1 + cAlto / 2);
              }
  } }

    void setAncho(int anch) {
        this.Ancho = anch;
    }

    void setAlto(int alt) {
        this.Ancho = alt;
    }

    void setXo(int xo) {
        this.Xo = xo;
    }

    void setYo(int yo) {
        this.Yo = yo;
    }
```

```
    void setAnguloX(int angX) {
        this.anguloX = angX;
    }

    void setAnguloY(int angY) {
        this.anguloY = angY;
    }

    void setAnguloZ(int angZ) {
        this.anguloZ = angZ;
    }

    void setEscala(int esc) {
        this.escala = esc;
    }

    void setVisible(boolean vis) {
        this.boolEjes = vis;
    }

    private Point pantallaC(double Xp, double Yp, double Zp, double
angX, double angY,double angZ) {
        double alfa = angX * Math.PI / 180; //pasamos de grado a
radianes
        double beta = angY * Math.PI / 180;
        double gamma= angZ * Math.PI / 180;
        double esca = escala / 10;
        Point pixel = new Point();
        double k = 0;
        if (Xp > 0) {
            k = 1.0;
        } else {
            k = 1.41;
        }

        pixel.x = Xo + (int) (esca * (Yp*Math.cos(gamma) - k * Xp *
Math.sin(alfa)*Math.cos(beta)));
        pixel.y = Yo - (int) (esca * (k * Xp *
Math.cos(alfa)*Math.cos(beta) - Zp*Math.sin(gamma)));

        return pixel;

    }


      public Point pantallaP(Punto punto,double angX,double angY
,double angZ) {
         double esca = escala / 10;
        double alfa = angX * Math.PI / 180; //pasamos de grado a
radianes
        double beta = angY * Math.PI / 180;
        double gamma= angZ * Math.PI / 180;
       // double esca = escala / 10;
        double Xp, Yp, Zp;
        Xp = punto.getX();
        Yp = punto.getY();
        Zp = punto.getZ();
        Point pixel = new Point();
        double k = 0;
        // if(Xp>0)k=1.41; else k=0.705;
        if (Xp > 0) {
```

```
            k = 1;
        } else {
            k = 1.41;
        }

       pixel.x = Xo + (int) (esca * (Yp*Math.cos(gamma) - k * Xp *
Math.sin(alfa)*Math.cos(beta)));
                  pixel.y = ( Yo + (int) (esca * (k * Xp *
Math.cos(alfa)*Math.cos(beta) - Zp*Math.cos(gamma))));
        return pixel;
    }


}
```

Clase EleFinito

```
package nodos;

import java.io.Serializable;

/**
 * Elementos finitos que constituyen una superficie. EstÃ¡n
constituidos por
 * nudos enlazados mediante barras. se trabajan elementos finitos que
forman
 * triÃ¡ngulos, cuadrados y cuadrados con diagonales.
 *
 * @author julio solo estar desarrollado para elementos rectangulares.
19
 * FEBRERO 2014
 */
public class EleFinito implements Serializable {

    private int orden = 0;
    char tipo;  // t,c,d  triangulo, cuadrado, cuadrado diagonado
    Punto centroGeometrico;
    Nudo nudo[];
    Barra arista[]; //sera 4 en cuadrado, 6 en diagonal y 3 en
triangular
    static String etiqueta;//en caso de c y d el indice de los nudos
es 4, en caso de t 3
    int numNudo; // que forman el EleFinito
    int numArista; // que forman el EleFinito
    double dXa = 0;
    double dYa = 0;
    double dXb = 0;
    double dYb = 0;
    JuVector superficieA;
    JuVector superficieB; //solo en el rectÃ¡ngulo diagonado
    JuVector peso;
    JuVector carga;
    double area = 0; //mÃ³dulo de spuerficie
    double areaB = 0; //solo se usa en el rectÃ¡ngulo diagonado
    static double grosor = 0;
    static double densiMasa = 0;
    double densiFuerzaForma = 0;
    double densiFuerzaCalculo = 0;
    double masa = 0;

    EleFinito() {
    }

    ;

  EleFinito(char TipoEleFinito) {
        this.tipo = TipoEleFinito;
        superficieA = new JuVector(1, 1, 1);
        superficieB = new JuVector(0, 0, 0); //solo en el rectÃ¡ngulo
diagonado
        peso = new JuVector(0, 0, 0);
        carga = new JuVector(0, 0, 0);
        centroGeometrico = new Punto();
```

```
        switch (this.tipo) {
            case 't': {
                this.numNudo = 3;
                this.numArista = 3;
                nudo = new Nudo[3];
                arista = new Barra[3];
                for (int i = 0; i < 3; i++) {
                    this.nudo[i] = new Nudo();
                }
                for (int j = 0; j < 3; j++) {
                    this.arista[j] = new Barra();
                }
            }// nVinIn=6;break; //3 nudos * 2 vinculos por cada nudo
            case 'c': {
                this.numNudo = 4;
                this.numArista = 4;
                nudo = new Nudo[4];
                arista = new Barra[4];
                for (int i = 0; i < 4; i++) {
                    this.nudo[i] = new Nudo();
                }
                for (int j = 0; j < 4; j++) {
                    this.arista[j] = new Barra();
                }
            }// nVinIn=8;break; //4 nudos * 2 vinculos por cada nudo
            case 'd': {
                this.numNudo = 4;
                this.numArista = 6;
                nudo = new Nudo[4];
                arista = new Barra[6];
                for (int i = 0; i < 4; i++) {
                    this.nudo[i] = new Nudo();
                }
                for (int j = 0; j < 6; j++) {
                    this.arista[j] = new Barra();
                }
            }// nVinIn=12;break; // 4 nudos * 3 vinculos por cada nudo
        }
    }

    ////////////////////////////otras funciones complementarias
    void setOrden(int n) {
        this.orden = n;
    }

    int getOrden() {
        return this.orden;
    }

    char getTipo() {
        return this.tipo;
    }

    void setEtiqueta(String eti) {
        this.etiqueta = eti;
    }

    String getEtiqueta() {
        return this.etiqueta;
    }
```

```
    void setTipo(char tp) {
        this.tipo = tp;
    }

    char getTipoEleFinito() {
        return this.tipo;
    }

    double getLongXa() {
        return this.dXa;
    }

    double getLongYa() {
        return this.dYa;
    }

    double getLongXb() {
        return this.dXb;
    }

    double getLongYb() {
        return this.dYb;
    }

    void setDensiMasa(double den) {
        this.densiMasa = den;
    }

    //la siguiente funciÃ³n asigna los nudos a los elementos finitos,
si puede.
    void setGrosor(double gro) {
        this.grosor = gro;
    }

    double getGrosor() {
        return this.grosor;
    }

    double getArea() {
        return this.area;
    }

    double getDensiMasa() {
        return this.densiMasa;
    }

    void setNudos(Superficie sup, int indice, int numNudoX) {
//indice del nudo de la superficie y nÃºmero de nudos por filas

        if (sup.nudo[indice].tipoNudo == 'g') {

            switch (this.tipo) {
                case 't':

                case 'c': {
                    this.nudo[0] = sup.nudo[indice];
                    this.nudo[1] = sup.nudo[indice + 1];
                    this.nudo[2] = sup.nudo[indice + numNudoX];
                    this.nudo[3] = sup.nudo[indice + numNudoX + 1];
                    break;
```

```
                }

                case 'd': {
                    this.nudo[0] = sup.nudo[indice];
                    this.nudo[1] = sup.nudo[indice + 1];
                    this.nudo[2] = sup.nudo[indice + numNudoX];
                    this.nudo[3] = sup.nudo[indice + numNudoX + 1];
                    break;
                }
                default: {
                    this.nudo[0] = null;
                    this.nudo[1] = null;
                    this.nudo[2] = null;
                    this.nudo[3] = null;
                    break;
                }
            }
        }
////////////////////////////////fin de SetNudos
    }
//////////////////////////////////////////////////////

    void actualizar() {
        switch (this.getTipoEleFinito()) {
            case 'c': {
                Barra ladoXa;
                Barra ladoYa;
                JuVector vectorTemporal = new JuVector(1, 1, 1);
                ladoXa = new Barra(this.nudo[0], this.nudo[1]);
                ladoYa = new Barra(this.nudo[0], this.nudo[3]);
                this.superficieA.productoVectorial(ladoXa.vector,
ladoYa.vector);
                this.area = this.superficieA.modulo; //al final hay
que multiplicar x 0,25
                // nudo 1
                ladoXa = new Barra(this.nudo[1], this.nudo[0]);
                ladoYa = new Barra(this.nudo[1], this.nudo[2]);
                vectorTemporal.productoVectorial(ladoXa.vector,
ladoYa.vector);
                this.superficieA.sumaVectores(this.superficieA,
vectorTemporal);
                this.area = this.area + vectorTemporal.modulo;
                // nudo 2
                ladoXa = new Barra(this.nudo[2], this.nudo[1]);
                ladoYa = new Barra(this.nudo[2], this.nudo[3]);
                vectorTemporal.productoVectorial(ladoXa.vector,
ladoYa.vector);
                this.superficieA.sumaVectores(this.superficieA,
vectorTemporal);
                this.area = this.area + vectorTemporal.modulo;
                // nudo 3
                ladoXa = new Barra(this.nudo[3], this.nudo[2]);
                ladoYa = new Barra(this.nudo[3], this.nudo[0]);
                vectorTemporal.productoVectorial(ladoXa.vector,
ladoYa.vector);
                this.superficieA.sumaVectores(this.superficieA,
vectorTemporal);
                this.area = this.area + vectorTemporal.modulo;
                //ahora calculo los promedios
                this.area = 0.25 * this.area;
```

```
                this.superficieA.productoConstanteVector(0.25,
this.superficieA);

                break;
            }
            case 'd': {
                Barra ladoXa;
                Barra ladoYa;
                JuVector vectorTemporal = new JuVector(1, 1, 1);
                ladoXa = new Barra(this.nudo[0], this.nudo[1]);
                ladoYa = new Barra(this.nudo[0], this.nudo[2]);
                vectorTemporal =
vectorTemporal.productoVectorial(ladoXa.vector, ladoYa.vector);
                this.superficieA.productoConstanteVector(0.5,
vectorTemporal);
                this.area = this.superficieA.modulo; //
                //ladoXa=null;ladoYa=null;
                ladoXa = new Barra(this.nudo[2], this.nudo[0]);
                ladoYa = new Barra(this.nudo[2], this.nudo[3]);
                vectorTemporal =
vectorTemporal.productoVectorial(ladoXa.vector, ladoYa.vector);
                this.superficieB.productoConstanteVector(0.5,
vectorTemporal);
                this.areaB = this.superficieB.modulo; //
                break;
            }
            case 't': {
                this.superficieA = new JuVector(1, 1, 1);
                Barra ladoXa = new Barra(this.nudo[0], this.nudo[1]);
                this.arista[0] = ladoXa;
                Barra ladoYa = new Barra(this.nudo[0], this.nudo[2]);
                this.arista[1] = ladoYa;

                this.superficieA.productoVectorial(ladoXa.vector,
ladoYa.vector);
//vectorTemporal= this.superficieA;
//this.superficieA=new JuVector(2,2,2);//CONTROL CONTROL CONTROL
CONTROCONTROL CONTROL CONTROL CONTROCONTROL CONTROL CONTROL CONTRO
                this.superficieA.productoConstanteVector(0.5,
this.superficieA);
                this.area = this.superficieA.modulo; //
                break;
            }
        }
       // this.setMaterial(Nodos.lienzo.superficie.dMaterial);
        this.masa = area * grosor * densiMasa;
        this.setEtiqueta("EF" + String.valueOf(this.getTipo()) +
String.valueOf(this.getOrden()));
    }

//////////////////////////////////////////////////////
    void setMaterial(double[] mat) {
        this.densiMasa = mat[0];
    }
}
```

Clase JuVector

```
package nodos;

import java.io.Serializable;

/**
 *
 * @author julio MuÃ±iz 2012
 */
class JuVector  implements Serializable {
Punto origen;
Punto extremo;
int nComponentes=3;
double comp[];
double modulo;
double X;
double Y;
double Z;
double alfa;
double beta;
double gamma;
double anguloY;
double anguloX;
double anguloZ;
double moduloX;
double moduloY;
double moduloZ;


public void setOrigen(double Xo,double Yo, double Zo){
this.origen.X=Xo;this.origen.Y=Yo;this.origen.Z=Zo;}
public void setExtremo(double Xo,double Yo, double Zo){
this.extremo.X=Xo;this.extremo.Y=Yo;this.extremo.Z=Zo;}
public void setComponentes(double cx, double cy, double cz){
this.nComponentes=3;
comp =new double[3];
this.X=cx;this.Y=cy;this.Z=cz;
this.comp[0]=cx;this.comp[1]=cy;this.comp[2]=cz;}


 JuVector(){}; // constructor por defecto;

JuVector(double x, double y, double z){
this.nComponentes=3;
comp =new double[3];
this.comp[0]=x;this.comp[1]=y;this.comp[2]=z;
this.X=x;this.Y=y;this.Z=z;
this.modulo=Math.sqrt((this.X*this.X)+(this.Y*this.Y)+(this.Z*this.Z))
;
    if(this.modulo>0)this.alfa=(this.X)/this.modulo;
    if(this.modulo>0)this.beta=(this.Y)/this.modulo;
    if(this.modulo>0)this.gamma=(this.Z)/this.modulo;
  }


JuVector(int nCom){
this.nComponentes=nCom;
this.comp= new double[this.nComponentes];
for (int i=0;i<nCom;++i){ this.comp[i]=0;};
```

```
}
void setX(double cX){ this.X=cX;comp[0]=cX;};
void setY(double cY){ this.Y=cY;comp[1]=cY;};
void setZ(double cZ){ this.Z=cZ;comp[2]=cZ;};

JuVector(Punto Ori, Punto Ext){  // crea un vector tridimensional.
    origen=Ori; extremo=Ext;
    this.modulo=Math.sqrt((extremo.X-origen.X)*(extremo.X-
origen.X)+(extremo.Y-origen.Y)*(extremo.Y-origen.Y)+(extremo.Z-
origen.Z)*(extremo.Z-origen.Z));
    if(this.modulo>0)this.alfa=(extremo.X-origen.X)/this.modulo;
    if(this.modulo>0)this.beta=(extremo.Y-origen.Y)/this.modulo;
    if(this.modulo>0)this.gamma=(extremo.Z-origen.Z)/this.modulo;
    this.X=extremo.X-origen.X;
    this.Y=extremo.Y-origen.Y;
    this.Z=extremo.Z-origen.Z;
    this.comp= new double[3];
    this.comp[0]=this.X;this.comp[1]=this.Y;this.comp[2]=this.Z;
    this.moduloX=Math.sqrt(this.comp[1]*this.comp[1]+
this.comp[2]*this.comp[2] );
    this.moduloY=Math.sqrt(this.comp[0]*this.comp[0]+
this.comp[2]*this.comp[2] );
    this.moduloZ=Math.sqrt(this.comp[0]*this.comp[0]+
this.comp[1]*this.comp[1] );
    if(this.Y !=0)Math.atan(this.anguloX=(this.Z)/this.Y); else
Math.atan(this.anguloZ=(this.Z)/0.0001);
    if(this.X !=0)Math.atan(this.anguloY=(this.Z)/this.X); else
Math.atan(this.anguloZ=(this.Z)/0.0001);
    if(this.Y !=0)Math.atan(this.anguloZ=(this.X)/this.Y);  else
Math.atan(this.anguloZ=(this.X)/0.0001);

    }

public void setNumComponentes(int nCom){
this.nComponentes=nCom;
this.comp= new double[this.nComponentes];
for (int i=0;i<nCom;++i){ this.comp[i]=0;}; // inicializamos
}

JuVector getUnitario(){
JuVector salida= new JuVector(3);
salida.X=salida.comp[0]=this.alfa;
salida.Y=salida.comp[1]=this.beta;
salida.Z=salida.comp[2]=this.gamma;
return salida;
}

public int getNumComponentes(){
return this.comp.length;
}

public double getCompoX(){

return this.comp[0];
}
public double getCompoY(){
return this.comp[1];
}
public double getCompoZ(){
return this.comp[2];
```

```
}

public double getModulo(){

    return Math.sqrt((this.X*this.X) + (this.Y*this.Y) +
(this.Z*this.Z));
}
////////////////////////////////////suma de vectores///////////////
JuVector sumaVectores(JuVector a, JuVector b){
JuVector salida=new JuVector(a.getNumComponentes());
for(int
n=0;n<a.getNumComponentes();n++){salida.comp[n]=a.comp[n]+b.comp[n];
}
return salida;
}
////////////////////////////////////producto
escalar//////////////////////
public double productoEscalar(JuVector a, JuVector b){
return
a.getCompoX()*b.getCompoX()+a.getCompoY()*b.getCompoY()+a.getCompoZ()*
b.getCompoZ();};
////////////////////////////////producto vectorial
public JuVector productoVectorial(JuVector a, JuVector b){
JuVector salida=new JuVector(3);
salida.comp[0]=salida.comp[0]=(a.comp[1]*b.comp[2])-
(a.comp[2]*b.comp[1]);
salida.comp[1]=salida.Y=(a.comp[2]*b.comp[0])-(a.comp[0]*b.comp[2]);
salida.comp[2]=salida.Z=(a.comp[0]*b.comp[1])-(a.comp[1]*b.comp[0]);
return salida;
}

public JuVector productoConstanteVector(double cte,JuVector vec){
JuVector salida=new JuVector(vec.getNumComponentes());
for(int
u=0;u<vec.getNumComponentes();u++){salida.comp[u]=cte*vec.comp[u];}
return salida;
}

}
```

Clase Lienzo

```
/*
 * @author julio MuÃ±iz Padilla y Ernesto MuÃ±iz MartÃn
 */

package nodos;
import java.awt.*;
import java.awt.Graphics2D;
import java.awt.geom.GeneralPath;
import java.io.Serializable;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import static nodos.Paneli.hilo;


public class Lienzo extends Canvas implements  Serializable{
 private static final long serialVersionUID = 00001L; //los clases
serializables pueden guardarse como datos.
     Shape pos1,pos2;
     boolean bDibujoDeformado=false;
     boolean boolEleFinito=false;
     boolean frozen=false;
 Superficie superficie;
  static int nBarActiva=0;
  static int nEFActiva=0;
  boolean boolEjes, boolCoordenadas, boolBorrar, boolNudos,boolBarras,
boolPeriApoyo, boolPeriEsquina;
   boolean boolNumNudo=true, boolNumOrNudo,
boolNudosSalida,boolExisteExterno=true,
boolReacciones=false,boolAnima, boolVibra=false;
   boolean bXY=false; boolean bXZ=false; boolean bYZ=false;boolean
bXYZ=true;
   boolean bAgota=false;
    int Xo, Yo, x1, y1;
    String cCoordenadas;
    int Ancho, Alto;
    double AnchoRect, AltoRect;
    int cAncho, cAlto;
    double escala = 10;
    double escalaVibra=10, escalaX=1,escalaY=1,escalaZ=1; //escala de
la imagen deformada PARA CADA NODO DE VIBRACIÃ“N.
    int nModoVibra=0; //modo de vibraciÃ³n para la imagen deformada.
    double escalaT=1;
    int giroZ = 45;
    int giroX=45;
    int giroY=45;
    double angX;
    double angY;
    double angZ;
    double alfa, beta, gamma;
    Color color;
    Color colorFondo;
```

```
    String cangulo;
    EjesCoordenadas ejes;
    Graphics g;
    public Lienzo() {
        cCoordenadas = new String();
        cangulo = new String();
        boolEjes = true;
        boolCoordenadas = false;
        boolBorrar = false;
        boolNudos = false;
        boolNumNudo = false;
        boolNumOrNudo = false;
        boolBarras=false;
        boolVibra=false;
        boolPeriApoyo = false;
        boolPeriEsquina=false;
        boolAnima=false;
        Xo = 100;
        Yo = 100;
        cAncho = 10;
        cAlto = 10;
        superficie=new Superficie();
        this.setBackground(Color.WHITE);
        colorFondo=this.getBackground();
        this.setSize(1500, 1000);
        Ancho = this.getWidth() - 100;
        Alto = this.getHeight() - 100;
        this.cCoordenadas = "(X,Y)"; // = new String();
        this.setVisible(true);
        ejes = new EjesCoordenadas();
        ejes.setPosicion(this.Xo,this.Yo, this.escala, 0, 0,
this.giroZ);
            }

 @Override
    public boolean mouseDown(Event event, int x, int y) {
        Xo = event.x;
        Yo = event.y;
        //if(frozen){
frozen=false;arrancarHilo();}else{frozen=true;pararHilo();}
        // repaint();
        return true;
    }

 @Override
    public boolean mouseDrag(Event event, int x, int y) {
        cCoordenadas=dameCoordenada(true);
       if(x> ejes.Xo  )giroX=giroX-1;
       if( x< ejes.Xo )giroX=giroX+1;
       if(y>ejes.Yo)giroY=giroY-1;
       if(y<ejes.Yo)giroY=giroY+1;
       repaint();
        return false;
    }

 @Override
 public boolean mouseUp(Event event, int x, int y) { return false; }

 public void paint(Graphics g) {
     super.paint(g);
        Graphics2D g2 = (Graphics2D) g;
```

```
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
        Shape sh;
        this.setBackground(colorFondo);
        ejes.setPosicion(this.Xo,this.Yo, this.escala, 0, 0,
this.giroZ);
        ejes.dibujaEjes(g2);

  if(boolBarras)dibujaBarras(g2,superficie.nudo,Color.ORANGE);
  if(boolPeriApoyo || boolPeriEsquina)  dibujarPerimetro(g2);  //
operadores  || or   y && and

  if(boolCoordenadas)  ponCoorApoyos(g2);
  if(boolEjes);{ ejes.setPosicion(Xo, Xo, escala, Xo, Xo, giroZ); }
//dibuja los nodos para eleFinitos cuadradas y cuadradas diagonadas.
  if(boolNudos)dibujaNudo(g2,superficie.nudo, Color.MAGENTA);
  if(boolNudosSalida) dibujaNudoSalida(g2); //196
  if(boolReacciones)dibujaReacciones(g2); //149

if(boolEleFinito)dibujaEleFinito(g2,Paneli.numEF,this.superficie.tipoE
leFinito,Color.YELLOW); //149
  //if(boolExisteExterno)dibujaNudoExterno( g2); //165
  if(boolVibra){try {
      mueve( g2,Paneli.m,this.escalaT);
           } catch (InterruptedException ex) {
               JOptionPane.showMessageDialog(  null, "ERROR de
animacion");;
           }
  }
  if(boolBorrar) borrar(g2); //debe ir el Ãºltimo para ejecutar la
limpieza de todos los elementos
  if(bDibujoDeformado)
this.pos2=dibujaBarras(g2,superficie.nudo,Color.ORANGE);
  else this.pos1=dibujaBarras(g2,superficie.nudo,Color.ORANGE);
 }

 void borrar(Graphics gra){
  gra.clearRect(0, 0, this.getWidth(), this.getHeight());
 }

  GeneralPath dibujaBarras(Graphics2D gr,Nudo[] aN, Color c){
   gr.setStroke( new BasicStroke(  1, 1,0 ));
 int n,i, j;
 gr.setColor(c);
  GeneralPath paso = new GeneralPath();
// dibujar los vÃƒÂnculos o barras a partir de cada nudo.
   Point pixelNudo; pixelNudo= new Point();
   Point pixelVinculo;pixelVinculo=new Point();
  for(n=0;n<aN.length;n++){
   int nor=aN[n].getOrden(); //obtendo en orden original de cada nudo.
  // pixelNudo=
pantallaC(superficie.nudo[n].coordenada.X,superficie.nudo[n].coordenad
a.Y,superficie.nudo[n].coordenada.Z,giroX,giroY,giroZ);
     pixelNudo= pantallaP(aN[nor].coordenada,giroX,giroY,giroZ);
    for( i=0;i<aN[nor].getNumBarrasT();i++){
    paso.moveTo(pixelNudo.x, pixelNudo.y);
    int vin=aN[nor].vinculo[i].getOrden(); //calculo el indice del
nudo vinculado
    pixelVinculo=pantallaP(aN[vin].coordenada,giroX,giroY,giroZ );
    paso.lineTo(pixelVinculo.x, pixelVinculo.y);
```

```
   }  }
      paso.closePath();
      pos1=paso; //introducido el 31 12 2012
      gr.draw(paso);
      //ahora resalto la barra seleccionada
    gr.setStroke( new BasicStroke(  2, 1,0 ));
 resaltarBarra(gr);
     gr.setStroke( new BasicStroke(  1, 1,0 ));
  //dibujar agotamiento.
  if(bAgota) mostrarAgotamiento(gr);
  return paso;
    }

 void dibujaReacciones(Graphics2D gra){
 Point pixelA=new Point(),pixelR=new Point();
  Punto P=new Punto();
 for(int n=0;n<Nodos.lienzo.superficie.numApoyos;n++){
 pixelA=
pantallaP(Nodos.lienzo.superficie.nuOrdenado[n].coordenada,giroX,giroY
,giroZ);

pixelR=pantallaP(P.restar(Nodos.lienzo.superficie.nuOrdenado[n].coorde
nada,Nodos.lienzo.superficie.aReac[n].escalar(0.001)),giroX,giroY,giro
Z);
 gra.setColor(Color.RED);
 gra.setStroke( new BasicStroke(  3, 1,1 ));
 gra.drawLine(pixelA.x,pixelA.y,pixelR.x,pixelR.y);
 }
 gra.setStroke( new BasicStroke(  1, 1,0 ));
 }


void dibujaNudoExterno( Graphics2D gra){
  Nudo ndEx=new Nudo();
  Nudo ndFin=new Nudo();
   Point pixelN=new Point();
   gra.setColor(Color.GREEN);
    if(Nodos.lienzo.superficie.nuExterno !=null ){
       ndEx=Nodos.lienzo.superficie.nuExterno;
   }
     pixelN=pantallaP(ndEx.coordenada,giroX,giroY,giroZ);
   if(ndEx.getApoyo()) gra.fillOval(pixelN.x, pixelN.y, 6, 6); else
gra.drawOval(pixelN.x, pixelN.y, 6, 6);
   if(boolNumNudo ) gra.setColor(Color.DARK_GRAY);
gra.drawString(String.valueOf(ndEx.getOrden()), pixelN.x, pixelN.y);

  for(int
k=0;k<Nodos.lienzo.superficie.nudo[Nodos.lienzo.superficie.nudo.length
-1].vinculo.length;k++){

ndFin=Nodos.lienzo.superficie.nudo[Nodos.lienzo.superficie.nudo.length
-1].vinculo[k];
   gra.drawLine(pantallaP(ndEx.getCoordenada(),giroX,giroY,giroZ).x,
pantallaP(ndEx.getCoordenada(),giroX,giroY,giroZ).y,pantallaP(ndFin.ge
tCoordenada(),giroX,giroY,giroZ).x,
pantallaP(ndFin.getCoordenada(),giroX,giroY,giroZ).y);
     }

 }
////////////////////////////////////////////////////////////////////
///
```

```
void dibujaEleFinito(Graphics2D gra, int n,char tipoMalla, Color c){
Point[] esquina; int or=0;
GeneralPath elemento = new GeneralPath();
int nEsquinas=0;
if(tipoMalla=='c'||tipoMalla=='d') nEsquinas=4;
if(tipoMalla=='t') nEsquinas=3;
 esquina = new Point[nEsquinas];
 for(int u=0;u<nEsquinas;u++){
    or=u;

esquina[u]=pantallaP(superficie.eleFinito[n].nudo[or].coordenada,giroX
,giroY,giroZ);}
elemento.moveTo(esquina[0].x,esquina[0].y);
for(int s=1;s<nEsquinas;s++){elemento.lineTo(esquina[s].x,
esquina[s].y);}
elemento.lineTo(esquina[0].x, esquina[0].y);
elemento.closePath();
gra.setColor(c);
gra.fill(elemento);
}

 void dibujaNudo(Graphics2D gra, Nudo[] nu, Color c ) {
       for (int n = 0; n < superficie.nudo.length ; n++) {
        gra.setColor(c);
           Point pixelN =new Point();
        pixelN=pantallaP(nu[n].coordenada, giroX,giroY,giroZ);
        if(nu[n].externo) gra.setColor(Color.GREEN); else
gra.setColor(Color.MAGENTA);
        if(nu[n].carga.getModulo() !=0){ gra.setColor(Color.BLUE);
gra.fillOval(pixelN.x, pixelN.y, 6, 6);}
        if(nu[n].getApoyo() ){gra.fillOval(pixelN.x, pixelN.y, 6, 6);
}else{ gra.drawOval(pixelN.x, pixelN.y, 6, 6);}
        if(boolNumNudo ) {gra.setColor(Color.DARK_GRAY);
gra.drawString(String.valueOf(nu[n].getOrden()), pixelN.x,
pixelN.y);}//se numeran por el parÃƒÆ'Ã‚Â¡metro orden,
            //  if(boolNumNudo && nuOrdenado!=null){
gra.setColor(Color.{();
gra.drawString(String.valueOf(nuOrdenado[n].getOrden()), pixelN.x-16,
pixelN.y);}
            //   if(boolNumOrNudo && this.nuOrdenado !=null) {
gra.setColor(Color.BLUE);
gra.drawString(String.valueOf(superficie.nudo[n].getIndice()),
pixelN.x, pixelN.y+16);}//se numeran por el parÃƒÆ'Ã‚Â¡metro orden,

              }
    }

////////////////////////////////////////////////////////////////////
//////////
        void dibujaNudoSalida(Graphics2D gra) {
                {
           // for (int n = 0; n < this.superficie.nudo.length ; n++) {
               for (int n = 0; n < superficie.nuSalida.length ; n++) {
                gra.setColor(Color.MAGENTA);
           Point pixelN =new Point();
        pixelN=pantallaP(this.superficie.nuSalida[n].coordenada,
giroX,giroY,giroZ);
         //    pixelN=pantallaP(NuOrdenado[n].coordenada,
giroX,giroY,giroZ);
            //    g2.drawOval(pixelN.x, pixelN.y, 6, 6);
```

```
        if(superficie.nudo[n].externo) gra.setColor(Color.GREEN); else
gra.setColor(Color.MAGENTA);
        if(this.superficie.nuSalida[n].carga.getCompoZ()!=0){
gra.setColor(Color.BLUE); gra.fillOval(pixelN.x, pixelN.y, 6, 6);}
        if(this.superficie.nuSalida[n].getApoyo())
gra.fillOval(pixelN.x, pixelN.y, 6, 6); else    gra.drawOval(pixelN.x,
pixelN.y, 6, 6);
        if (boolNumNudo ) {gra.setColor(Color.DARK_GRAY);
gra.drawString(String.valueOf(this.superficie.nuSalida[n].getOrden()),
pixelN.x, pixelN.y);}//se numeran por el parÃƒÂ¡metro orden,
        if(superficie.nudo[n].getApoyo() && superficie.nudo[n].externo
){gra.setColor(Color.GREEN); gra.fillOval(pixelN.x, pixelN.y, 6, 6);
}else{gra.setColor(Color.GREEN);  gra.drawOval(pixelN.x, pixelN.y, 6,
6);}
                //   if (boolNumNudo && nuOrdenado!=null){
gra.setColor(Color.GREEN);
gra.drawString(String.valueOf(nuOrdenado[n].getOrden()), pixelN.x-16,
pixelN.y);}
             //  if (boolNumOrNudo) {  gra.setColor(Color.BLUE);
gra.drawString(String.valueOf(NuOrdenado[n].getOrden()), pixelN.x,
pixelN.y+16);}//se numeran por el parÃƒÂ¡metro orden,
             //    if (boolNumOrNudo) {  gra.setColor(Color.BLUE);
gra.drawString(String.valueOf(NuOrdenado[this.superficie.nudo[n].getOr
den()].getOrden()), pixelN.x, pixelN.y+16);}//se numeran por el
parÃƒÂ¡metro orden,
         if (boolNumOrNudo && this.superficie.nuOrdenado !=null) {
gra.setColor(Color.BLUE);
gra.drawString(String.valueOf(this.superficie.nuSalida[n].getIndice())
, pixelN.x, pixelN.y+16);}//se numeran por el parÃƒÂ¡metro orden,

               }}
    }

    void dibujaImagen(Image imagen, Graphics g) {
        g = getGraphics();
        g.drawImage(imagen, 0, 0, this);
    }

    public String dameCoordenada(boolean control) {
        String retorno = new String();
        retorno="          ";
        if (control) {
            retorno = "(" + Integer.toString(Xo) + "," +
Integer.toString(Yo) + ")";
        } else {
            retorno = "                                                  ";
        }
        return retorno;
    }

    public Point pantallaP(Punto punto,double angX,double angY ,double
angZ) {
        alfa = angX * Math.PI / 180; //pasamos de grado a radianes
        beta = angY * Math.PI / 180;
        gamma= angZ * Math.PI / 180;
        double esca = escala / 10;
        double Xp, Yp, Zp;
        Xp = punto.getX();
        Yp = punto.getY();
        Zp = punto.getZ();
        Point pixel = new Point();
```

```
        double k = 0;
        // if(Xp>0)k=1.41; else k=0.705;
        if (Xp > 0) {
            k = 1;
        } else {
            k = 1.41;
        }
    if(bXYZ){
      pixel.x = Xo + (int) (esca * (Yp - k * Xp * Math.sin(alfa)));
      pixel.y = Yo+(int) (esca * (k * Xp *
Math.cos(alfa)*Math.cos(beta) - Zp*Math.sin(gamma)));
    }
     if(bXY){
      pixel.x = Xo + (int) (esca * (Yp - k * Xp * Math.sin(alfa)));
      pixel.y = Yo+(int) (esca * (k * Xp *
Math.cos(alfa)*Math.cos(beta)));
    }
     if(bXZ){
      pixel.x = Xo + (int) (esca * (- k * Xp * Math.sin(alfa)));
      pixel.y = Yo+(int) (esca * (k * Xp *
Math.cos(alfa)*Math.cos(beta) - Zp*Math.sin(gamma)));
    }
     if(bYZ){
      pixel.x = Xo + (int) (esca * (Yp));
      pixel.y = Yo+(int) (esca * (- Zp*Math.sin(gamma)));
    }

        return pixel;
    }



 void dibujarPerimetro(Graphics2D g){ //no usada en esta versiÃ³n
   }

void  ponCoorApoyos(Graphics2D gra){
    Point  pixelN;
 gra.setColor(Color.BLUE);
 for(int n=0;n< this.superficie.nudo.length;n++){
     if(this.superficie.nudo[n].getApoyo()){
  pixelN =new Point();

pixelN=pantallaP(this.superficie.nudo[n].coordenada,giroX,giroY,giroZ)
;

 gra.drawString("("+this.superficie.nudo[n].toPrint(),pixelN.x,
pixelN.y-3);

     }
 }
}

void mostrarAgotamiento(Graphics2D gra){
for(int n=0;n<superficie.barras.length;n++){
if (Nodos.lienzo.superficie.barras !=null){
 Nudo ndEx=new Nudo(); ndEx=Nodos.lienzo.superficie.barras[n].origen;
 Nudo ndFin=new Nudo(); ndFin=Nodos.lienzo.superficie.barras[n].fin;
 double rat=Nodos.lienzo.superficie.barras[n].ratioAgota;
 if(rat>=1) gra.setColor(Color.RED);
 if(rat>0.8 && rat <1)gra.setColor(Color.ORANGE);
 if(rat>0.6 && rat <0.8)gra.setColor(Color.YELLOW);
```

```
 if(rat>0.4 && rat <0.6)gra.setColor(Color.GREEN);
 if(rat>0.2 && rat <0.4)gra.setColor(Color.BLUE);
 if(rat<0.2)gra.setColor(Color.BLACK);
 gra.drawLine(pantallaP(ndEx.getCoordenada(),giroX,giroY,giroZ).x,
pantallaP(ndEx.getCoordenada(),giroX,giroY,giroZ).y,pantallaP(ndFin.ge
tCoordenada(),giroX,giroY,giroZ).x,
pantallaP(ndFin.getCoordenada(),giroX,giroY,giroZ).y);
}
}
}
//////////////////////////////////////////
//////////////////////////////////////////
 void resaltarBarra( Graphics2D gra){
if (Nodos.lienzo.superficie.barras !=null){
int ordBar=this.nBarActiva;
 Nudo ndEx=new Nudo();
ndEx=Nodos.lienzo.superficie.barras[ordBar].origen;
 Nudo ndFin=new Nudo();
ndFin=Nodos.lienzo.superficie.barras[ordBar].fin;
 double rat=Nodos.lienzo.superficie.barras[ordBar].ratioAgota;
 if(rat>=1) gra.setColor(Color.RED);
 if(rat>0.8 && rat <1)gra.setColor(Color.ORANGE);
 if(rat>0.6 && rat <0.8)gra.setColor(Color.YELLOW);
 if(rat>0.4 && rat <0.6)gra.setColor(Color.GREEN);
 if(rat>0.2 && rat <0.4)gra.setColor(Color.BLUE);
 if(rat<0.2)gra.setColor(Color.BLACK);
 gra.setStroke( new BasicStroke(  2, 1,0 ));
 gra.drawLine(pantallaP(ndEx.getCoordenada(),giroX,giroY,giroZ).x,
pantallaP(ndEx.getCoordenada(),giroX,giroY,giroZ).y,pantallaP(ndFin.ge
tCoordenada(),giroX,giroY,giroZ).x,
pantallaP(ndFin.getCoordenada(),giroX,giroY,giroZ).y);
}

 }
 ////////////////////////////////////////////////
 void guardarImagen(){
  File fichero = new File("foto.jpg");
 String formato = "jpg";
  BufferedImage imagen = new BufferedImage(100, 100,
                        BufferedImage.TYPE_INT_RGB);
 try {
                 ImageIO.write(imagen, formato, fichero);
            } catch (IOException e) {
                 System.out.println("Error de escritura");
            }
 }

 void mueve( Graphics2D gra, int m, double esT) throws
InterruptedException {
             // hilo=new Thread();
                double T=1000000/superficie.valoresPropios[m]; //valor
de la frecuencia

Paneli.lPeriodo.setText("T(ms)="+String.valueOf(Math.round(T))+"
Esc.Tiempo (x)"+String.valueOf(esT));
                long tiempo=1000; tiempo=Math.round(T*escalaT);
                if(tiempo<1)tiempo=1;

              // while(Thread.currentThread()==hilo){
                for(int t=0;t<30;t++){
```

```
                    try{
                      gra.draw(pos1);
                      hilo.sleep(tiempo);
                    gra.clearRect(0, 0, this.getWidth(),
this.getHeight());
                    gra.draw(pos2);
                     hilo.sleep(tiempo);
                     gra.clearRect(0, 0, this.getWidth(),
this.getHeight());
                      }   catch(InterruptedException ex){ break;  }
                   }
                   superficie.restaura();
                hilo=null;

}////////////////////////////////////////////////////////////()
{}

 void arrancarHilo(){
 if(Paneli.bAnima) if(hilo==null) hilo=new Thread();
 };
 void pararHilo(){hilo=null;};

}  // fin de la clase Lienzo.
```

Clase Matriz

```
// Julio MuÃ±iz Padilla diciembre 2012-Agosto 2013


package nodos;

import java.io.Serializable;

public class Matriz  implements Cloneable,Serializable
{  //el mÃ©todo debe ser croneable para hacer copias de campo a campo,
usando
   public int n =0;      //dimensiÃ³n  originalmente public
//punteros. Si no es asÃ, el procedimiento se relantiza al infinito
  double[][] x;  //private en el original
  int maxIter=300;
  public int numFilas;
  public int numColumnas;
////////////////////////////////////constructor matrices
cuadradas/////////////////////////////////////////
    public Matriz(int n) {   // constructor por defeto para matrices
cuadradas de n x n
        this.n=n;
        if (n !=0) { this.numFilas=n;  this.numColumnas=n; }

        x=new double[numFilas][numColumnas];
        for(int i=0; i<numFilas; i++){ for(int j=0; j<numColumnas;
j++){ x[i][j]=0; } }
                        }
 ////////////////////////////////// constructor matrices no
cuadradas/////////////////////////////////////////
 public Matriz(int nF, int nC){
if(nF==nC)n=nC;else n=0;  //una matriz cuyo n=0 no es cuadrada. Si n>0
es cuadrada.
this.numFilas=nF;
this.numColumnas=nC;
x= new double[this.numFilas][this.numColumnas];
for(int i=0; i< numFilas; i++) {for(int j=0; j< numColumnas; j++)
{this.x[i][j]=0;} }
}
/////////////////////////////////////////////fin//////////////////////////
////////////////////////////////////
public Matriz(double[][] x) {
        this.x=x;
         this.numFilas=this.x.length;
         this.numColumnas=this.x[numFilas].length;
         if(numFilas==numColumnas)n=numFilas;else n=0;
   }
//////// Fin de las funciones de
construcciÃ³n////////////////////////////////////////////////
 void setMaxIter(int max){this.maxIter=max;}

 public Matriz clone(){
        Matriz obj=null;
        try{
            obj=(Matriz)super.clone();
```

```
            }catch(CloneNotSupportedException ex){
             System.out.println(" no se puede duplicar");
            }
          //aquÃ estÃ¡ la clave  para clonar la matriz bidimensional
          obj.x=(double[][])obj.x.clone();
          for(int i=0; i<obj.x.length; i++){
              obj.x[i]=(double[])obj.x[i].clone();
          }
          return obj;
          }

  Matriz verificaMatriz(Matriz a){
  int nc=a.numColumnas;
  int nf=a.numFilas;
      Matriz res=new Matriz(1,nc);
  //for(int j=0;j<res.numColumnas;j++) res.x[1][j]=0;
  for(int j=0;j<nc;j++){
  for(int k=0;k<nf;k++){
     if(a.x[k][j] !=0.0) res.x[0][j]++;
  }
  }
  return res;
  }
   //
  void setElemento(int f,int c,double valor){
  //if(f<this.numFilas && c<this.numColumnas)
      this.x[f][c]=valor;
  }

  double getElemento(int f, int c){
  double salida=0.0;
  //if(f<this.numFilas && c<this.numColumnas)
  salida =this.x[f][c];
  return salida;
  }
  ////////////////////////////////////////////////////
  int getNumFilas(){
  return this.numFilas;
  }

  int getNumColumnas(){
  return this.numColumnas;
  }


  // funciÃ³n ausiliar que traza la matriz
      double traza(){
          double tr=0.0;
          for(int i=0; i<this.numFilas; i++){for (int
  j=0;j<this.numColumnas;j++){  tr+=x[i][j];}}

          return tr;
      }
  //suma de dos matrices
   public   Matriz suma(Matriz a, Matriz b){
       Matriz resultado=new Matriz(a.numFilas, a.numColumnas);
         if(a.numFilas==b.numFilas && a.numColumnas==b.numColumnas){

              for(int i=0; i<a.numFilas; i++){
              for(int j=0; j<a.numColumnas; j++){
                  resultado.x[i][j]=a.x[i][j]+b.x[i][j];
```

```
            }
        }} else{System.out.println("las matricies no son de idÃ©nticas
dimensiones ");}
        return resultado;
    }
  public   Matriz resta(Matriz a, Matriz b){
     Matriz resultado=new Matriz(a.numFilas, a.numColumnas);
       if(a.numFilas==b.numFilas && a.numColumnas==b.numColumnas){

            for(int i=0; i<a.numFilas; i++){
            for(int j=0; j<a.numColumnas; j++){
                resultado.x[i][j]=a.x[i][j]-b.x[i][j];
            }
        }} else{System.out.println("las matricies no son de idÃ©nticas
dimensiones ");}
        return resultado;
    }


//////////////////////////////////producto de dos matrices
/*
  public  Matriz producto(Matriz Ma, Matriz Mb){
        Matriz resultado=new Matriz(Ma.numFilas, Mb.numColumnas);
        if (Ma.numColumnas==Mb.numFilas){
        for(int i = 0; i < Ma.numFilas; i++){
        for(int j = 0; j<Mb.numColumnas;j++) {
             for(int k=0; k<Mb.numColumnas;k++){
              resultado.x[i][j]+=Ma.x[i][k]*Mb.x[k][j];}}}

        }

        else System .out.println("No se pueden multiplicar las
matrices");
        return resultado;
    }  */

  public  Matriz producto(Matriz Ma, Matriz Mb){
        Matriz resultado=new Matriz(Ma.getNumFilas(),
Mb.getNumColumnas());
        if (Ma.numColumnas==Mb.numFilas){
        for(int i = 0; i < Ma.numFilas; i++){
        for(int j = 0; j<Mb.numColumnas;j++) {
             for(int k=0; k<Ma.numColumnas;k++){
              resultado.x[i][j]+=Ma.x[i][k]*Mb.x[k][j];}}}

        }

        else System .out.println("No se pueden multiplicar las
matrices");
        return resultado;
    }



//producto de una matriz por un escalar
    public Matriz producto(Matriz a, double d){
        Matriz resultado=new Matriz(a.n);
        for(int i=0; i<a.n; i++){
            for(int j=0; j<a.n; j++){
                resultado.x[i][j]=a.x[i][j]*d;
            }
```

```
        }
        return resultado;
    }
//producto de un escalar por una matriz
 public   Matriz producto(double d, Matriz a){
        Matriz resultado=new Matriz(a.n);
        for(int i=0; i<a.n; i++){
            for(int j=0; j<a.n; j++){
                resultado.x[i][j]=a.x[i][j]*d;
            }
        }
        return resultado;

    }
//producto de un vector fila por una matriz da un vector fila (1xn)
(nxn)= (1xn)
 public  JuVector producto(JuVector v, Matriz a){
        int nu=v.nComponentes;
        JuVector b =new JuVector(v.getNumComponentes());
        b.setNumComponentes(nu);
         //dimensiÃ³n
        for(int j=0; j<nu; j++){
            for(int k=0; k<nu; k++){

                b.comp[j]+=v.comp[k]*a.x[k][j];
            }
        }
        return b;
    }
//producto de una matriz por un vector columna (nxn) (nx1)= (nx1)
public   JuVector producto(Matriz a, JuVector v){
        int nu=v.nComponentes;  //dimensiÃ³n
        JuVector b=new JuVector(v.getNumComponentes());
        b.setNumComponentes(nu);
        for(int i=0; i<nu; i++){
            for(int k=0; k<nu; k++){
                b.comp[i]+=a.x[i][k]*v.comp[k];
            }
        }
        return b;
    }// esta parte me estÃ¡ dando errores, no sÃ© por quÃ©. Julio 28
de enero.

//determinante de una matriz
    double determinante(){
        Matriz a=(Matriz)clone();

        for(int k=0; k<n-1; k++){
            for(int i=k+1; i<n; i++){
                for(int j=k+1; j<n; j++){
                    a.x[i][j]-=a.x[i][k]*a.x[k][j]/a.x[k][k];
                }
            }
        }
        double deter=1.0;
        for(int i=0; i<n; i++){
            deter*=a.x[i][i];
        }
        return deter;
    }
//matriz inversa
```

```
  public   Matriz inversa(Matriz d){
        int n=1;
       int fil=d.numFilas; // la matriz debe de tener igual nÃºmero de
filas que de columnas
        int col=d.numColumnas;
    // Matriz a=(Matriz)d.clone();

        Matriz a=new Matriz(fil,col);
        a=(Matriz)d.clone();
       /*
       for(int i=0;i<fil;i++){
        for(int j=0;j<col;j++){
         a.x[i][j]=d.x[i][j];
        }}
       */
      Matriz b=new Matriz(fil,col);   //matriz de los tÃ©rminos
independientes
      Matriz c=new Matriz(fil,col);   //matriz de las incÃ³gnitas

      if(fil!=col){n=0;System.out.print("Las matrices que no son
cuadradas no tienen inversa");return d;}
        else{ n=d.numFilas;

//matriz unidad
        for(int i=0; i<n; i++){
            b.x[i][i]=1.0;
        }
//transformaciÃ³n de la matriz y de los tÃ©rminos independientes
        for(int k=0; k<n-1; k++){
            for(int i=k+1; i<n; i++){
//tÃ©rminos independientes
                for(int s=0; s<n; s++){
                    b.x[i][s]-=a.x[i][k]*b.x[k][s]/a.x[k][k];
                }
//elementos de la matriz
                for(int j=k+1; j<n; j++){
                    a.x[i][j]-=a.x[i][k]*a.x[k][j]/a.x[k][k];
                }
            }
        }
//cÃ¡lculo de las incÃ³gnitas, elementos de la matriz inversa
        for(int s=0; s<n; s++){
            c.x[n-1][s]=b.x[n-1][s]/a.x[n-1][n-1];
            for(int i=n-2; i>=0; i--){
                c.x[i][s]=b.x[i][s]/a.x[i][i];
                for(int k=n-1; k>i; k--){
                    c.x[i][s]-=a.x[i][k]*c.x[k][s]/a.x[i][i];
                }
            }
       }
       return c;
      }
       // return c;
    }
//matriz traspuesta
public    Matriz traspuesta(Matriz a){
   Matriz resultado = new Matriz(a.numColumnas, a.numFilas);
   Matriz copiaA=new Matriz( a.numFilas,a.numColumnas);
   for(int i=0;i<a.numFilas;i++){for(int
j=0;j<a.numColumnas;j++){copiaA.x[i][j]=a.x[i][j];}}
   for (int i=0;i<a.numFilas;i++){
```

```
        for(int j=0;j<a.numColumnas;j++){
            resultado.x[j][i]=copiaA.x[i][j];
        }
    }

        return resultado;
    }
//polinomio caracterÃstico

    double[] polCaracteristico(){
        Matriz pot=new Matriz(n);
//matriz unidad con todos los elementos de la diagonal igual a 1.00
        for(int i=0; i<n; i++){
            pot.x[i][i]=1.0;
        }
        double[] p=new double[n+1];
        double[] s=new double[n+1];
        for(int i=1; i<=n; i++){
            pot=this.producto(pot, this);
            s[i]=pot.traza();
        }
        p[0]=1.0;
        p[1]=-s[1];
        for(int i=2; i<=n; i++){
            p[i]=-s[i]/i;
            for(int j=1; j<i; j++){
                p[i]-=s[i-j]*p[j]/i;
            }
        }
        return p;
    }
//    Vector de valores propios  sobre la matriz.
// la salida es un array de una sola dimensiÃ³n llamado valores.
// Se debe poner un lÃmite a las interacciones, por ejemplo
maxIter=1000
 public  Matriz valoresPropios(double[] valores, int maxIter)throws
ValoresExcepcion{
        final double CERO=1e-8; //declaro el valor final para que se
comporte como una constante incambiable.
        double maximo, tolerancia, sumsq;
        double x, y, z, c, s;
        int contador=0;   //pongo a 0 el contador que tomarÃ¡ com
mÃ¡ximo maxIter;
        int i, j, k, l;
        Matriz a=(Matriz)clone();      //copio los valores de la
matriz para compararlos luego
        Matriz p=new Matriz(n);
        Matriz q=new Matriz(n);
//matriz unidad
        for(i=0; i<n; i++){
            q.x[i][i]=1.0;
        }
        do{     //la condiciÃ³n while se hace al final comparando el
contador con el mÃ¡ximo de interaciones.
            k=0; l=1;
            maximo=Math.abs(a.x[k][1]);
            for(i=0; i<n-1; i++){
                for(j=i+1; j<n; j++){
                    if(Math.abs(a.x[i][j])>maximo){
                        k=i;        l=j;
                        maximo=Math.abs(a.x[i][j]);
```

```
                    }
                }
            }
            sumsq=0.0;
            for(i=0; i<n; i++){
                sumsq+=a.x[i][i]*a.x[i][i];
            }
            tolerancia=0.0001*Math.sqrt(sumsq)/n;
            if(maximo<tolerancia) break;
                                //calcula la matriz ortogonal de p
                                    //inicialmente es la matriz unidad
            for(i=0; i<n; i++){
                for(j=0; j<n; j++){
                    p.x[i][j]=0.0;
                }
            }
            for(i=0; i<n; i++){
                p.x[i][i]=1.0;
            }
            y=a.x[k][k]-a.x[l][l];
            if(Math.abs(y)<CERO){
                c=s=Math.sin(Math.PI/4); //el coseno y el seno es
igual para 45Âº
            }else{
                x=2*a.x[k][l];
                z=Math.sqrt(x*x+y*y);
                c=Math.sqrt((z+y)/(2*z));
                s=signo(x/y)*Math.sqrt((z-y)/(2*z));
            }
            p.x[k][k]=c;
            p.x[l][l]=c;
            p.x[k][l]=s;
            p.x[l][k]=-s;
            a=this.producto(p, this.producto(a, this.traspuesta(p)));
            q=this.producto(q, this.traspuesta(p));
            contador++; // contador=contador +1;
        }while(contador<maxIter);

        if(contador==maxIter){
                throw new ValoresExcepcion("Con este nÃºmero de
iteracciones no se consigue obtrener los valores propios");
        }
//valores propios
        //double[] valores=new double[n];
        for(i=0; i<n; i++){
        valores[i]=(double)Math.round(a.x[i][i]*1000)/1000;
        }
//vectores propios
        return q;
    }
////////////////////////////////////////////////////////////////////
//////
 int signo(double x){
       int retorno;
        if(x>0) retorno=1; else retorno=-1; return retorno;
       // return (x>0 ? 1 : -1);
    }

 //*****************************************************************
  String salidaMatriz(){
```

```
        String texto="\n";              // tiene interes para imprimir
matrices de enteros
        for(int i=0; i<x.length; i++){
            for(int j=0; j<x[i].length; j++)
            {texto+= "  "+x[i][j];}
             texto+="\n";
                           } texto+="\n";
           return texto;
                         }
    /*************************************************************/
  public   String _salidaMatrizInt(){  //igual que la anterior pero
solo saca valores enteros
        String texto="\n";              // tiene interes para imprimir
matrices de enteros
        for(int i=0; i<x.length; i++){
            for(int j=0; j<x[i].length; j++)
            {texto+= "  "+(int)Math.round(10*x[i][j]/10);}
             texto+="\n";
                           } texto+="\n";
           return texto;
                         }
    ////////////////////////////////////////////////////////////

  public   String salidaMatrizInt(){  //igual que la anterior pero
solo saca valores enteros
        String texto="\n";              // tiene interes para imprimir
matrices de enteros
        for(int i=0; i<this.getNumFilas(); i++){
            for(int j=0; j<this.getNumColumnas(); j++)
            {texto+= "  "+(int)Math.round(10*this.getElemento(i,
j)/10);}
             texto+="\n";
                           } texto+="\n";
           return texto;
                         }




//*****************************************************************
 String salidaVector(double[] vp){
         String texto="( ";
         for(int i=0; i<vp.length; i++){texto+="\t
"+(double)Math.round(1000*vp[i])/1000; }
         return texto+" )";
}
}

//*****************************************************************

class ValoresExcepcion extends Exception {

  public ValoresExcepcion() {
         super();
  }
  public ValoresExcepcion(String s) {
         super(s);
  }
}
```

Clase NavegaNudos

```
/*
()Julio MuÃ±iz Padilla
para una tesis doctoral del ingeniero Ernesto MuÃ±iz MartÃn
 */
package nodos;
import java.awt.Color;
import javax.swing.JFrame;
import java.awt.event.ActionEvent;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JTextField;
import static nodos.Paneli.cTipoVinculo;
import java.awt.event.*;
import java.io.Serializable;
import java.awt.GridLayout;


/**
 *
 * @author Julio MuÃƒÂ±iz y Ernesto MuÃ±iz
 */
public class NavegaNudos extends JFrame implements Serializable {

JPanel pExterno1;
JPanel  pExterno2;
JPanel   pExterno3  ;
JPanel pCarga=new JPanel(); // aparece cuando el punto externo no es
un apoyo
  JPanel pNumEnlaces ;
 JButton btnAceptarNuevoExterno=new  JButton("Aceptar");
JButton   btnMasEnlace=new  JButton("Confirmar enlace");
 JButton   btnEnlazarNudos=new  JButton("Entrar vinculos");
 JButton   btnCoorApo=new JButton("Apo(X,Y,Z)");
 JLabel lNumExterno=new JLabel("N="); JTextField   tNumExterno=new
JTextField("000",4);
 JLabel lExternoX= new JLabel("Origen (X");JTextField tExternoX=new
JTextField("100",4);
 JLabel lExternoY= new JLabel(",Y");JTextField       tExternoY=new
JTextField("100",4);
 JLabel lExternoZ= new JLabel(",Z");  JTextField     tExternoZ=new
JTextField("100",4);
 JLabel lparentesis=new JLabel(")");
JRadioButton chExtApoyo = new JRadioButton("Apoyo",false);
JRadioButton chExtNuevo= new JRadioButton("Nudo Nuevo", true);
 JLabel lExCargaX= new JLabel("Fx=");   JTextField  tExCargaX=new
JTextField("0",6);
 JLabel lExCargaY= new JLabel("Fy=");     JTextField tExCargaY=new
JTextField("0",6);
 JLabel lExCargaZ= new JLabel("Fz=");   JTextField  tExCargaZ=new
JTextField("0",6);
// JLabel   lNumeroVinculoExterno= new JLabel("");
JLabel lNudoEnlace=new JLabel("Punto enlazado 0");JTextField
tNudoEnlace=new JTextField("04",4);
 JButton  btnExtCrear=new JButton("CONFIRMAR NUDO EXTERNO");
```

```
JLabel lNumEnlaces=new JLabel("Numero enlaces"); JTextField
tNumEnlaces=new JTextField("001",10);
JButton  btnNumEnlaces=new JButton("Aceptar nu,.Enalaces");
 JTextField[]  tVinculo=new JTextField[Nodos.RESERVAVINCULO];
   JRadioButton[] chExterno=new JRadioButton[Nodos.RESERVAVINCULO];
   JRadioButton[] chSuperficie= new
JRadioButton[Nodos.RESERVAVINCULO];
 ////////////////////////////final de las
declaraciones//////////////////////////////////////////
 //////Clase constructora////////////////////////////////////

 NavegaNudos( int Ancho, int Alto,boolean bEnd){
  super();
  this.setTitle("Nudos Externos.");
 this.setSize(Ancho,Alto);
  if(bEnd) {
  this.addWindowListener(new WindowAdapter() {
  public void windowClosing(WindowEvent e){System.exit(0);}}
             );  }

 }

 void ponNavegaNudos(){
   JPanel panelExterno=new JPanel();
   pExterno1=new JPanel();
   pExterno2=new JPanel();
   pExterno3=new JPanel();
   pExterno1.setBackground(Color.CYAN);

pCarga.add(lExCargaX);pCarga.add(tExCargaX);pCarga.add(lExCargaY);pCar
ga.add(tExCargaY);pCarga.add(lExCargaZ);pCarga.add(tExCargaZ);

lNudoEnlace=new JLabel("Punto enlazado 0");
lNudoEnlace.setForeground(Color.BLUE); tNudoEnlace=new
JTextField("00",4); tNudoEnlace.setForeground(Color.BLUE);
  pExterno1.add(lNumExterno);
  lNumExterno.setText("Numeros
Externos="+String.valueOf(Nodos.lienzo.superficie.nudo.length));//Orde
n="+ String.valueOf(Nodos.lienzo.nOrdenExt+1));// pExterno1.add(
tNumExterno);
  pExterno1.add(chExtApoyo);
  pExterno1.add(lExternoX);  pExterno1.add( tExternoX);
  pExterno1.add(lExternoY ); pExterno1.add(tExternoY );
  pExterno1.add(lExternoZ);  pExterno1.add(tExternoZ );
pExterno1.add(lparentesis);
  pExterno1.add(pCarga);
  tExCargaX.setEnabled(true); tExCargaY.setEnabled(true);
tExCargaZ.setEnabled(true); //en principio los externos no tienen
carga sise consideran apoyos.


pExterno2.add(lNumEnlaces);pExterno2.add(tNumEnlaces);pExterno2.add(bt
nNumEnlaces);
  panelExterno.add(pExterno1);  pExterno1.setVisible(true);
  panelExterno.add(pExterno2);pExterno2.setVisible(false);
  panelExterno.add(pExterno3);pExterno3.setVisible(false);
  pExterno1.add(btnAceptarNuevoExterno);

if(Nodos.lienzo.superficie.numApoyos==0){JOptionPane.showMessageDialog
(  null, "Para evitar errores, marque ahor los apoyos sobre la
superficie"); }
```

```
 /*************************************************************/
  ////////////1Ã‚Âº boton de
aceptar//////////////////////////////////////////
 btnAceptarNuevoExterno.addActionListener(new
java.awt.event.ActionListener() {
  @Override
     public void actionPerformed(ActionEvent e) {


      String etiq="      ";
      String apx="X";
  if(Nodos.lienzo.superficie.nuExterno !=null)
Nodos.lienzo.superficie.nuExterno=null;
     Nodos.lienzo.superficie.nuExterno=new Nudo();
     Nodos.lienzo.superficie.nuExterno.coordenada=new Punto();
     Nodos.lienzo.superficie.nuExterno.carga=new JuVector(0,0,0);

Nodos.lienzo.superficie.nuExterno.setOrden(Nodos.lienzo.superficie.nud
o.length);
     Nodos.lienzo.superficie.nuExterno.tipoNudo='x';
     Nodos.lienzo.superficie.nuExterno.externo=true;

Nodos.lienzo.superficie.nuExterno.setX(Double.valueOf(tExternoX.getTex
t()).doubleValue());

Nodos.lienzo.superficie.nuExterno.setY(Double.valueOf(tExternoY.getTex
t()).doubleValue());

Nodos.lienzo.superficie.nuExterno.setZ(Double.valueOf(tExternoZ.getTex
t()).doubleValue());
     Nodos.lienzo.superficie.nuExterno.setExterno(true);
if(chExtApoyo.isSelected()){
            Nodos.lienzo.superficie.nuExterno.setApoyo(true);
            apx="A";Nodos.lienzo.superficie.setNumApoyos(
Nodos.lienzo.superficie.getNumApoyos()+1);
            Nodos.lienzo.superficie.nuExterno.carga.setComponentes(0,
0, 0);   }
else{
Nodos.lienzo.superficie.nuExterno.setApoyo(false);apx="X";//
            tExCargaX.setEnabled(true); tExCargaY.setEnabled(true);
tExCargaZ.setEnabled(true);pExterno3.setEnabled(true);

Nodos.lienzo.superficie.nuExterno.carga.setComponentes(Double.valueOf(
tExCargaX.getText()).doubleValue(),
Double.valueOf(tExCargaY.getText()).doubleValue(),
Double.valueOf(tExCargaZ.getText()).doubleValue());
    }
String sOrden =String.format("%04d",
Nodos.lienzo.superficie.nuExterno.getOrden());
etiq="N"+apx+sOrden+"X";
   Nodos.lienzo.superficie.nuExterno.setEtiqueta(etiq);
   pExterno2.setVisible(true);
  }});

 ////////////////////////////////////incluye los paneles de los
vinculos

btnNumEnlaces. addActionListener(new java.awt.event.ActionListener() {
  @Override
```

```
   public void actionPerformed(ActionEvent e) {
   pExterno2.setVisible(true);pExterno1.setEnabled(false);
   btnAceptarNuevoExterno.setVisible(false);
   int numVinculosNuExt=Integer.valueOf(tNumEnlaces.getText().trim());
   pExterno3.setLayout(new GridLayout(numVinculosNuExt,1,10,10));
   JPanel[] pVinculo=new JPanel[numVinculosNuExt];
   JLabel[] lVinculo= new JLabel[numVinculosNuExt];
   Nodos.lienzo.superficie.nuExterno.vinculo=new
Nudo[numVinculosNuExt];
   for(int n=0; n<numVinculosNuExt;n++){
   pVinculo[n]=new JPanel();
   lVinculo[n]=new JLabel("Orden del"+ String.valueOf(n+1)+"NÂº nudo a
enlazar=");
   tVinculo[n]=new JTextField("000");
   pVinculo[n].add(lVinculo[n]); pVinculo[n].add(tVinculo[n]);
  // chExterno[n]=new JRadioButton("Nudo Externo", false);
   chSuperficie[n]=new JRadioButton("Nudo Superficie", true);
  // pVinculo[n].add(chExterno[n]);
   pVinculo[n].add(chSuperficie[n]);
   pExterno3.add(pVinculo[n]);
   }

   pExterno3.setVisible(true);
   pExterno3.add(btnEnlazarNudos); btnEnlazarNudos.setVisible(true);
    }});

  ///////////2Ã‚Âº ////////////////////////////////////
   btnEnlazarNudos.addActionListener(new
java.awt.event.ActionListener() {
  @Override
     public void actionPerformed(ActionEvent e) {
     int
numVinculosNuExt=Integer.valueOf(tNumEnlaces.getText().trim());

Nodos.lienzo.superficie.setNumBarrasEX(Nodos.lienzo.superficie.getNumB
arrasEx()+numVinculosNuExt);

       for(int n=0; n<numVinculosNuExt;n++){
         if ( chSuperficie[n].isSelected()) cTipoVinculo='s'; else
cTipoVinculo='x';
         enlazarExterno(n,
Integer.valueOf(tVinculo[n].getText().trim()));
           }
       pExterno1.setVisible(true);
       pExterno2.setVisible(false);
       pExterno3.setVisible(false);
       setVisible(false);
       Paneli.btnSumExternos.doClick();
         }});

panelExterno.add(pExterno1);
panelExterno.add(pExterno2);
panelExterno.add(pExterno3);
this.add(panelExterno);
}

////////////////////////////////////////////////////////////////////
////////////
void  enlazarExterno(int vi, int en){
 Nudo temporal=new Nudo();
```

```
//se indica cuantos enlazas tiene el Nudo Externo origen que serÃƒÂ¡
igual al enlace activo mÃƒÂ¡s uno
 Nodos.lienzo.superficie.nuExterno.setNumBarrasE(vi+1);
//se busca el enlace del nudo de la superficie.
 int k=Nodos.lienzo.superficie.nudo[en].getOrden();
 //se reserva memoria para un puntero tipo nudo
 Nodos.lienzo.superficie.nuExterno.vinculo[vi]=new Nudo();
 //se vincula el nuevo vinculo del externo con el de la superficie.
 //se asigna el nudo de la superficie al vinculo del nudo externo

Nodos.lienzo.superficie.nuExterno.vinculo[vi]=(Nudo)Nodos.lienzo.super
ficie.nudo[k];
 //ahora se debe ir al nudo destino para vincular el origen
 //volcamos los datos anteriores en un Nudo temporal que le
aÃƒÂ±adimos un nuevo vinculo
 int
nVinculosTemporal=Nodos.lienzo.superficie.nudo[k].vinculo.length+1;
  temporal.vinculo=new Nudo[nVinculosTemporal];
 for(int i=0;i<nVinculosTemporal-1;i++){
 temporal.vinculo[i]=new Nudo();
temporal.vinculo[i]=Nodos.lienzo.superficie.nudo[k].vinculo[i];
                                        }
  temporal.vinculo[nVinculosTemporal-1]=new Nudo();
 //temporal.vinculo[nVinculosTemporal-1]=  Nodos.nuExtTemp[nda];
  temporal.vinculo[nVinculosTemporal-1]=
Nodos.lienzo.superficie.nuExterno;
 Nodos.lienzo.superficie.nudo[k].vinculo=null;
 Nodos.lienzo.superficie.nudo[k].vinculo= new Nudo[nVinculosTemporal];
 for(int i=0;i<nVinculosTemporal;i++){
  Nodos.lienzo.superficie.nudo[k].vinculo[i]=new Nudo();
  Nodos.lienzo.superficie.nudo[k].vinculo[i]=temporal.vinculo[i];
 }
  String eti= new String();
  eti="   ";
  char va='O'; if(Nodos.lienzo.superficie.nudo[k].apoyo)va='A';else
va='O';
  char ex='I'; if(Nodos.lienzo.superficie.nudo[k].externo)ex='X';else
ex='I';
  int pos= Nodos.lienzo.superficie.nudo[k].getOrden();
String sOrden =String.format("%04d", pos);
eti="N"+String.valueOf(va)+sOrden+String.valueOf(ex)+String.valueOf(No
dos.lienzo.superficie.nudo[pos].getTipoNudo())+String.valueOf(Nodos.li
enzo.superficie.nudo[pos].getTipoEleFinito());
Nodos.lienzo.superficie.nudo[k].setEtiqueta(eti);
 //suma automÃƒÂ¡tica de la matriz de externos

  }


}
```

Clase Nodos

```
/* Julio MuÃ±iz Padilla y Ernesto MuÃ±iz MartÃn 2014
version 13 de marzo de 2014 11:30
*/

package nodos;



import java.awt.*;
import java.io.Serializable;
import javax.swing.JFrame;

class Nodos implements Serializable {

 static final int RESERVAVINCULO=10;
 static double dSeccion=4;  //secciÃ³n de las barras por defecto
 static Lienzo lienzo ;
 static Paneli pabo, pagr,paca, paen,pana,paco,nuevo,pabar,paar;
//deben ser extÃ¡tica para poderlas llamar desde Paneli.
 static Ventana ventNavegacion,ventGrafica,ventCalculo,ventEntrada,
ventBarras, ventArchivo,ventVibracion;
static JFrame ventEleFinito=null ;
 ////////////////////////////////////CONSTRUCTOR DE LA CLASE
APLICACION Nodos//////////////////////////
static  boolean boolExtSumado=false;
static boolean bAlerta=false;
//static boolean bMostrar=false;

//////////////////////////////////////////////////////
public Nodos(){

 lienzo = new Lienzo();
 paco = new Paneli();
 paen=new Paneli();
 paca=new Paneli(); paca.PonVerCalculo();
 pana=new Paneli();
 nuevo=new Paneli();
 paar=new Paneli(); //panel de archivos
 pabar=new Paneli();//panel de la ventana de navegaciÃ³n de barras
 ventArchivo = new Ventana("Tesis Nodos: Dialogo de
archivos",225,325,paar,false);

 ventNavegacion = new Ventana("Tesis Nodos: ajuste de
nudos",600,250,paco,false);ventNavegacion.setResizable(false);
  paar.setBackground(Color.YELLOW);
 ventEntrada = new Ventana("Tesis Nodos: control",250,400,paen,false);
ventEntrada.setResizable(false);
 paen.setBackground(Color.YELLOW);

 ventCalculo = new Ventana("Tesis
Nodos;CÃ¡lculos",1200,600,paca,false);
 ventBarras= new Ventana("Tesis Nodos: ajuste de
barras",750,210,pabar,false);
 ventBarras.setResizable(false);
```

```
  Nodos.ventEleFinito = new JFrame("Tesis Nodos: Elementos Finitos");
paen.setLayout(new BorderLayout());
 paen.add(
paen.PonMenu(),BorderLayout.NORTH);paen.PonMenu().setVisible(true);
// paen.PonVentanaInicio();
 paen.add(
paen.PonVentanaInicio(),BorderLayout.CENTER);paen.PonVentanaInicio().s
etVisible(false);
 paen.add(new javax.swing.JLabel("NODOS, 1.(c)2014 Ernesto
MuÃ±iz,Julio MuÃ±iz") ,BorderLayout.SOUTH);
 pana.ponNavegaNudos();
 pabar.PonPanelBarras();
 nuevo.add(pana.lEtiqueta);
 nuevo.add(pana.tEtiqueta); // nuevo.add(pana.lSelecNudo);
 nuevo.add(pana.lSelecNudo) ;
 nuevo.add(pana.tSelecNudo);
 nuevo.add(pana.chApoyo);
 nuevo.add(pana.ponCargas());
 nuevo.add(pana.chCargaMasa);
 nuevo.add(pana.lCoordenadas) ;

 nuevo.add(nuevo.ponCoordenadas());
 paca.setBackground(Color.CYAN);
 pana.setBackground(Color.GREEN);
 paco.setLayout(new BorderLayout());
 paco.add(pana,BorderLayout.NORTH);
 paco.add(nuevo,BorderLayout.CENTER);
 paco.add(pana.btnOkNavegaNudo,BorderLayout.SOUTH);
 paca.areaTexto.setColumns(180);
 paca.areaTexto.setRows(55);
 ventEntrada.setVisible(true);
}

public static void empezar(){

pabo=new Paneli();
if(Nodos.lienzo.superficie.nudo!=null)Nodos.lienzo.superficie.nudo=nul
l;
if(Nodos.lienzo.superficie.nuOrdenado!=null)Nodos.lienzo.superficie.nu
Ordenado=null;
if(Nodos.lienzo.superficie.nuSalida!=null)Nodos.lienzo.superficie.nuSa
lida=null;
//if(Nodos.lienzo.pixelApo!=null)Nodos.lienzo.pixelApo=null;
if(Nodos.lienzo.superficie.NudoO!=null)Nodos.lienzo.superficie.NudoO=n
ull;
if(Nodos.lienzo.superficie.apoyo!=null)Nodos.lienzo.superficie.apoyo=n
ull;
if(Nodos.lienzo.superficie.barras!=null)Nodos.lienzo.superficie.barras
=null;
if(Nodos.lienzo.superficie.esquina!=null)Nodos.lienzo.superficie.esqui
na=null;
pabo.setLayout(new BorderLayout());
pabo.add(pabo.PonBotonesGraficos(),BorderLayout.NORTH);
//pabo.add(pabo.PonExternos(),BorderLayout.SOUTH);
pagr=new Paneli(); pagr.PonPizarraGrafica();
if(ventGrafica!=null){ ventGrafica.dispose();}

ventGrafica = new Ventana("Tesis Nodos: Graficos",1200,800,true);
ventGrafica.setEnabled(true);
ventGrafica.setLayout(new BorderLayout());
ventGrafica.add(pabo,BorderLayout.NORTH); pabo.setVisible(true);
```

```
ventGrafica.add(pagr,BorderLayout.SOUTH);pagr.setVisible(true);;
 ventCalculo.setVisible(true);
 ventGrafica.setVisible(true);
 ventNavegacion.setVisible(true);

}


public static void main(String[] args){
Nodos nodos;

  System.out.print("Programa para el cÃ¡lculo de superficies,
catenarias, puentes y cubiertas. (c) Julio MuÃ±iz Padilla y Ernesto
MuÃ±iz Â´MartÃn 2014.Todos los derechos reservados\n\n");
  System.out.print("Compilacion 13/03/2014.  Todos los derechos
reservados\n\n");
  nodos = new Nodos();
}

public static Nudo[] ampliarArrayNudos(Nudo[] nd, int m){
Nudo[] nuevo= new Nudo[nd.length+m];
//vuelco los datos de nd sobre los primeros elementos de nuevo
for (int j=0;j<nd.length;j++){nuevo[j]=new Nudo();nuevo[j]=nd[j];}
// aÃ±ado los nuevos elementos y los creos, pero no los inicializo
for (int i=nd.length;i<nd.length+m;i++){nuevo[i]=new Nudo();}
nd=null;
nd=nuevo.clone();
return nd;
        }
 // funcion sobrecargada que genera un nuevo array de nudos pero con 1
elemento incorporados

public static Nudo[] ampliarArrayNudos(Nudo[] nd, Nudo otro){
 int s=nd.length+1;
Nudo[] nuevo= new Nudo[s];
//vuelco los datos de nd sobre los primeros elementos de nuevo
for (int j=0;j<s-1;j++){nuevo[j]=new Nudo();nuevo[j]=nd[j];}
// aÃ±ado el nuevo elemento y lo creo, pero no los inicializo
nuevo[s-1]=new Nudo();nuevo[s-1]=otro;
return nuevo;
        }
// funcion sobrecargada que amplia un array de nudos con otro array de
nudo
public static Nudo[] ampliarArrayNudos(Nudo[] nd, Nudo[] otro){
Nudo[] nuevo= new Nudo[nd.length+otro.length];
//vuelco los datos de nd sobre los primeros elementos de
nuevoNodos.lienzo.superficie
for (int j=0;j<nd.length;j++){nuevo[j]=new Nudo();nuevo[j]=nd[j];}
// aÃ±ado los nuevos elementos y los creos, pero no los inicializo
int c=0;
for (int i=nd.length;i<nd.length+otro.length;i++){
    nuevo[i]=new Nudo(); nuevo[i]=otro[c];c++;}
nd=null;
nd=nuevo.clone();
return nd;}
//////////////////////////////////////////////////////////
static void killerMatrices(){
// if(Nodos.lienzo.superficie.D!=null)Nodos.lienzo.superficie.D=null;
//La matriz Densidad debe mantener los valores. Si se le cambia las
dimensiones se restructura
```

```
if(Nodos.lienzo.superficie.FormaOut!=null)Nodos.lienzo.superficie.Form
aOut=null;
if(Nodos.lienzo.superficie.FuApo!=null)Nodos.lienzo.superficie.FuApo=n
ull;
if(Nodos.lienzo.superficie.FuNudo!=null)Nodos.lienzo.superficie.FuNudo
=null;
if(Nodos.lienzo.superficie.FuSiste!=null)Nodos.lienzo.superficie.FuSis
te=null;
if(Nodos.lienzo.superficie.G!=null)Nodos.lienzo.superficie.G=null;
if(Nodos.lienzo.superficie.ME!=null)Nodos.lienzo.superficie.ME=null;
if(Nodos.lienzo.superficie.MEC!=null)Nodos.lienzo.superficie.MEC=null;
if(Nodos.lienzo.superficie.MESiste!=null)Nodos.lienzo.superficie.MESis
te=null;
if(Nodos.lienzo.superficie.MESisteInv!=null)Nodos.lienzo.superficie.ME
SisteInv=null;
if(Nodos.lienzo.superficie.transG!=null)Nodos.lienzo.superficie.transG
=null;
if(Nodos.lienzo.superficie.MapXYZ!=null)Nodos.lienzo.superficie.MapXYZ
=null;
}

 }
```

Clase Nudo

```
package nodos;

//import java.awt.Point;

import java.io.Serializable;


/**
 *
 * @author julio MuÃ±iz Padilla y Ernesto MuÃ±iz MartÃn
 */
public class Nudo implements Serializable  {
    String etiqueta;
    Punto coordenada;
    private int orden;
    private int ordenEx;
    private int indice;
    private double masa; //masa inicial introducida por el usuario
    private double masaTotal; //masa inicial + masa generada por las
barras. Inicialmente masaTotal=masa
    public JuVector carga;
    boolean activo;
    boolean enfocado=false;
    char tipoEleFinito; // t,c,d trinagulo, cuadrado, coudadrado con
diagonal.
    char tipoNudo; // s,l,e generico, lado,  esquina;
    boolean apoyo;  //por efecto false Un apoyo no es movible. No debe
tener carga sino que responderÃ¡ con reacciones
    boolean externo; //por defecto false  //los puntos externos pueden
ser apoyos tipo esquina.
    int nBarrasS=0; // es la dimensiÃ³n del array vinculo[]
    int nBarrasE=0;
    int nBarrasT=0;
    Nudo vinculo[];

  void Nudo() {
      this.etiqueta = new String();
      this.coordenada = new Punto();  this.coordenada.X=0;
this.coordenada.Y=0; this.coordenada.Z=0;
      carga=new JuVector(0,0,0);  //carga.comp[0]=0; carga.comp[1]=0;
carga.comp[2]=0;
        }


   public void setOrden(int ord){ this.orden=ord;  }
   public void setOrdenEx(int ordEx){ this.ordenEx=ordEx; }
   public void setOrdenIndice(int inOrd){ this.indice   =inOrd; }
   public void setNudo(double coordx, double coordy, double coordz,
char tipoMa) {
        this.tipoEleFinito = tipoMa;
        this.etiqueta = new String();
        this.coordenada = new Punto();
        this.coordenada.X = coordx;
        this.coordenada.Y = coordy;
```

```
        this.coordenada.Z = coordz;

        switch (this.tipoEleFinito) {

            case 'c': {
                if (this.tipoNudo == 'g') {
                    this.nBarrasS = 4;
                }
                if (this.tipoNudo == 'b') {
                    this.nBarrasS = 3;
                }
                if (this.tipoNudo == 'e') {
                    this.nBarrasS = 2;
                }
            }
            case 'd': {
                if (this.tipoNudo == 'g') {
                    this.nBarrasS = 8;
                }
                if (this.tipoNudo == 'b') {
                    this.nBarrasS = 5;
                }
                if (this.tipoNudo == 'e') {
                    this.nBarrasS = 3;
                }
            }
        }

    }

////////////////////////////////////////////////////////////////////
////////////////////

  void Nudo(char tipoMa) {
        this.tipoNudo = ' ';
        this.etiqueta = new String();
        this.etiqueta = "                              ";
        this.coordenada = new Punto();
        this.tipoEleFinito = tipoMa;
    }


////////////////////////////////////////////////////////////////////
///////////////
boolean esExterno(){boolean salida; if(this.externo)salida=true;else
salida=false; return salida;}

 public void setEtiqueta(String eti) {
        this.etiqueta = eti;
    }
 public void setNumBarrasS(int ba) {
        this.nBarrasS = ba;
this.nBarrasT=this.nBarrasE+ this.nBarrasS;
 }
 public void setNumBarrasE(int ba){
this.nBarrasE=ba;
this.nBarrasT=this.nBarrasE+ this.nBarrasS;
 }
 public int getNumBarrasS() { return this.nBarrasS; }
 public int getNumBarrasE() { return this.nBarrasE; }
 public int getNumBarrasT() { return ( this.nBarrasS+this.nBarrasE); }
```

```
public void settipoEleFinito(char cMalla) {
        this.tipoEleFinito = cMalla;  }

    public void setTipoNudo(char cNudo) {
        this.tipoNudo = cNudo;
    }

    public void setX(double coordx) {
        this.coordenada.X = coordx;
    }

    public void setY(double coordy) {
        this.coordenada.Y = coordy;
    }

    public void setZ(double coordz) {
        this.coordenada.Z = coordz;
    }
    public void setApoyo( boolean apo){this.apoyo=apo; }
    public void setExterno( boolean ext){this.externo=ext; }

    public int getOrden(){return this.orden;}
    public int getOrdenEx(){return this.ordenEx;}

    public int getIndice(){return this.indice;}

    public double getX() {
        return this.coordenada.X;
    }

    public double getY() {
        return this.coordenada.Y;
    }

    public double getZ() {
        return this.coordenada.Z;
    }

////////////////////////////las cargas deben incorporarse ante de las
masas////////
  public void setMasa (double mas){ this.masa=mas;}


  public double getMasa(boolean isCarga) {
        double masaSalida=0.0;
        if(isCarga){
            if(this.carga.getCompoZ()<0) masaSalida
=this.masa+Math.abs((this.carga.getCompoZ()/9.8));
            else masaSalida=this.masa;
        }
        else masaSalida=this.masa;
        return masaSalida;
    }

  public void setMasaTotal (double masT){ this.masaTotal=masT;}
  public double getMasaTotal(){return this.masaTotal;}

 public String getEtiqueta() {
        return this.etiqueta;
    }
```

```
 public char getTipoEleFinito() {
        return this.tipoEleFinito;
    }
 public char getTipoNudo() {
        return this.tipoNudo;
    }
 public Punto getCoordenada() {
        return this.coordenada;
    } //devuelve un elemento de la clase punto


    public boolean getApoyo(){ return this.apoyo;}
    public boolean getExterno(){ return this.externo;}
    public String toPrint() {
        return "(" + coordenada.X + "," + coordenada.Y + "," +
coordenada.Z + ")";
    }
   public String toPrint2d() {
        return "\t (X,Y,Z)=(" +
String.valueOf(Math.round(100*coordenada.X)/100) + ","
+String.valueOf( Math.round(100*coordenada.Y)/100)+ "," +
String.valueOf( Math.round(100*coordenada.Z)/100)+
               ")  Carga=(" +
String.valueOf(Math.round(100*carga.getCompoX())/100) + ","
+String.valueOf( Math.round(100*carga.getCompoY())/100)+ "," +
String.valueOf( Math.round(100*carga.getCompoZ())/100)+")
Masa="+this.getMasa(false)+" MaCa="+this.getMasa(true);
    }
    protected void finaleze(){}
}
```

Clase Paneli

```
/*
 * To Julio MuÃ±iz Padilla
 *26122013  VersiÃ³n 17.00 horas.
 */
/*
 AjusteMasas() 262
 calculoDinamico() 1410
 calculoForma() 1492
 test() 1304

 Recordatorio:
 Double to String
 Double.toString(numero)
 String to Double.
 Double.valueOf(texto numerico).doubleValue();
 */
package nodos;

import javax.swing.ImageIcon;
import javax.swing.*;
import java.awt.*;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;
//import java.io.ObjectOutputStream;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JComboBox;
import java.awt.TextArea;
import java.awt.TextField;
import javax.swing.JRadioButton;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
//import static nodos.Nodos.pabo;
import static nodos.Nodos.ventCalculo;
//import static nodos.Nodos.ventNavegacion;
import static nodos.Paneli.argumento;

public class Paneli extends JPanel implements Serializable {   //
static Matriz tempo;

    static JFrame frameVibra = null;
    static JLabel lPeriodo;
    static int m = 0;
    static JRadioButton rbAnima;
    static boolean bAnima = false;
    static double es = 0;
    static double esX = 0;
    static double esY = 0;
    static double esZ = 0;
```

```
    static Thread hilo;
    static JButton btnSumExternos;
    static JMenuItem barrasMenuItem;
    JButton btnCalBarras;
    JButton btnReacciones;
    JButton btnImagenVibra;
    JButton btnEleFinitos;
    static JButton btnCalMatrices;
    static JButton btnCalDinamico;
    static JButton btnComprueba;
    NavegaNudos navegaNudos;
    static int nActivo = 0;
    static JLabel lNudoEnlace;
    static int numbar = 1; //indicei del vinculo y nudos externos
activos
    static int numEF = 0; //indicei del vinculo y nudos externos
activos
    public JButton btnExtCrear, btnAceptarNuevoExterno, btnEnlazarCon,
btnMasEnlace, btnCoorApo;
    static JPanel pCarga;
    static JRadioButton chExtApoyo;
    JRadioButton chExtNuevo = new JRadioButton("Nuevo externo", true);
    static JTextField tNumExterno;
    static JTextField tFactorMasa;
    static JTextField tExternoX, tExternoY, tExternoZ;
    static JTextField tExCargaX, tExCargaY, tExCargaZ;
    static JTextField tNudoEnlace;
    static JTextField tNumEF, tEFEtiqueta, tEFDensiMasa, tEFGrosor,
tEFSuperficie, tEFMasa;
    JButton btnEFInicio, btnEFMas, btnEFMenos, btnEFFinal,
btnEFAceptarCambios;
    // static  JPanel panelBotones=new JPanel();
    JPanel pSalida = new JPanel();
    static ButtonGroup rbGrupo = new ButtonGroup();
    static ButtonGroup rbAngulo = new ButtonGroup(); //eleccion del
angulo respecto al eje de giro
    static char cTipoEleFinito = 'c';
    static char cTipoVinculo = 's';
    static String argumento;
    static TextArea texControl = new TextArea(argumento, 10, 20, 1);
    static JPanel pExterno1, pExterno2, pExterno3;
    static JPanel pMuestraEF;
    FileDialog dialogo;
    String[] cbItemEF = {"Rectangulo", "Rec.Diagonado", "Triangulo"};
    JComboBox cbTipoEleFinito = new JComboBox(cbItemEF);

    String[] comboItems = {"Externo", "Superficie"};
    String[] sMaterial = {"Acero B500", "Acero S235", "Acero S275",
"Acero S355", "Acero Y1760", "Acero Y1830", "Aluminio", "Textil T502",
"Textil T1202", "HA 30", "HA 40", "HA 50", "DEFINIDO USUARIO"};

    JComboBox cbMaterial = new JComboBox(sMaterial);
    JComboBox cbMaterialInicio = new JComboBox(sMaterial);
    JComboBox cbMaterialEF = new JComboBox(sMaterial);
    JButton btnNuevoExterno = new JButton("+ Externo");
    JButton btnAjusteMasas;
    JComboBox cbPlanos = null;
    String str, strTipoNudo;
    boolean verVentanaNudos = false, verVentanaBarras = false,
verVentanaGrafica = false, verVentanaCalculo = false;
```

```
    ImageIcon arrancar = new
ImageIcon(getClass().getResource("imagenes/ejecutar.gif"));
    ImageIcon dinamico = new
ImageIcon(getClass().getResource("imagenes/dinamico.gif"));
    ImageIcon actualizar = new
ImageIcon(getClass().getResource("imagenes/actualizar.gif"));
    ImageIcon lupaMas = new
ImageIcon(getClass().getResource("imagenes/lupamas.png"));
    ImageIcon lupaMenos = new
ImageIcon(getClass().getResource("imagenes/lupamenos.png"));
    ImageIcon reaccion = new
ImageIcon(getClass().getResource("imagenes/reaccion.png"));
    JButton btnCalculo = new JButton("Inicia");
    JButton btnActualizar = new JButton("Actualiza");
    JButton btnCoorApoyo = new JButton("A(x,y,z)");
    JButton btnEscalaMas = new JButton();
    JButton btnEscalaMenos = new JButton();
    JButton btnExit = new JButton("Exit");
    JButton btnBorrar = new JButton("Limpiar");
    JButton btnEjes = new JButton("");
    JButton btnCoordenadas = new JButton("A(x,y,z)");
    JButton btnNudos = new JButton("");
    JButton btnNudosSalida = new JButton("");
    //JButton btnPeriEsquina = new JButton("PerÃ-Esquina");
    JButton btnBarra = new JButton("Barras");
    //JButton btnEleFinito = new JButton("EF");
    // JScrollBar scrollAngulo = new JScrollBar();
    JButton btnNumNudo = new JButton("NumNudo");
    static JButton btnEjecutar = new JButton("");
    JButton btnMatrizMasas = new JButton("Masas");
    String cadena = new String();
    JButton btnXmas = new JButton("   X +   ");
    JButton btnXmenos = new JButton("   - X   ");
    JButton btnYmas = new JButton(" Y + ");
    JButton btnYmenos = new JButton(" - Y ");
    JButton btnZmas = new JButton("   Z +   ");
    JButton btnZmenos = new JButton("   - Z   ");
    JButton btnBarraMas = new JButton("=>");
    JButton btnBarraMenos = new JButton("<=");
    JButton btnBarraInicio = new JButton("|<<");
    JButton btnBarraFinal = new JButton(">>|");
    JButton btnBarraAceptarCambio = new JButton("Confirmar Cambio");
    JLabel lIteraciones = new JLabel("Iteraciones");
    static TextField tIteraciones = new TextField("10000", 6);
    JLabel lLongEleFinito = new JLabel("E.Finito dX,dY");
    TextField tLongEleFinitoX = new TextField("45", 3);
    TextField tLongEleFinitoY = new TextField("45", 3);
    TextField tAnchoRectangulo = new TextField("200", 6);
    TextField tAltoRectangulo = new TextField("150", 6);
    JLabel lAnchoRectangulo = new JLabel("Ancho rectÃ¡ngulo");
    JLabel lAltoRectangulo = new JLabel("Alto rectÃ¡ngulo");
    JLabel lGrosorEF = new JLabel("Grosor EleFinito");
    TextField tGrosorEF = new TextField("2", 6);
    JLabel lDensidad = new JLabel("Den.General F N/m");
    JLabel lMaterial = new JLabel("Material");
    JLabel lDensiEF = new JLabel("Densidad EF kg/m3");
    JLabel lSeccion = new JLabel("SecciÃ³n barra cm2");
    JLabel lTipoEleFinito = new JLabel("Tipo Elemento Finito");
    TextField tDensidad = new TextField("1000", 6);
    TextField tBarraDensFuerzaF = new TextField("1000", 12);
```

```
    TextField tBarraDensFuerzaC = new TextField("1000", 12);
    JLabel lNumBarra = new JLabel("Barra num: ");
    TextField tNumBarra = new TextField("", 4);
    JLabel lNuInicio = new JLabel("Inicio");
    TextField tNuInicio = new TextField("", 4);
    JLabel lNuFin = new JLabel("Fin");
    TextField tNuFin = new TextField("", 4);
    JLabel lBarraEtiqueta = new JLabel("Etiqueta: ");
    TextField tBarraEtiqueta = new TextField("", 12);
    JLabel lBarraLongitud = new JLabel("Longitud: ");
    TextField tBarraLongitud = new TextField("", 12);
    JLabel lBarraFuerzaAxial = new JLabel("Fuerza Axial:");
    TextField tBarraFuerzaAxial = new TextField("", 12);
    JLabel lBarraDensidad = new JLabel("Densidad kg/m3:");
    TextField tBarraDensiMasa = new TextField("", 12);
    JLabel lBarraModuloElastico = new JLabel("Modulo Elastico");
    TextField tBarraModuloElastico = new TextField("", 12);
    JLabel lBarraLimiteElastico = new JLabel("LÃmite Elastico");
    TextField tBarraLimiteElastico = new TextField("", 12);
    JLabel lBarraSeguridad = new JLabel("Coeficiente Seguridad");
    TextField tBarraSeguridad = new TextField("", 12);
    JLabel lBarraSeccion = new JLabel("SecciÃ³n cm2");
    TextField tBarraSeccion = new TextField("12", 12);
    TextField tDensiEF = new TextField("0.00", 6);
    TextField tBarraSeccionEntrada = new TextField("", 12);
    TextField tBarraSeccionC = new TextField("", 12);
    TextField tBarraMasa = new TextField("", 12);
    TextField tBarraRatio = new TextField("", 12);
    TextField tBarraAcortamiento = new TextField("", 12);
    JLabel lSelecNudo = new JLabel("Nudo Seleccionado");
    static int num = 0, nTotalNudos;
    static String etiq = new String();
    TextField tSelecNudo = new TextField("0", 3);
    TextArea areaTexto = new TextArea(cadena, 10, 20, 1);
    JLabel lCoordenadas = new JLabel("( X    , Y ,   Z )");
    JButton btnMasAngulo = new JButton("+"), btnMenosAngulo = new
JButton("-");
    JLabel lAngulo = new JLabel("angulo");
    JLabel lEtiqueta = new JLabel("Etiqueta Nudo");
    TextField tEtiqueta = new TextField("10", 6);
    JButton btnOkNavegaNudo = new JButton("Confirma");
    JRadioButton chApoyo = new JRadioButton("Apoyo", false);
    JRadioButton chTraccion = new JRadioButton("Traccion", false);
    JRadioButton chSuperficie = new JRadioButton("Superficie", true);
    JRadioButton chExterno = new JRadioButton("Externo", false);
    static JRadioButton chCargaMasa = new JRadioButton("+Masa Carga",
false);
    JRadioButton chAlerta = new JRadioButton("alertas", false);
    static JRadioButton chMostrar = new JRadioButton("Ver Calculo",
false);
    JRadioButton chAgotamiento = new JRadioButton("Agota", false);
    static JRadioButton chGiroX = new JRadioButton("Giro X", false);
    static JRadioButton chGiroY = new JRadioButton("Giro Y", true);
    static JRadioButton chGiroZ = new JRadioButton("Giro Z", false);
    ////////////////////nuevas entradas ////////////////////// 07 de
julio de 2013
    JLabel lCargaX = new JLabel("Carga X");
    static TextField tCargaX = new TextField("0 ", 6);
    JLabel lCargaY = new JLabel("Carga Y");
    static TextField tCargaY = new TextField("0", 6);
    JLabel lCargaZ = new JLabel("Carga Z");
```

```
    static TextField tCargaZ = new TextField("0 ", 6);
    ////////////////////
    JLabel lCoorX = new JLabel("X=");
    static TextField tCoorX = new TextField("0 ", 6);
    JLabel lCoorY = new JLabel("Y=");
    static TextField tCoorY = new TextField("0", 6);
    JLabel lCoorZ = new JLabel("Z=");
    static TextField tCoorZ = new TextField("0 ", 6);
    ////////////////////
    static JLabel lNumeroIteraciones = new JLabel("N.I=  ");
    static JLabel lNumeroVinculoExterno = new JLabel();

    /**
     *
**************************************************************************
************************
     */
    public void calculoInicio() {
        String str = "";
        Nodos.lienzo.superficie = new Superficie();
        str = (String) cbTipoEleFinito.getSelectedItem();
        if (str == "Triangulo") {
            cTipoEleFinito = 't';
        }
        if (str == "Rectangulo") {
            cTipoEleFinito = 'c';
        }
        if (str == "Rec.Diagonado") {
            cTipoEleFinito = 'd';
        }
        Nodos.lienzo.superficie.grosorEF = 0.0001 *
Double.valueOf(tGrosorEF.getText()).doubleValue();
        Nodos.lienzo.superficie.densiEF =
Double.valueOf(tDensiEF.getText()).doubleValue();
        //
Nodos.lienzo.superficie.setGeoInicio(Double.valueOf(tAnchoRectangulo.g
etText()).doubleValue(),
Double.valueOf(tAltoRectangulo.getText()).doubleValue(),
Double.valueOf(tLongEleFinitoX.getText()).doubleValue(),
Double.valueOf(tLongEleFinitoY.getText()).doubleValue(),
cTipoEleFinito);
        Nodos.lienzo.superficie.setTipoEleFinito(cTipoEleFinito);
        Nodos.lienzo.boolNudos = true;
        Nodos.lienzo.superficie.densiFuerzaBarraForma =
Double.valueOf(tDensidad.getText()).doubleValue();
        Nodos.lienzo.superficie.seccionBarra = 0.0001 *
Double.valueOf(tBarraSeccion.getText()).doubleValue(); //se guarda
para usarlo cuando se creen las barras
//inicializamos el numero de barras que se asignarÃ¡n cuando estas se
creen en forma
        Nodos.lienzo.superficie.setNumBarras(0);
        str = (String) cbMaterialInicio.getSelectedItem();
        if (str == "Acero B500") {
            Nodos.lienzo.superficie.dMaterial[0] = 7850.0;
            Nodos.lienzo.superficie.dMaterial[1] = 500e6;
            Nodos.lienzo.superficie.dMaterial[2] = 19e10;
            Nodos.lienzo.superficie.dMaterial[3] = 2.5;
        }
        if (str == "Acero S235") {
            Nodos.lienzo.superficie.dMaterial[0] = 7850.0;
            Nodos.lienzo.superficie.dMaterial[1] = 235e6;
```

```
        Nodos.lienzo.superficie.dMaterial[2] = 21e10;
        Nodos.lienzo.superficie.dMaterial[3] = 2.5;
    };
    if (str == "Acero S275") {
        Nodos.lienzo.superficie.dMaterial[0] = 7850.0;
        Nodos.lienzo.superficie.dMaterial[1] = 2750e6;
        Nodos.lienzo.superficie.dMaterial[2] = 21e10;
        Nodos.lienzo.superficie.dMaterial[3] = 2.5;
    };
    if (str == "Acero S355") {
        Nodos.lienzo.superficie.dMaterial[0] = 7850.0;
        Nodos.lienzo.superficie.dMaterial[1] = 355e6;
        Nodos.lienzo.superficie.dMaterial[2] = 21e10;
        Nodos.lienzo.superficie.dMaterial[3] = 2.5;
    };
    if (str == "Acero Y1760") {
        Nodos.lienzo.superficie.dMaterial[0] = 7850.0;
        Nodos.lienzo.superficie.dMaterial[1] = 1760e6;
        Nodos.lienzo.superficie.dMaterial[2] = 21e10;
        Nodos.lienzo.superficie.dMaterial[3] = 2.5;
    };
    if (str == "Acero Y1830") {
        Nodos.lienzo.superficie.dMaterial[0] = 7850.0;
        Nodos.lienzo.superficie.dMaterial[1] = 1830e6;
        Nodos.lienzo.superficie.dMaterial[2] = 21e10;
        Nodos.lienzo.superficie.dMaterial[3] = 2.5;
    };
    if (str == "Aluminio") {
        Nodos.lienzo.superficie.dMaterial[0] = 2700.0;
        Nodos.lienzo.superficie.dMaterial[1] = 70e6;
        Nodos.lienzo.superficie.dMaterial[2] = 2.1e10;
        Nodos.lienzo.superficie.dMaterial[3] = 2.5;
    };
    if (str == "Textil T502") {
        Nodos.lienzo.superficie.dMaterial[0] = 1000.0;
        Nodos.lienzo.superficie.dMaterial[1] = 5.0e8;
        Nodos.lienzo.superficie.dMaterial[2] = 21e10;
        Nodos.lienzo.superficie.dMaterial[3] = 2.5;
    };
    if (str == "Textil T1202") {
        Nodos.lienzo.superficie.dMaterial[0] = 1000.0;
        Nodos.lienzo.superficie.dMaterial[1] = 5.0e8;
        Nodos.lienzo.superficie.dMaterial[2] = 21e10;
        Nodos.lienzo.superficie.dMaterial[3] = 2.5;
    };
    if (str == "HA 30") {
        Nodos.lienzo.superficie.dMaterial[0] = 2500.0;
        Nodos.lienzo.superficie.dMaterial[1] = 5.0e8;
        Nodos.lienzo.superficie.dMaterial[2] = 30e10;
        Nodos.lienzo.superficie.dMaterial[3] = 1.5;
    };
    if (str == "HA 40") {
        Nodos.lienzo.superficie.dMaterial[0] = 2500.0;
        Nodos.lienzo.superficie.dMaterial[1] = 5.0e8;
        Nodos.lienzo.superficie.dMaterial[2] = 40e10;
        Nodos.lienzo.superficie.dMaterial[3] = 1.5;
    };
    if (str == "HA 50") {
        Nodos.lienzo.superficie.dMaterial[0] = 2500.0;
        Nodos.lienzo.superficie.dMaterial[1] = 5.0e8;
        Nodos.lienzo.superficie.dMaterial[2] = 50e10;
```

```
            Nodos.lienzo.superficie.dMaterial[3] = 1.5;
        };
        if (str == "DEFINIDO USUARIO") {
            Nodos.lienzo.superficie.dMaterial[0] = 2500.0;
            Nodos.lienzo.superficie.dMaterial[1] = 5.0e8;
            Nodos.lienzo.superficie.dMaterial[2] = 21e10;
            Nodos.lienzo.superficie.dMaterial[3] = 2.5;
        };

Nodos.lienzo.superficie.setGeoInicio(Double.valueOf(tAnchoRectangulo.g
etText()).doubleValue(),
Double.valueOf(tAltoRectangulo.getText()).doubleValue(),
Double.valueOf(tLongEleFinitoX.getText()).doubleValue(),
Double.valueOf(tLongEleFinitoY.getText()).doubleValue(),
Nodos.lienzo.superficie.getTipoEleFinito());
    }

    ////////////////////////////////VENTANA DONDE SE VEN LOS
CACULOS/////////////////////////////////////77
    void PonVerCalculo() {
        //  Dimension dim =this.getSize();
        tFactorMasa = new
JTextField(Double.toString(Nodos.lienzo.superficie.FactorMasa), 6);
        JButton btnDesplazar = new JButton("Desplaza^");
        JButton btnVerNudos = new JButton("List nudos");
        JButton btnVerExternos = new JButton("List Externos");
        btnAjusteMasas = new JButton("Ajust.MASA");
        // JButton btnEliminarExterno=new JButton("Eliminar Externo");
        btnSumExternos = new JButton("Sumar Externos");
        JButton btnOrdNudos = new JButton("Ordena nudos");
        //miButton.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/miimagen.JPG")));
        // ImageIcon arrancar = new
ImageIcon(getClass().getResource("imagenes/ejecutar.gif"));
        btnCalMatrices = new JButton("Form");
        btnCalDinamico = new JButton("Din");
        btnComprueba = new JButton("Test");
        btnCalMatrices.setIcon(arrancar);
        btnComprueba.setIcon(arrancar);
        btnCalDinamico.setIcon(arrancar);
        JButton btnBorrarPantalla = new JButton("CLEAR");
        btnCalBarras = new JButton("Genera Barras");
        JButton btnVerBarras = new JButton("List Barras");
        JButton btnVerEF = new JButton("Ver EF");
        add(chMostrar);
        add(btnDesplazar);
        btnDesplazar.setBackground(Color.GREEN);
        add(btnBorrarPantalla);
        add(btnVerNudos);
        add(btnOrdNudos);
        add(btnVerExternos);//add(btnEliminarExterno);
        //  add(btnSumExternos);
        add(btnCalMatrices);

btnCalMatrices.setBackground(Color.PINK);//btnCalMatrices.setIcon(new
ImageIcon("./imagenes/ejecutar.ico"));

        // add(btnCalBarras); setIcon( new ImageIcon("ruta de la
imagen en tu pc") )
        add(btnVerBarras);
        add(btnVerEF);
```

```
        add(new JLabel("Factor Masa"));
        add(tFactorMasa);
        add(btnMatrizMasas);
        add(btnAjusteMasas);
        btnComprueba.setBackground(Color.GREEN);
        btnAjusteMasas.setEnabled(false);
        add(lNumeroIteraciones);
        add(lIteraciones);
        add(tIteraciones);
        add(btnCalDinamico);
        btnCalDinamico.setBackground(Color.ORANGE);
        btnCalDinamico.setEnabled(false);
        add(btnComprueba);
        btnComprueba.setBackground(Color.BLUE);
        btnComprueba.setEnabled(false);

        btnVerNudos.setVisible(true);
        add(areaTexto);
        areaTexto.setBackground(Color.WHITE);
        areaTexto.setVisible(true);
        ///////////////////////////////NUEVA
26712/2013////////////////////////////

//Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿
Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿
Â¿Â¿Â¿Â¿Â¿Â¿
        btnAjusteMasas.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                int numero =
Nodos.lienzo.superficie.nuOrdenado.length;
                //se crean las matrices
                double[] masaTemp = new double[numero];
                boolean salir = false;
                Nodos.lienzo.superficie.FuSiste = new Matriz(numero,
3);
                Nodos.lienzo.superficie.MESiste = new
Matriz(Nodos.lienzo.superficie.ME.numFilas,
Nodos.lienzo.superficie.ME.numColumnas);
                Nodos.lienzo.superficie.MESisteInv = new
Matriz(Nodos.lienzo.superficie.MESiste.numFilas,
Nodos.lienzo.superficie.MESiste.numColumnas);
                Nodos.lienzo.superficie.FormaOut = new
Matriz(Nodos.lienzo.superficie.MESiste.numFilas, 3);
                Nodos.lienzo.superficie.Masa = new Matriz(numero,
numero);
                //se inicia la iteraciÃ³n
                int q = 0;
                do {
                    for (int n = 0; n < numero; n++) {

Nodos.lienzo.superficie.nuOrdenado[n].setMasaTotal(Nodos.lienzo.superf
icie.Masa.getElemento(n, n));
                        masaTemp[n] =
Nodos.lienzo.superficie.nuOrdenado[n].getMasaTotal();
                    }

                    for (int i = 0; i <
Nodos.lienzo.superficie.getNumApoyos(); i++) {
                        Nodos.lienzo.superficie.FuSiste.x[i][0] =
Nodos.lienzo.superficie.MapXYZ.x[i][0];
```

```
                        Nodos.lienzo.superficie.FuSiste.x[i][1] =
Nodos.lienzo.superficie.MapXYZ.x[i][1];
                        Nodos.lienzo.superficie.FuSiste.x[i][2] =
Nodos.lienzo.superficie.MapXYZ.x[i][2];
                    }
                    for (int i =
Nodos.lienzo.superficie.getNumApoyos(); i < numero; i++) {

//Nodos.lienzo.superficie.FuSiste.x[i][0]=Nodos.lienzo.superficie.nuOr
denado[i].carga.getCompoX();
                        Nodos.lienzo.superficie.FuSiste.setElemento(i,
0, Nodos.lienzo.superficie.nuOrdenado[i].carga.getCompoX());

//Nodos.lienzo.superficie.FuSiste.x[i][1]=Nodos.lienzo.superficie.nuOr
denado[i].carga.getCompoY();
                        Nodos.lienzo.superficie.FuSiste.setElemento(i,
1, Nodos.lienzo.superficie.nuOrdenado[i].carga.getCompoY());

//Nodos.lienzo.superficie.FuSiste.x[i][2]=Nodos.lienzo.superficie.nuOr
denado[i].carga.getCompoZ();
                        Nodos.lienzo.superficie.FuSiste.setElemento(i,
2, (-9.8 * Nodos.lienzo.superficie.nuOrdenado[i].getMasaTotal()));
                    }
                    Nodos.lienzo.superficie.FuSiste =
Nodos.lienzo.superficie.FuSiste.resta(Nodos.lienzo.superficie.FuSiste,
Nodos.lienzo.superficie.FuApo);

                    // Creo la matriz MESiste, y la clono con ME;
                    Nodos.lienzo.superficie.MESiste =
Nodos.lienzo.superficie.ME.clone();
                    for (int i = 0; i <
Nodos.lienzo.superficie.getNumApoyos(); i++) {
                        for (int j = 0; j <
Nodos.lienzo.superficie.getNumApoyos(); j++) {
                            if (i == j) {

Nodos.lienzo.superficie.MESiste.x[i][j] = 1;
                            } else {

Nodos.lienzo.superficie.MESiste.x[i][j] = 0;
                            }
                        }
                    }
                    for (int k =
Nodos.lienzo.superficie.getNumApoyos(); k <
Nodos.lienzo.superficie.MESiste.numFilas; k++) {
                        for (int l = 0; l <
Nodos.lienzo.superficie.getNumApoyos(); l++) {
                            Nodos.lienzo.superficie.MESiste.x[k][l] =
0;
                        }
                    }

// Calcular la inversa de la matriz MESiste
                    Nodos.lienzo.superficie.MESisteInv =
Nodos.lienzo.superficie.MESisteInv.inversa(Nodos.lienzo.superficie.MES
iste);

                    Nodos.lienzo.superficie.FormaOut =
Nodos.lienzo.superficie.FormaOut.producto(Nodos.lienzo.superficie.MESi
steInv, Nodos.lienzo.superficie.FuSiste);
```

```
                    for (int k = 0; k <
Nodos.lienzo.superficie.nudo.length; k++) {
                        Nodos.lienzo.superficie.nuSalida[k] =
Nodos.lienzo.superficie.nuOrdenado[k];
                    }
                    for (int k =
Nodos.lienzo.superficie.getNumApoyos(); k <
Nodos.lienzo.superficie.nuOrdenado.length; k++) {

Nodos.lienzo.superficie.nuSalida[k].setX(Nodos.lienzo.superficie.Forma
Out.x[k][0]);

Nodos.lienzo.superficie.nuSalida[k].setY(Nodos.lienzo.superficie.Forma
Out.x[k][1]);

Nodos.lienzo.superficie.nuSalida[k].setZ(Nodos.lienzo.superficie.Forma
Out.x[k][2]);
                    }
 ///////////////////

                    //Creacion de la matriz Masa y asignamos a la
diagonal el valor 1. el resto 0
                    for (int j = 0; j <
Nodos.lienzo.superficie.Masa.numFilas; j++) {
                        for (int k = 0; k <
Nodos.lienzo.superficie.Masa.numColumnas; k++) {
                            if (j != k) {
                                Nodos.lienzo.superficie.Masa.x[j][k] =
0;
                            } else {
                                Nodos.lienzo.superficie.Masa.x[j][k] =
1;
                            }
                        }
                    }
                    boolean activar;
                    if (chCargaMasa.isSelected()) {
                        activar = true;
                    } else {
                        activar = false;
                    }
                    Nodos.lienzo.superficie.isCarga = activar;
                    Nodos.lienzo.superficie.Masa =
Barra.asignarMasaNudos(Nodos.lienzo.superficie.barras, activar);
                    for (int i = 0; i < numero;) {
                        if (Math.abs(masaTemp[i] -
Nodos.lienzo.superficie.Masa.getElemento(i, i)) >= 1) {
                            salir = true;
                            i = numero;
                        } else {
                            i++;
                            salir = false;
                        }
                    }
                    lNumeroIteraciones.setText("N.I.=" +
String.valueOf(q++));
                } while (salir);
// finaliza la iteracciÃ³n
                ////////////////////////////
```

```
                //llamamos la funciÃ³n que genera el array de 
Rracciones
                if (chMostrar.isSelected()) {
                    areaTexto.append("\n(A1) Matriz de Fuerzas del 
Sistema, FUSiste ");
                }
                if (chMostrar.isSelected()) {

areaTexto.append(Nodos.lienzo.superficie.FuSiste.salidaMatriz());
                }
                if (chMostrar.isSelected()) {
                    areaTexto.append("\nMatriz de ResoluciÃ³n del 
sistema, MESiste ");
                }
                if (chMostrar.isSelected()) {

areaTexto.append(Nodos.lienzo.superficie.MESiste.salidaMatriz());
                }
                if (chMostrar.isSelected()) {
                    areaTexto.append("\nMatriz inversa de la funciÃ³n 
MESiste, MESisteInv ");
                }
                if (chMostrar.isSelected()) {

areaTexto.append(Nodos.lienzo.superficie.MESisteInv.salidaMatriz());
                }
                if (chMostrar.isSelected()) {
                    areaTexto.append("\nMatriz de Resultados de la 
forma de la superficie ");
                }
                if (chMostrar.isSelected()) {

areaTexto.append(Nodos.lienzo.superficie.FormaOut.salidaMatriz());
                }
                areaTexto.append("\nMatriz de Masas asignada a los 
nudos ");

areaTexto.append(Nodos.lienzo.superficie.Masa.salidaMatriz());
                Nodos.lienzo.superficie.reacciones();
                Nodos.lienzo.repaint();

            }
        });
        //////////////////////////////////fin NUEVA 
/////////////////////////

//Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿
Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿Â¿
Â¿Â¿
        btnDesplazar.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                areaTexto.append("  \n\n\n\n");
            }
        });

////////////////////////////////////////////////////////////////ORDEN
AR NUDOS//////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////
/////////////
        btnCalBarras.addActionListener(new ActionListener() {
```

```
            public void actionPerformed(ActionEvent e) {
                int numNuTotal = Nodos.lienzo.superficie.nudo.length;
                Barra[] copia = null; //declaro un array de barras que
luego uso.
                double numBa = 0;
                for (int n = 0; n < numNuTotal; n++) {
                    numBa = numBa +
Nodos.lienzo.superficie.nudo[n].vinculo.length;
                }
                int numBarras = (int) (numBa / 2);
                Nodos.lienzo.superficie.setNumBarras(numBarras);

                //Establecido el numero de barras hayan o no hayan
nudos externos debemos crear el Array de barras si no existe
                if (Nodos.lienzo.superficie.barras == null) {
                    Nodos.lienzo.superficie.barras = new
Barra[Nodos.lienzo.superficie.getNumBarras()];
                    for (int n = 0; n <
Nodos.lienzo.superficie.getNumBarras(); n++) {
                        Nodos.lienzo.superficie.barras[n] = new
Barra();

Nodos.lienzo.superficie.barras[n].setEtiqueta("NA");

Nodos.lienzo.superficie.barras[n].setMaterial(Nodos.lienzo.superficie.
dMaterial);

Nodos.lienzo.superficie.barras[n].densiFuerzaForma =
Nodos.lienzo.superficie.densiFuerzaBarraForma;
                    }

                } else {/* copia= new
Barra[Nodos.lienzo.superficie.barras.length];
                     for(int
k=0;k<Nodos.lienzo.superficie.barras.length;k++)
                     {copia[k]=new Barra();
copia[k]=Nodos.lienzo.superficie.barras[k]; }
                     */

                    copia = (Barra[])
Nodos.lienzo.superficie.barras.clone();
                }
                //creado el Array de barras debemos asignarle sus
valores a la Matriz de Densidad
                Nudo[] cNudo = new
Nudo[Nodos.lienzo.superficie.nudo.length];
                cNudo = (Nudo[]) Nodos.lienzo.superficie.nudo.clone();
                Nodos.lienzo.superficie.barras =
Barra.generaArrayBarras(cNudo);
                cNudo = null;
                ////////////////////////////////////SE copoian los
datos anteriores
                if (copia != null) {
                    if (Nodos.bAlerta) {
                        JOptionPane.showMessageDialog(null,
String.valueOf(Nodos.lienzo.superficie.barras.length + "," +
copia.length));
                    }
                    for (int h = 0; h <
Nodos.lienzo.superficie.barras.length; h++) {
                        for (int i = 0; i < copia.length; i++) {
```

```
                            if
(Nodos.lienzo.superficie.barras[h].etiqueta.compareTo(copia[i].etiquet
a) == 0) {
                                //JOptionPane.showMessageDialog(
null,Nodos.lienzo.superficie.barras[h].etiqueta+","+copia[i].etiqueta
);

Nodos.lienzo.superficie.barras[h].coeficienteSeguridad =
Nodos.lienzo.superficie.barras[i].coeficienteSeguridad;

Nodos.lienzo.superficie.barras[h].setSeccion(copia[i].getSeccion());

Nodos.lienzo.superficie.barras[h].modElas = copia[i].modElas;

Nodos.lienzo.superficie.barras[h].limElas = copia[i].limElas;

Nodos.lienzo.superficie.barras[h].densiMasa = copia[i].densiMasa;

Nodos.lienzo.superficie.barras[h].densiFuerzaForma =
Nodos.lienzo.superficie.barras[i].densiFuerzaForma;
                                break;
                            }
                        }
                    }
                }
                copia = null;
                for (int i = 0; i <
Nodos.lienzo.superficie.barras.length; i++) {
                    Nodos.lienzo.superficie.barras[i].actualizar();
                }
                verBarras(Nodos.lienzo.superficie.barras);
            }
        });
////////////////////////////////////////////////////////////////////
//////////////////////////////////////

        btnVerBarras.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                verBarras(Nodos.lienzo.superficie.barras);
            }
        });
        ////////////////////////////////////////////////

        btnVerEF.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                verEF(Nodos.lienzo.superficie.eleFinito);
            }
        });


////////////////////////////////////////////////////////////////////
//////////////////////////////////
        btnSumExternos.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                int a = Nodos.lienzo.superficie.nudo.length;
                int s = a + 1;
                Nudo[] temporal = new Nudo[s];
                for (int i = 0; i < s; i++) {
                    temporal[i] = new Nudo();
                    temporal[i] = new Nudo();
                    temporal[i].carga = new JuVector(0, 0, 0);
```

```
                }
                temporal =
Nodos.ampliarArrayNudos(Nodos.lienzo.superficie.nudo,
Nodos.lienzo.superficie.nuExterno);

//Nodos.lienzo.superficie.nudo=Nodos.ampliarArrayNudos(Nodos.lienzo.su
perficie.nudo, temporal);
                Nodos.lienzo.superficie.nudo = null;
                Nodos.lienzo.superficie.nudo = new
Nudo[temporal.length];

                Nodos.lienzo.superficie.nudo = (Nudo[])
temporal.clone();
                temporal = null;
                Nodos.lienzo.repaint();
                // en lÃnea 950
            }
        });

////////////////////////////////////ORDENAR**************************
******************/
        btnOrdNudos.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if (Nodos.lienzo.superficie.nuOrdenado == null) {
                    Nodos.lienzo.superficie.nuOrdenado = new
Nudo[Nodos.lienzo.superficie.nudo.length]; //se creab el array
                }
                for (int k = 0; k <
Nodos.lienzo.superficie.nuOrdenado.length; k++) {
                    Nodos.lienzo.superficie.nuOrdenado[k] = new
Nudo();
                }
                Nodos.lienzo.superficie.nuOrdenado =
Nodos.lienzo.superficie.nudo.clone();
                Nodos.lienzo.superficie.nuOrdenado =
Nodos.lienzo.superficie.OrdenarNudos(Nodos.lienzo.superficie.nuOrdenad
o);
                verNudos(Nodos.lienzo.superficie.nuOrdenado);
            }
        });
        ////////////////////////////////////////////////////
        btnVerNudos.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                verNudos(Nodos.lienzo.superficie.nudo);
            }
        });

////////////////////////////////////////////////////////////////////
/////
        btnVerExternos.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String vin;
                int numVin = 0;
                for (int k = Nodos.lienzo.superficie.numNuNoExt; k <
Nodos.lienzo.superficie.nudo.length; k++) {
                    int n =
Nodos.lienzo.superficie.nudo[k].getOrden();
                    vin = " ";
                    numVin =
Nodos.lienzo.superficie.nudo[n].vinculo.length;
                    for (int m = 0; m < numVin; m++) {
```

```
                        vin = vin + "(" + String.valueOf(m) + "Âº) " +
String.valueOf(Nodos.lienzo.superficie.nudo[n].vinculo[m].getOrden())
+ " , ";
                    }

                    areaTexto.append("\n Etiqueta " +
String.valueOf(n) + "orden " +
String.valueOf(Nodos.lienzo.superficie.nudo[n].getOrden()) + "," +
Nodos.lienzo.superficie.nudo[n].toPrint() + "\t NÂº vin=" +
String.valueOf(numVin) + "v: " + vin);
                }
            }
        });

//////////////////////////////////////////////////////CALCULO DE
MATRICES***************/
        btnCalMatrices.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                calculoFormas();
            }
        });
        ////////////////////////////////////////////////////////////
        btnCalDinamico.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Nodos.lienzo.superficie.interacciones =
Integer.parseInt(tIteraciones.getText().trim());
                JOptionPane.showMessageDialog(null, "Â¡Tomalo con
calma!"
                        + ".El proceso dura tiempo,segÃºn la
complejidad");
                calculoDinamico();
            }
        });
        /**
         * ******************************************************
         */
        btnBorrarPantalla.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                areaTexto.setText(""); //es la manera mÃ¡s simple de
borrar el texto de la pantalla
            }
        });
        ////////////////////////////////////Genera las masas
        btnMatrizMasas.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {

                //Creacion de la matriz Masa y asignamos a la diagonal
el valor 1. el resto 0
                Nodos.lienzo.superficie.Masa = new
Matriz(Nodos.lienzo.superficie.nudo.length,
Nodos.lienzo.superficie.nudo.length);
                for (int j = 0; j <
Nodos.lienzo.superficie.Masa.numFilas; j++) {
                    for (int k = 0; k <
Nodos.lienzo.superficie.Masa.numColumnas; k++) {
                        if (j != k) {
                            Nodos.lienzo.superficie.Masa.x[j][k] = 0;
                        } else {
                            Nodos.lienzo.superficie.Masa.x[j][k] = 1;
```

```
                        }
                    }
                }
                boolean activar;
                if (chCargaMasa.isSelected()) {
                    activar = true;
                } else {
                    activar = false;
                }
                Nodos.lienzo.superficie.isCarga = activar;
                Nodos.lienzo.superficie.Masa =
Barra.asignarMasaNudos(Nodos.lienzo.superficie.barras, activar);
                areaTexto.append("\nMatriz de Masas asignada a los
nudos ");

areaTexto.append(Nodos.lienzo.superficie.Masa.salidaMatriz());
                btnCalDinamico.setEnabled(true);
            }
        });

        this.btnComprueba.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                test();
            }
        });
    }

    /**
     * ************************************************
     */
    JPanel PonMenu() {
        JPanel panelMenu = new JPanel();
        JMenuBar barraMenu = new JMenuBar();
        JMenu MenuArchivo = new JMenu("Archivos");
        JMenu MenuNavegadores = new JMenu("Navegar");
        JMenu MenuConfiguracion = new JMenu("Confi");
        JMenu MenuCredito = new JMenu("Creditos");
        JMenuItem autorMenuItem = new JMenuItem("Autores");
        JMenuItem nuevoMenuItem = new JMenuItem("Nuevo");
        JMenuItem abrirMenuItem = new JMenuItem("Abrir");
        JMenuItem confiGuardarMenu = new JMenuItem("Salvar
configura");
        JMenuItem confiCargarMenu = new JMenuItem("Abrir Configura");
        dialogo = new FileDialog(Nodos.ventArchivo, "Nodos: abrir
archivo", FileDialog.LOAD);
        JMenuItem guardarMenuItem = new JMenuItem("Guardar");
        JMenuItem exportarMenuItem = new JMenuItem("Exportar DXF");
        JMenuItem salirMenuItem = new JMenuItem("Exit");
        JMenuItem nudosMenuItem = new JMenuItem("Navega Nudos");
        barrasMenuItem = new JMenuItem("Navega Barras");
        JMenuItem eleFinitoMenuItem = new JMenuItem("Navega Elemento
Finito");
        JMenuItem vibraMenuItem = new JMenuItem("Navega Modos Vibra");
        JMenuItem calculoMenuItem = new JMenuItem("Calculos");
        // openMenuItem.addActionListener(this);
        MenuNavegadores.add(nudosMenuItem);
        MenuNavegadores.add(barrasMenuItem);
        MenuNavegadores.add(calculoMenuItem);
        MenuNavegadores.add(vibraMenuItem);
        MenuNavegadores.add(eleFinitoMenuItem);
        MenuArchivo.add(nuevoMenuItem);
```

```
        MenuArchivo.add(abrirMenuItem);
        MenuArchivo.add(guardarMenuItem);
        MenuArchivo.add(exportarMenuItem);
        MenuArchivo.add(salirMenuItem);
        barraMenu.add(MenuArchivo);
        barraMenu.add(MenuNavegadores);

        barraMenu.add(MenuConfiguracion);
        barraMenu.add(MenuCredito);
        MenuCredito.add(autorMenuItem);
        MenuConfiguracion.add(confiGuardarMenu);
        MenuConfiguracion.add(confiCargarMenu);
        panelMenu.add(barraMenu);

////////////////////////////////////////////////////////////
        vibraMenuItem.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if (!Nodos.ventVibracion.isActive()) {
                    ventVibra();
                }
            }
        });

        eleFinitoMenuItem.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                ventEleFinito();
            }
        });


////////////////////////////////////////////////////////////////
        calculoMenuItem.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if (Nodos.ventCalculo == null) {
                    Nodos.ventCalculo = new Ventana("Tesis
Nodos;CÃ¡lculos", 1200, 600, Nodos.paca, false);
                };
                verVentanaCalculo = !verVentanaCalculo;
                Nodos.ventCalculo.setVisible(verVentanaCalculo);
            }
        });
        ////////////////////////////////////////////////////////
        nudosMenuItem.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if (Nodos.ventNavegacion == null) {
                    Nodos.ventNavegacion = new Ventana("Tesis Nodos:
ajuste de nudos", 600, 250, Nodos.paco, false);
                }
                verVentanaNudos = !verVentanaNudos;
                Nodos.ventNavegacion.setVisible(verVentanaNudos);
            }
        });

////////////////////////////////////////////////////////////////
        barrasMenuItem.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if (Nodos.ventBarras == null) {
                    Nodos.ventBarras = new Ventana("Tesis Nodos:
ajuste de barras", 875, 120, Nodos.pabar, false);
                }
                verVentanaBarras = !verVentanaBarras;
```

```
                Nodos.ventBarras.setVisible(verVentanaBarras);
            }
        });
        ////////////////////////////////////////////////
        salirMenuItem.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.exit(1);
            }
        });
        ////////////////////////////////////leer
archivo//////////////////////////////////////////////////
        abrirMenuItem.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Nodos.empezar();
                // if(Nodos.lienzo!=null) Nodos.lienzo=null;
                Nodos.lienzo.superficie = new Superficie();
                Nodos.lienzo.superficie = (Superficie)
leerArchivo("Nodos: Abrir Archivo", "./Tesis", "*.nod");

tAnchoRectangulo.setText(String.valueOf(Nodos.lienzo.superficie.Ancho)
);

tAltoRectangulo.setText(String.valueOf(Nodos.lienzo.superficie.Alto));

tLongEleFinitoX.setText(String.valueOf(Nodos.lienzo.superficie.dX));

tLongEleFinitoY.setText(String.valueOf(Nodos.lienzo.superficie.dY));

tDensidad.setText(String.valueOf(Nodos.lienzo.superficie.densiFuerzaBa
rraForma));

tGrosorEF.setText(String.valueOf(Nodos.lienzo.superficie.grosorEF));

tDensiEF.setText(String.valueOf(Nodos.lienzo.superficie.densiEF));
                // panelBotones.setVisible(true);
                Nodos.paen.PonVentanaInicio().setVisible(true);
                Nodos.lienzo.repaint();
                Nodos.lienzo.boolBarras = true;
                Nodos.lienzo.boolNudos = true;
                Nodos.lienzo.repaint();
            }
        });
        ////////////////////////////////////////////////////guardar
el archivo
        guardarMenuItem.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                guardarArchivo("Nodos: Guardar Archivo", "/Tesis",
"*.nod");
            }
        });
        ////////////////////////////////////NUEVO/////NUEVO/////
        nuevoMenuItem.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (Nodos.lienzo.superficie.nudo != null) {
                    Nodos.lienzo.superficie.nudo = null;
                }
                if (Nodos.lienzo.superficie.nuOrdenado != null) {
                    Nodos.lienzo.superficie.nuOrdenado = null;
```

```
                }
                if (Nodos.lienzo != null) {
                    Nodos.lienzo = null;
                }
                Nodos.lienzo = new Lienzo();
                Nodos.paen.PonVentanaInicio().setVisible(true);

tGrosorEF.setText(String.valueOf(Nodos.lienzo.superficie.grosorEF));

tDensiEF.setText(String.valueOf(Nodos.lienzo.superficie.densiEF));
                // panelBotones.setVisible(true);
            }
        });

        autorMenuItem.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                muestraCreditos();
            }
        });

        return panelMenu;
    }


////////////////////////////////////////////////////////////////////
///
    void muestraCreditos() {
        JFrame ventCreditos = new JFrame("Tesis Nodos: Creditos");
        ventCreditos.setIconImage(new
javax.swing.ImageIcon(getClass().getResource("imagenes/nodos2.png")).g
etImage());
        ventCreditos.setTitle("NODOS:CrÃ©ditos");
        ventCreditos.setSize(400, 200);
        JPanel pCreditos = new JPanel();
        pCreditos.setLayout(new FlowLayout());
        pCreditos.add(new JLabel("                    NODOS
"));
        pCreditos.add(new JLabel("Programa para el diseÃ±o de redes,
catenarias, puentes y cubiertas "));
        pCreditos.add(new JLabel("        VersiÃ³n V.1 compilaciÃ³n
31-12-2013           "));
        pCreditos.add(new JLabel("(c) Ernesto MuÃ±iz y Julio MuÃ±iz
2013-14.     "));
        pCreditos.add(new JLabel("Sobre una tesis doctoral de Ernesto
MuÃ±iz.    "));
        pCreditos.add(new JLabel("Todos los derechos reservados
"));
        ventCreditos.add(pCreditos);
        ventCreditos.setVisible(true);
    }


////////////////////////////////////////////////////////////////////
//////
    public JPanel PonVentanaInicio() {

        pSalida.setBackground(Color.YELLOW);
        pSalida.setLayout(new GridLayout(12, 2, 0, 4));

//GridLayout(int rows, int cols, int hgap, int vgap)
```

```
        // cbTipoEleFinito.addItem("Rectangulo");
        // cbTipoEleFinito.addItem("Triangulo");
        // cbTipoEleFinito.addItem("Rec. con diagonales");
        //panelBotones.add(new JLabel("")); panelBotones.add(new
JLabel(""));
        pSalida.add(lAnchoRectangulo);
        pSalida.add(tAnchoRectangulo);
        pSalida.add(lAltoRectangulo);
        pSalida.add(tAltoRectangulo);
        pSalida.add(lTipoEleFinito);;
        pSalida.add(cbTipoEleFinito);
        JPanel pEleFinito = new JPanel();
        pEleFinito.add(tLongEleFinitoX);
        pEleFinito.add(tLongEleFinitoY);
        pSalida.add(lLongEleFinito);
        pSalida.add(pEleFinito);
        pSalida.add(lGrosorEF);
        pSalida.add(tGrosorEF);
        //  panelBotones.
add(lIteraciones);panelBotones.add(tIteraciones);
        pSalida.add(lDensidad);
        pSalida.add(tDensidad);
        //pSalida.add(new JLabel("SecciÃ³n Barra cm2") );
        pSalida.add(lSeccion);
        pSalida.add(tBarraSeccion);
        pSalida.add(lDensiEF);
        pSalida.add(tDensiEF);
        pSalida.add(lMaterial);
        pSalida.add(cbMaterial);
        //panelBotones.add(new JLabel("INICIAR"));
        pSalida.add(btnCalculo);
        btnCalculo.setIcon(arrancar);
        // panelBotones.add(new JLabel("Actualizar"));
        pSalida.add(btnActualizar);
        btnActualizar.setIcon(actualizar);
////////////////////////////////////////////////////////////////
//////////////////////////
        btnCalculo.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Nodos.empezar();
                calculoInicio(); // llamada a la funciÃ³n que inicia
el programa

tAnchoRectangulo.setText(String.valueOf(Nodos.lienzo.superficie.Ancho)
);

tAltoRectangulo.setText(String.valueOf(Nodos.lienzo.superficie.Alto));

                Nodos.lienzo.superficie.densiFuerzaBarraForma =
Double.valueOf(tDensidad.getText()).doubleValue();
                Nodos.lienzo.superficie.grosorEF = 0.0001 *
Double.valueOf(tGrosorEF.getText()).doubleValue();
                Nodos.lienzo.superficie.densiEF =
Double.valueOf(tDensiEF.getText()).doubleValue();
                Nodos.lienzo.repaint();
                //Nodos.lienzo.superficie.setNumBarras(0);
            }
        });
////////////////////////////////////////////////////////////
        btnActualizar.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
```

```
                if (chCargaMasa.isSelected()) {
                    Nodos.lienzo.superficie.isCarga = true;
                } else {
                    Nodos.lienzo.superficie.isCarga = false;
                }
                // llamada a la funciÃ³n que inicia el programa
                str = (String) cbTipoEleFinito.getSelectedItem();
                if (str == "Triangulo") {
                    cTipoEleFinito = 't';
                }
                if (str == "Rectangulo") {
                    cTipoEleFinito = 'c';
                }
                if (str == "Rec. con diagonales") {
                    cTipoEleFinito = 'd';
                }
                Nodos.lienzo.superficie.densiFuerzaBarraForma =
Double.valueOf(tDensidad.getText()).doubleValue();

Nodos.lienzo.superficie.setTipoEleFinito(cTipoEleFinito);
                Nodos.lienzo.superficie.densiFuerzaBarraForma =
Double.valueOf(tDensidad.getText()).doubleValue();
                Nodos.lienzo.superficie.grosorEF =
Double.valueOf(tGrosorEF.getText()).doubleValue();

Nodos.lienzo.superficie.setGeometria(Nodos.lienzo.superficie.getTipoEl
eFinito());
                Nodos.lienzo.superficie.densiEF =
Double.valueOf(tDensiEF.getText()).doubleValue();
                Nodos.lienzo.superficie.Ancho =
Double.valueOf(tAnchoRectangulo.getText().trim());
                Nodos.lienzo.superficie.Ancho =
Double.valueOf(tAltoRectangulo.getText().trim());
                Nodos.lienzo.repaint();
            }
        });
        return pSalida;
    }

    ////////////////////////////////VENTANA
GRAFICA/////////////////////////////////////////
    public void PonPizarraGrafica() {
        add(Nodos.lienzo);
    }


////////////////////////////////////////////////////////////////////
////////////////////////////////
    public JPanel PonBotonesGraficos() {
        JPanel pBotones = new JPanel();
        if (cbPlanos == null) {
            cbPlanos = new JComboBox();
            cbPlanos.addItem("XYZ");
            cbPlanos.addItem("XY");
            cbPlanos.addItem("XZ");
            cbPlanos.addItem("YZ");
        }
        btnReacciones = new JButton("");
        btnImagenVibra = new JButton("");
        btnImagenVibra.setIcon(new
javax.swing.ImageIcon(getClass().getResource("imagenes/vibra.png")));
```

```
        btnNudos.setIcon(new
javax.swing.ImageIcon(getClass().getResource("imagenes/nudos.png")));
        btnEjes.setIcon(new
javax.swing.ImageIcon(getClass().getResource("imagenes/eje.png")));
        btnReacciones.setIcon(new
javax.swing.ImageIcon(getClass().getResource("imagenes/reaccion.png"))
);
        btnEleFinitos = new JButton("EF");
        btnEscalaMas.setIcon(lupaMas);
        btnEscalaMenos.setIcon(lupaMenos);
        pBotones.add(cbPlanos);
        pBotones.add(btnExit);
        pBotones.add(btnEjes);
        pBotones.add(btnNudos);
        pBotones.add(btnNudosSalida);
        pBotones.add(btnNuevoExterno);
        pBotones.add(btnBarra);
        pBotones.add(btnEleFinitos);
        pBotones.add(chAgotamiento);
        pBotones.add(btnBorrar);
        pBotones.add(btnCoordenadas);
        pBotones.add(btnEscalaMas);
        pBotones.add(btnEscalaMenos);
        pBotones.add(btnNumNudo);

        pBotones.add(btnReacciones);//
btnReacciones.setEnabled(false);
        pBotones.add(btnImagenVibra);//
btnReacciones.setEnabled(false);

        pBotones.add(btnEjecutar);
        btnEjecutar.setBackground(Color.PINK);
        btnEjecutar.setIcon(new
javax.swing.ImageIcon(getClass().getResource("imagenes/ejecutar.gif"))
);

        pBotones.add(lCoordenadas);

        pBotones.add(chAlerta);
        ////////////////////////AÃ±adimos las acciones de los
botones//////////////////////
        btnBarra.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Nodos.lienzo.boolBarras = !Nodos.lienzo.boolBarras;
                if (chAgotamiento.isSelected()) {
                    Nodos.lienzo.bAgota = true;
                } else {
                    Nodos.lienzo.bAgota = false;
                }
                Nodos.lienzo.repaint();
            }
        });
        btnEleFinitos.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {

                Nodos.lienzo.boolEleFinito =
!Nodos.lienzo.boolEleFinito;
                ventEleFinito();
                Nodos.lienzo.repaint();
            }
        });
```

```
        ////////////////////////////////////////////////////
        btnReacciones.addActionListener(new
java.awt.event.ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Nodos.lienzo.boolReacciones =
!Nodos.lienzo.boolReacciones;
                Nodos.lienzo.repaint();
                //    }
            }
        });

        btnImagenVibra.addActionListener(new
java.awt.event.ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Nodos.lienzo.boolVibra=!Nodos.lienzo.boolVibra;
//  if (Nodos.lienzo.boolVibra) ventVibra();
                ventVibra();
                //    }
            }
        });

////////////////////////////////////////////////////////////////
        ////////////////////////////////////////////////////////////
        btnNuevoExterno.addActionListener(new
java.awt.event.ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (navegaNudos != null) {
                    navegaNudos = null;
                }
                navegaNudos = new NavegaNudos(900, 350, false);
                navegaNudos.ponNavegaNudos();
                navegaNudos.setVisible(true);
                //    }
            }
        });

////////////////////////////////////////////////////////////////
        cbPlanos.addActionListener(new java.awt.event.ActionListener()
{
            @Override
            public void actionPerformed(java.awt.event.ActionEvent e)
{
                if (cbPlanos.getSelectedItem() == "XYZ") {
                    Nodos.lienzo.bXYZ = true;
                    Nodos.lienzo.bXY = false;
                    Nodos.lienzo.bXZ = false;
                    Nodos.lienzo.bYZ = false;
                }
                if (cbPlanos.getSelectedItem() == "XY") {
                    Nodos.lienzo.bXYZ = false;
                    Nodos.lienzo.bXY = true;
                    Nodos.lienzo.bXZ = false;
                    Nodos.lienzo.bYZ = false;
                }
                if (cbPlanos.getSelectedItem() == "XZ") {
                    Nodos.lienzo.bXYZ = false;
                    Nodos.lienzo.bXY = false;
                    Nodos.lienzo.bXZ = true;
```

```
                    Nodos.lienzo.bYZ = false;
                }
                if (cbPlanos.getSelectedItem() == "YZ") {
                    Nodos.lienzo.bXYZ = false;
                    Nodos.lienzo.bXY = false;
                    Nodos.lienzo.bXZ = false;
                    Nodos.lienzo.bYZ = true;
                }
                Nodos.lienzo.repaint();
            }
        });
////////////////////////////////////////////
        pBotones.add(btnMenosAngulo);
        pBotones.add(lAngulo);
        pBotones.add(btnMasAngulo);
        rbAngulo.add(chGiroX);
        rbAngulo.add(chGiroY);
        rbAngulo.add(chGiroZ);
        pBotones.add(chGiroX);
        pBotones.add(chGiroY);
        pBotones.add(chGiroZ);

////////////////////////////////////////////////////////////////////
/
        btnMenosAngulo.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (chGiroX.isSelected()) {
                    Nodos.lienzo.giroX = Nodos.lienzo.giroX - 5;
                    if (Nodos.lienzo.giroX < 0) {
                        Nodos.lienzo.giroX = 360;
                    }

lAngulo.setText(String.valueOf(Nodos.lienzo.giroX));
                }
                if (chGiroY.isSelected()) {
                    Nodos.lienzo.giroY = Nodos.lienzo.giroY - 5;
                    if (Nodos.lienzo.giroY < 0) {
                        Nodos.lienzo.giroY = 360;
                    }

lAngulo.setText(String.valueOf(Nodos.lienzo.giroY));
                }
                if (chGiroZ.isSelected()) {
                    Nodos.lienzo.giroZ = Nodos.lienzo.giroZ - 5;
                    if (Nodos.lienzo.giroZ < 0) {
                        Nodos.lienzo.giroZ = 360;
                    }

lAngulo.setText(String.valueOf(Nodos.lienzo.giroZ));
                }
                Nodos.lienzo.angX = Nodos.lienzo.giroX * Math.PI /
180; //pasamos de grado a radianes
                Nodos.lienzo.angY = Nodos.lienzo.giroY * Math.PI /
180; //pasamos de grado a radianes
                Nodos.lienzo.angZ = Nodos.lienzo.giroZ * Math.PI /
180; //pasamos de grado a radianes
                Nodos.lienzo.repaint();
            }
        });
```

```
////////////////////////////////////////////////////////////
        btnMasAngulo.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (chGiroX.isSelected()) {
                    Nodos.lienzo.giroX = Nodos.lienzo.giroX + 5;
                    if (Nodos.lienzo.giroX > 360) {
                        Nodos.lienzo.giroX = 0;
                    }

lAngulo.setText(String.valueOf(Nodos.lienzo.giroX));
                }
                if (chGiroY.isSelected()) {
                    Nodos.lienzo.giroY = Nodos.lienzo.giroY + 5;
                    if (Nodos.lienzo.giroY > 360) {
                        Nodos.lienzo.giroY = 0;
                    }

lAngulo.setText(String.valueOf(Nodos.lienzo.giroY));
                }
                if (chGiroZ.isSelected()) {
                    Nodos.lienzo.giroZ = Nodos.lienzo.giroZ + 5;
                    if (Nodos.lienzo.giroZ > 360) {
                        Nodos.lienzo.giroZ = 0;
                    }

lAngulo.setText(String.valueOf(Nodos.lienzo.giroZ));
                }

                Nodos.lienzo.angX = Nodos.lienzo.giroX * Math.PI /
180; //pasamos de grado a radianes
                Nodos.lienzo.angY = Nodos.lienzo.giroY * Math.PI /
180; //pasamos de grado a radianes
                Nodos.lienzo.angZ = Nodos.lienzo.giroZ * Math.PI /
180; //pasamos de grado a radianes
                Nodos.lienzo.repaint();
            }
        });
        ////////////////////////////////////////////////
        this.cbTipoEleFinito.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String str = "                                    ";
                cbTipoEleFinito = (JComboBox) e.getSource();
//TipoEleFinito);
                str = (String) cbTipoEleFinito.getSelectedItem();
                if (str == "Triangulo") {
                    cTipoEleFinito = 't';
                }
                if (str == "Rectangulo") {
                    cTipoEleFinito = 'c';
                }
                if (str == "Rec. con diagonales") {
                    cTipoEleFinito = 'd';
                }
            }
        });
        ////////////////////////////////////////////////////
        btnEjes.addActionListener(new ActionListener() {
            @Override
```

```
            public void actionPerformed(ActionEvent e) {
                Nodos.lienzo.boolEjes = !Nodos.lienzo.boolEjes;

lCoordenadas.setText(Nodos.lienzo.dameCoordenada(true));
                Nodos.lienzo.ejes.setVisible(Nodos.lienzo.boolEjes);
                Nodos.lienzo.repaint();
                if (Nodos.lienzo.boolEjes) {
                    btnEjes.setText("=> Ejes");
                } else {
                    btnEjes.setText("<= Ejes");
                }
            }
        });
        ////////////////////////////////////////////////
        btnExit.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.exit(1);
            }
        });
        ////////////////////////////////////////////////////
        btnNumNudo.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Nodos.lienzo.boolNumNudo = !Nodos.lienzo.boolNumNudo;
                Nodos.lienzo.repaint();
            }
        });
        ////////////////////////////////////////////////////
        btnEjecutar.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (chCargaMasa.isSelected()) {
                    Nodos.lienzo.superficie.isCarga = true;
                } else {
                    Nodos.lienzo.superficie.isCarga = false;
                }
                Paneli.btnCalMatrices.doClick();
                Nodos.lienzo.repaint();
                Paneli.barrasMenuItem.doClick();
            }
        });
        ////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////
        btnCoordenadas.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Nodos.lienzo.boolCoordenadas =
!Nodos.lienzo.boolCoordenadas;
                Nodos.lienzo.repaint();
                if (Nodos.lienzo.boolCoordenadas) {
                    btnCoordenadas.setText("A<=(x,y,z)");
                } else {
                    btnCoordenadas.setText("A=>(x,y,z)");
                }
            }
        });
        ////////////////////////////////////////////
        btnEscalaMas.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Nodos.lienzo.escala = Nodos.lienzo.escala + 1;
                Nodos.lienzo.repaint();
            }
```

```
        });
        ////////////////////////////////////////////////////
        btnEscalaMenos.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if (Nodos.lienzo.escala >= 1) {
                    Nodos.lienzo.escala = Nodos.lienzo.escala - 1;
                } else {
                    Nodos.lienzo.escala = 1;
                }
                Nodos.lienzo.repaint();
            }
        });
        ////////////////////////////////////////////////////
        btnNudos.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Nodos.lienzo.boolNudos = !Nodos.lienzo.boolNudos;
                Nodos.lienzo.repaint();
                if (Nodos.lienzo.boolNudos) {
                    Nodos.lienzo.boolNudosSalida = false;
                    btnNudos.setText("=>o");
                } else {
                    btnNudos.setText("<=o");
                }
            }
        });
        ////////////////////////////////////////////////////
        btnNudosSalida.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Nodos.lienzo.boolNudosSalida =
!Nodos.lienzo.boolNudosSalida;
                Nodos.lienzo.repaint();
                if (Nodos.lienzo.boolNudosSalida) {
                    btnNudosSalida.setText("=>OUT");
                    Nodos.lienzo.boolNudos = false;
                } else {
                    btnNudos.setText("<=OUT");
                }
            }
        });

        this.btnBorrar.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Nodos.lienzo.boolBorrar = !Nodos.lienzo.boolBorrar;
                if (Nodos.lienzo.boolBorrar == false) {
                    btnBorrar.setText("Borra");
                }
                if (Nodos.lienzo.boolBorrar == true) {
                    btnBorrar.setText("Dibuja");
                }
                Nodos.lienzo.repaint();
            }
        });

        return pBotones;
    }


////////////////////////////////////////////////////////////////
/
    void ventVibra() {
```

```
        Nodos.lienzo.superficie.vibra = new
Nudo[Nodos.lienzo.superficie.nudo.length];
        for (int n = 0; n < Nodos.lienzo.superficie.nudo.length; n++)
{
            Nodos.lienzo.superficie.vibra[n] = new Nudo();

Nodos.lienzo.superficie.vibra[n].equals(Nodos.lienzo.superficie.nuSali
da[n]);
        }
        JFrame ventVibracion = new JFrame("Tesis Nodos: Modos de
vibraciÃ³n");
        ventVibracion.setBackground(Color.green);
        final JButton btnAceptarVibra = new JButton("=>");
        final JButton btnAtrasVibra = new JButton("<=");
        final JButton btnAnimaVibra = new JButton("<=>");
        final JButton btnTmas = new JButton("x10");
        final JButton btnTmenos = new JButton("/10");
        final JPanel pSalida = new JPanel();
        pSalida.setBackground(Color.GREEN);
        pSalida.setEnabled(false);
        final JPanel pVibraXYZ = new JPanel();
        pVibraXYZ.setBackground(Color.YELLOW);
        final JTextField tModoV = new JTextField("0", 4);
        final JComboBox comboModoV = new JComboBox();
        final JTextField tEscalaV = new JTextField("20", 4);
        final JTextField tEscalaY = new JTextField("0.5", 4);
        final JTextField tEscalaZ = new JTextField("1", 4);
        final JTextField tEscalaX = new JTextField("0.5", 4);
        final JLabel lEscalaAmplitud = new JLabel("Escala Amplitud=
");
        final JTextField tEscalaAmplitud = new JTextField("0.00", 4);
        final JLabel lAmplitudMax = new JLabel("Amplitud mÃ¡xima= ");
        final JTextField tAmplitudMax = new JTextField("0.00", 4);
        final JLabel lEnergiaSis = new JLabel("EnergÃa oscilaciÃ³n=
");
        final JTextField tEnergiaSis = new JTextField("0.00", 4);
        final JLabel lFrecAdapt = new JLabel("Frecuencia adaptada= ");
        final JTextField tFrecAdapt = new JTextField("0.00", 4);
        rbAnima = new JRadioButton("AnimaciÃ³n", true);
        lPeriodo = new JLabel();
        ventVibracion.setIconImage(new
javax.swing.ImageIcon(getClass().getResource("imagenes/nodos2.png")).g
etImage());
        ventVibracion.setTitle("Nodos: modos de vibraciÃ³n ");
        ventVibracion.setSize(450, 225);
        ventVibracion.setResizable(false);
        JPanel pVibra = new JPanel();
        pVibra.setLayout(new FlowLayout());
        //pVibra.add(new JLabel(" Eligue el nÃºmero del modo de
vibraciÃ³n  "));pVibra.add(tModoV);

        for (int i = 0; i <
Nodos.lienzo.superficie.valoresPropios.length; i++) {
            comboModoV.addItem("Modo " + i + " , " +
String.valueOf(Nodos.lienzo.superficie.valoresPropios[i]));
        }
        pVibra.add(comboModoV);
        pVibra.add(new JLabel("Escala Visual "));
        pVibra.add(tEscalaV);
        pVibraXYZ.add(new JLabel("Factor masa X "));
        pVibraXYZ.add(tEscalaX);
```

```
        pVibraXYZ.add(new JLabel("Factor masa Y "));
        pVibraXYZ.add(tEscalaY);
        pVibraXYZ.add(new JLabel("Factor masa Z "));
        pVibraXYZ.add(tEscalaZ);
        pSalida.setLayout(new GridLayout(4, 2, 2, 5));
        pSalida.add(lEscalaAmplitud);
        pSalida.add(tEscalaAmplitud);
        pSalida.add(lAmplitudMax);
        pSalida.add(tAmplitudMax);
        pSalida.add(lEnergiaSis);
        pSalida.add(tEnergiaSis);
        pSalida.add(lFrecAdapt);
        pSalida.add(tFrecAdapt);
        pVibra.add(pVibraXYZ);
        pVibra.add(pSalida);
        pVibra.add(btnAceptarVibra);
        pVibra.add(btnAtrasVibra);
        ventVibracion.add(pVibra);
        ventVibracion.setVisible(true);
        int nA = Nodos.lienzo.superficie.numApoyos;
        pVibra.add(rbAnima);
        pVibra.add(btnAnimaVibra);
        pVibra.add(lPeriodo);
        pVibra.add(btnTmas);
        pVibra.add(btnTmenos);
// indica el peridod
//aÃ±ado los eventos de los botones y el combo.
        comboModoV.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                int mod = comboModoV.getSelectedIndex();
                double resultado =
Nodos.lienzo.superficie.escalaAmplitud(Nodos.lienzo.superficie.barras,
mod, Nodos.lienzo.superficie.AL);
                double resultado2 =
Nodos.lienzo.superficie.amplitudMax(Nodos.lienzo.superficie.VP, mod,
resultado);
                double resultado3 = 0.0;
                tEscalaAmplitud.setText(String.valueOf(resultado));

tEnergiaSis.setText(String.valueOf(Nodos.lienzo.superficie.EPE[mod] /
(resultado * resultado)));
                tAmplitudMax.setText(String.valueOf(resultado2));
                double sumaFactores = Nodos.lienzo.escalaX +
Nodos.lienzo.escalaY + Nodos.lienzo.escalaZ;
                if (sumaFactores == 0) {
                    JOptionPane.showMessageDialog(null, "Hay que
introducir al menos un valor fe cactor de masa diferente a 0 y uno al
menos igual a 1");
                }
                if (sumaFactores > 3 || sumaFactores < 1) {
                    JOptionPane.showMessageDialog(null, "Valores de
factores de masas incongruentes");
                } else {
                    resultado3 =
Nodos.lienzo.superficie.valoresPropios[mod] *
Nodos.lienzo.superficie.FactorMasa / sumaFactores;

tFrecAdapt.setText(String.valueOf(Math.sqrt(resultado3) / (2 *
Math.PI)));
                }
```

```
            }
        });
        //la siguiente funciÃ³n debe ir aquÃ para que se inicialice
automÃ¡ticamente el combo
        comboModoV.setSelectedIndex(0);
//
Nodos.lienzo.vibra=Nodos.lienzo.superficie.OrdenarNudos(Nodos.lienzo.v
ibra);
////////////////////////////////////////////////////////////////////
///
////////////////////////////////////////////////////////////////////
///
        btnTmas.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Nodos.lienzo.escalaT = 10 * Nodos.lienzo.escalaT;
            }
        });
        btnTmenos.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Nodos.lienzo.escalaT = 0.1 * Nodos.lienzo.escalaT;
            }
        });

        btnAceptarVibra.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Nodos.lienzo.boolVibra = false;

                m = Nodos.lienzo.nModoVibra =
comboModoV.getSelectedIndex();
                es = Nodos.lienzo.escalaVibra =
Double.valueOf(tEscalaV.getText().trim());
                esX = Nodos.lienzo.escalaX =
Double.valueOf(tEscalaX.getText().trim());
                esY = Nodos.lienzo.escalaY =
Double.valueOf(tEscalaY.getText().trim());
                esZ = Nodos.lienzo.escalaZ =
Double.valueOf(tEscalaZ.getText().trim());
                Nodos.lienzo.superficie.vibra =
Nodos.lienzo.superficie.restaura();
                Nodos.lienzo.superficie.vibra =
Nodos.lienzo.superficie.deforma(m, es, esX, esY, esZ);
                Nodos.lienzo.repaint();
                Nodos.lienzo.bDibujoDeformado = true;
            }
        });
////////////////////////////////////////////////////////////////////
////////////////////
        btnAtrasVibra.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Nodos.lienzo.boolVibra = false;
                Nodos.lienzo.superficie.vibra =
Nodos.lienzo.superficie.restaura();
                Nodos.lienzo.repaint();
                Nodos.lienzo.bDibujoDeformado = false;
            }
        });
```

```
        btnAnimaVibra.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                bAnima = !bAnima;
                Nodos.lienzo.boolVibra = bAnima;
                rbAnima.setSelected(bAnima);
                bAnima = rbAnima.isSelected();

                if (bAnima) {
                    Nodos.lienzo.arrancarHilo();
                } else {
                    Nodos.lienzo.pararHilo();
                }
                Nodos.lienzo.repaint();

            }
        });

        rbAnima.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (rbAnima.isSelected()) {
                    bAnima = true;
                    btnAnimaVibra.setEnabled(true);
                } else {
                    bAnima = false;
                    btnAnimaVibra.setEnabled(false);
                }
            }
        });

    }

////////////////////////////////////////////////////////////////////
    ////////////////////////////////VENTANA DE NAVEGACION POR LOS
NUDOS//////////////////////////////

    public JPanel ponCargas() {
        JPanel panelCargas = new JPanel();
        panelCargas.setBackground(Color.GREEN);
        panelCargas.add(lCargaX);
        panelCargas.add(tCargaX);
        panelCargas.add(lCargaY);
        panelCargas.add(tCargaY);
        panelCargas.add(lCargaZ);
        panelCargas.add(tCargaZ);
        return panelCargas;
    }
////////////////////////////////////////////////////////////////////
/

    public JPanel ponCoordenadas() {
        JPanel panelCoor = new JPanel();
        panelCoor.setBackground(Color.CYAN);
        panelCoor.add(lCoorX);
        panelCoor.add(tCoorX);
        panelCoor.add(lCoorY);
        panelCoor.add(tCoorY);
        panelCoor.add(lCoorZ);
        panelCoor.add(tCoorZ);
```

```
        return panelCoor;
    }
    ////////////////////////////////////////PANEL de
BARRAS/////////////////////////////////////////

    void PonPanelBarras() {
        JLabel lAviso = new JLabel("Barra,por nÃºmero,inicio o
final\n");
        JPanel pNavega = new JPanel();
        pNavega.setBackground(Color.YELLOW);
        JPanel pMuestra = new JPanel();
        pNavega.add(lAviso);
        pNavega.add(lNumBarra);
        pNavega.add(tNumBarra);
        pNavega.add(btnBarraInicio);
        pNavega.add(btnBarraMenos);
        pNavega.add(btnBarraMas);
        pNavega.add(btnBarraFinal);
        pNavega.add(lNuInicio);
        pNavega.add(tNuInicio);
        pNavega.add(lNuFin);
        pNavega.add(tNuFin);

        pMuestra.setLayout(new GridLayout(5, 4, 10, 5)); //filas,
columnas,separaciÃ³n columnas ,separaciÃ³n filas
        pMuestra.add(lBarraEtiqueta);
        pMuestra.add(tBarraEtiqueta);
        tBarraEtiqueta.setEditable(false);
        pMuestra.add(chTraccion);
        pMuestra.add(cbMaterial);
        pMuestra.add(new JLabel("Den.Fuerza Form."));
        pMuestra.add(tBarraDensFuerzaF);
        pMuestra.add(new JLabel("Den.Fuerza Calc."));
        pMuestra.add(tBarraDensFuerzaC);
        tBarraDensFuerzaC.setEditable(false);
        pMuestra.add(lBarraLongitud);
        pMuestra.add(tBarraLongitud);
        pMuestra.add(lBarraFuerzaAxial);
        pMuestra.add(tBarraFuerzaAxial);
        tBarraLongitud.setEditable(false);
        tBarraFuerzaAxial.setEditable(false);
        pMuestra.add(lBarraDensidad);
        pMuestra.add(tBarraDensiMasa);
        pMuestra.add(lBarraModuloElastico);
        pMuestra.add(tBarraModuloElastico);
        pMuestra.add(lBarraLimiteElastico);
        pMuestra.add(tBarraLimiteElastico);
        pMuestra.add(lBarraSeguridad);
        pMuestra.add(tBarraSeguridad);
        pMuestra.add(new JLabel("Seccion cm2"));
        pMuestra.add(tBarraSeccionEntrada);
        pMuestra.add(new JLabel("Seccion Calculo cm2"));
        pMuestra.add(tBarraSeccionC);
        tBarraSeccionC.setEditable(false);
        pMuestra.add(new JLabel("Ratio Aprovechamiento"));
        pMuestra.add(tBarraRatio);
        tBarraRatio.setEditable(false);
        pMuestra.add(new JLabel("Masa (Kg)"));
        pMuestra.add(tBarraMasa);
        tBarraMasa.setEditable(false);
        pMuestra.add(new JLabel("Acortamiento inicial mm"));
```

```
        pMuestra.add(tBarraAcortamiento);
        tBarraAcortamiento.setEditable(false);
        ////////////////////////////
        setLayout(new BorderLayout());
        add(pNavega, BorderLayout.NORTH);
        add(pMuestra, BorderLayout.CENTER);
        add(btnBarraAceptarCambio, BorderLayout.SOUTH);
////////////////////////////////////////

        this.tNumBarra.addKeyListener(new java.awt.event.KeyAdapter()
{
            public void keyReleased(java.awt.event.KeyEvent evt) {
                int t =
Nodos.lienzo.superficie.barras[Integer.valueOf(tNumBarra.getText().tri
m())].getOrden();

tNuInicio.setText(String.valueOf(Nodos.lienzo.superficie.barras[t].ori
gen.getOrden()));

tNuFin.setText(String.valueOf(Nodos.lienzo.superficie.barras[t].fin.ge
tOrden()));

chTraccion.setSelected(Nodos.lienzo.superficie.barras[t].traccion);

tBarraEtiqueta.setText(Nodos.lienzo.superficie.barras[t].getEtiqueta()
);

Nodos.lienzo.superficie.barras[t].setDensiFuerza(Nodos.lienzo.superfic
ie.DFF.getElemento(t, t));

tBarraDensFuerzaF.setText(String.valueOf(Nodos.lienzo.superficie.barra
s[t].getDensiFuerzaF()));
                Nodos.lienzo.nBarActiva = t;
                Nodos.lienzo.repaint();
                numbar = t;
            }
        });

        this.tNuInicio.addKeyListener(new java.awt.event.KeyAdapter()
{
            public void keyReleased(java.awt.event.KeyEvent evt) {
                int t = numbar;
                for (int b = 0; b <
Nodos.lienzo.superficie.barras.length; b++) {
                    if
(Nodos.lienzo.superficie.barras[b].getOrigen().getOrden() ==
Integer.valueOf(tNuInicio.getText().trim())) {
                        t = b;
                        break;
                    }
                }
                tNumBarra.setText(String.valueOf(t));

tNuFin.setText(String.valueOf(Nodos.lienzo.superficie.barras[t].getFin
().getOrden()));

chTraccion.setSelected(Nodos.lienzo.superficie.barras[t].traccion);

tBarraEtiqueta.setText(Nodos.lienzo.superficie.barras[t].getEtiqueta()
);
```

```
Nodos.lienzo.superficie.barras[t].setDensiFuerza(Nodos.lienzo.superfic
ie.DFF.getElemento(t, t));

tBarraDensFuerzaF.setText(String.valueOf(Nodos.lienzo.superficie.barra
s[t].getDensiFuerzaF()));
                Nodos.lienzo.nBarActiva = t;
                Nodos.lienzo.repaint();
                numbar = t;
            }
        });
        ////////////////////////////////////////////////////
        this.tNuFin.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyReleased(java.awt.event.KeyEvent evt) {
                int t = numbar;
                for (int b = 0; b <
Nodos.lienzo.superficie.barras.length; b++) {
                    if
(Nodos.lienzo.superficie.barras[b].getFin().getOrden() ==
Integer.valueOf(tNuFin.getText().trim())) {
                        t = b;
                        break;
                    }
                }
                tNumBarra.setText(String.valueOf(t));

tNuInicio.setText(String.valueOf(Nodos.lienzo.superficie.barras[t].get
Origen().getOrden()));

chTraccion.setSelected(Nodos.lienzo.superficie.barras[t].traccion);

tBarraEtiqueta.setText(Nodos.lienzo.superficie.barras[t].getEtiqueta()
);

Nodos.lienzo.superficie.barras[t].setDensiFuerza(Nodos.lienzo.superfic
ie.DFF.getElemento(t, t));

tBarraDensFuerzaF.setText(String.valueOf(Nodos.lienzo.superficie.barra
s[t].getDensiFuerzaF()));
                Nodos.lienzo.nBarActiva = t;
                Nodos.lienzo.repaint();
                numbar = t;
            }
        });

        ////////////////////////////////////
        this.btnBarraAceptarCambio.addActionListener(new
ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (chCargaMasa.isSelected()) {
                    Nodos.lienzo.superficie.isCarga = true;
                } else {
                    Nodos.lienzo.superficie.isCarga = false;
                }
                int k =
Nodos.lienzo.superficie.barras[numbar].getOrden();

Nodos.lienzo.superficie.barras[k].setDensiFuerza(Double.parseDouble(tB
arraDensFuerzaF.getText()));
```

```
Nodos.lienzo.superficie.barras[k].setOrdenMatrizDensidad(k);
                Nodos.lienzo.superficie.DFF.setElemento(k, k,
Nodos.lienzo.superficie.barras[k].getDensiFuerzaF());
                Nodos.lienzo.superficie.barras[k].setSeccion(0.0001 *
Double.parseDouble(tBarraSeccionEntrada.getText()));
                //void setParametrosMaterial(double den, double lE,
double mE, double coeSeg){

Nodos.lienzo.superficie.barras[k].setParametrosMaterial(Double.parseDo
uble(tBarraDensiMasa.getText()),
Double.parseDouble(tBarraLimiteElastico.getText()),
Double.parseDouble(tBarraModuloElastico.getText()),
Double.parseDouble(tBarraSeguridad.getText()));
                Nodos.lienzo.superficie.barras[k].actualizar();

Nodos.lienzo.superficie.barras[k].setMaterial(Nodos.lienzo.superficie.
dMaterial);
                Paneli.btnEjecutar.doClick();
            }
        });
////////////////////////////////////////////
        this.btnBarraMas.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (numbar < Nodos.lienzo.superficie.barras.length - 1
&& numbar > -1) {
                    numbar++;
                }
                mostrarParametroBarra(numbar);
            }
        });
        ////////////////////////////////////////////
        this.btnBarraMenos.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (numbar > 0 && numbar <
Nodos.lienzo.superficie.barras.length) {
                    numbar--;
                }
                mostrarParametroBarra(numbar);
            }
        });
        ////////////////////////////////////////////
        this.btnBarraInicio.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                mostrarParametroBarra(0);
                numbar = 0;
            }
        });
        ////////////////////////////////////////////////////
        this.btnBarraFinal.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

mostrarParametroBarra(Nodos.lienzo.superficie.barras.length - 1);
                numbar = Nodos.lienzo.superficie.barras.length - 1;
            }
        });
```

```
        // String[] sMaterial={"Acero B500","Acero S235","Acero
S275","Acero S355","Acero Y1760","Acero Y1830","Aluminio","Textil
T502","Textil T1202", "HA 30","HA 40","HA 50","DEFINIDO USUARIO"};
        cbMaterial.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String str = "                                    ";
                cbMaterial = (JComboBox) e.getSource();
//TipoEleFinito);
                str = (String) cbMaterial.getSelectedItem();
                if (str == "Acero B500") {
                    Nodos.lienzo.superficie.dMaterial[0] = 7850.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 500e6;
                    Nodos.lienzo.superficie.dMaterial[2] = 19e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 2.5;
                }
                if (str == "Acero S235") {
                    Nodos.lienzo.superficie.dMaterial[0] = 7850.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 235e6;
                    Nodos.lienzo.superficie.dMaterial[2] = 21e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 2.5;
                };
                if (str == "Acero S275") {
                    Nodos.lienzo.superficie.dMaterial[0] = 7850.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 2750e6;
                    Nodos.lienzo.superficie.dMaterial[2] = 21e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 2.5;
                };
                if (str == "Acero S355") {
                    Nodos.lienzo.superficie.dMaterial[0] = 7850.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 355e6;
                    Nodos.lienzo.superficie.dMaterial[2] = 21e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 2.5;
                };
                if (str == "Acero Y1760") {
                    Nodos.lienzo.superficie.dMaterial[0] = 7850.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 1760e6;
                    Nodos.lienzo.superficie.dMaterial[2] = 21e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 2.5;
                };
                if (str == "Acero Y1830") {
                    Nodos.lienzo.superficie.dMaterial[0] = 7850.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 1830e6;
                    Nodos.lienzo.superficie.dMaterial[2] = 21e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 2.5;
                };
                if (str == "Aluminio") {
                    Nodos.lienzo.superficie.dMaterial[0] = 2700.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 70e6;
                    Nodos.lienzo.superficie.dMaterial[2] = 2.1e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 2.5;
                };
                if (str == "Textil T502") {
                    Nodos.lienzo.superficie.dMaterial[0] = 7850.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 5.0e8;
                    Nodos.lienzo.superficie.dMaterial[2] = 21e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 2.5;
                };
                if (str == "Textil T1202") {
                    Nodos.lienzo.superficie.dMaterial[0] = 7850.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 5.0e8;
```

```
                    Nodos.lienzo.superficie.dMaterial[2] = 21e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 2.5;
                };
                if (str == "HA 30") {
                    Nodos.lienzo.superficie.dMaterial[0] = 2500.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 5.0e8;
                    Nodos.lienzo.superficie.dMaterial[2] = 30e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 1.5;
                };
                if (str == "HA 40") {
                    Nodos.lienzo.superficie.dMaterial[0] = 2500.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 5.0e8;
                    Nodos.lienzo.superficie.dMaterial[2] = 40e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 1.5;
                };
                if (str == "HA 50") {
                    Nodos.lienzo.superficie.dMaterial[0] = 2500.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 5.0e8;
                    Nodos.lienzo.superficie.dMaterial[2] = 50e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 1.5;
                };
                if (str == "DEFINIDO USUARIO") {
                    Nodos.lienzo.superficie.dMaterial[0] = 2500.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 5.0e8;
                    Nodos.lienzo.superficie.dMaterial[2] = 21e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 2.5;
                };

Nodos.lienzo.superficie.barras[numbar].setMaterial(Nodos.lienzo.superf
icie.dMaterial);
            }
        });
    }
////////////////////////////////////////////////////////////////////

    void mostrarParametroBarra(int n) {
        if (chCargaMasa.isSelected()) {
            Nodos.lienzo.superficie.isCarga = true;
        } else {
            Nodos.lienzo.superficie.isCarga = false;
        }
//int t=Nodos.lienzo.superficie.nudo[n].getOrden();
//int t= Nodos.lienzo.superficie.barras[numbar].getOrden();
//int t=n;
        int t = numbar;
        Nodos.lienzo.nBarActiva = t;
        tNumBarra.setText(String.valueOf(t));
        Nodos.lienzo.superficie.barras[t].actualizar();

tNuInicio.setText(String.valueOf(Nodos.lienzo.superficie.barras[t].ori
gen.getOrden()));

tNuFin.setText(String.valueOf(Nodos.lienzo.superficie.barras[t].fin.ge
tOrden()));

chTraccion.setSelected(Nodos.lienzo.superficie.barras[t].traccion);

tBarraEtiqueta.setText(Nodos.lienzo.superficie.barras[t].getEtiqueta()
);
```

```
Nodos.lienzo.superficie.barras[t].setDensiFuerza(Nodos.lienzo.superfic
ie.DFF.getElemento(t, t));

tBarraDensFuerzaF.setText(String.valueOf(Nodos.lienzo.superficie.barra
s[t].getDensiFuerzaF()));

tBarraDensFuerzaC.setText(String.valueOf(Nodos.lienzo.superficie.barra
s[t].densiFuerzaDina));
        //  Nodos.lienzo.nBarActiva=t;Nodos.lienzo.repaint();
        tBarraLongitud.setText(String.valueOf(0.001 * Math.round(1000
* Nodos.lienzo.superficie.barras[t].getLongitud())));
        tBarraFuerzaAxial.setText(String.valueOf(0.01 * Math.round(100
* Nodos.lienzo.superficie.barras[t].fuerzAxial)));
        //
Nodos.lienzo.superficie.barras[t].setParametrosMaterial(dMaterial[0],
dMaterial[1], dMaterial[2], dMaterial[3]);

tBarraDensiMasa.setText(String.valueOf(Nodos.lienzo.superficie.barras[
t].densiMasa));

tBarraModuloElastico.setText(String.valueOf(Nodos.lienzo.superficie.ba
rras[t].modElas));

tBarraLimiteElastico.setText(String.valueOf(Nodos.lienzo.superficie.ba
rras[t].limElas));

tBarraSeguridad.setText(String.valueOf(Nodos.lienzo.superficie.barras[
t].coeficienteSeguridad));
        tBarraSeccionEntrada.setText(String.valueOf(10000 *
Nodos.lienzo.superficie.barras[t].getSeccion()));

tBarraRatio.setText(String.valueOf(Nodos.lienzo.superficie.barras[t].r
atioAgota));

tBarraMasa.setText(String.valueOf(Nodos.lienzo.superficie.barras[t].ge
tMasa()));
        tBarraSeccionC.setText(String.valueOf(10000 *
Nodos.lienzo.superficie.barras[t].seccionC));

tBarraAcortamiento.setText(String.valueOf(Nodos.lienzo.superficie.barr
as[t].acortaInicial));

        Nodos.lienzo.repaint();
    }

    ;
////////////////////////////////////////////////////////////////////
/////////



////////////////////////////////////////////////////////////////////
/////////////////
 void ponNavegaNudos() {   //de nudos
        str = new String();
        // JLabel coordenada = new JLabel(str);
        btnZmas.setBackground(Color.CYAN);
        btnZmenos.setBackground(Color.CYAN);
        JPanel panelZ = new JPanel();
        panelZ.setBackground(Color.LIGHT_GRAY);
```

```
        panelZ.add(btnZmenos, BorderLayout.WEST);
        panelZ.add(btnZmas, BorderLayout.EAST);
        this.setLayout(new BorderLayout());
        this.add(btnYmas, BorderLayout.NORTH);
        this.add(btnYmenos, BorderLayout.SOUTH);
        this.add(btnXmas, BorderLayout.EAST);
        this.add(panelZ, BorderLayout.CENTER);
        this.add(btnXmenos, BorderLayout.WEST);
////////////////////////////////////////////////////////////
        this.tSelecNudo.addKeyListener(new java.awt.event.KeyAdapter()
{
            public void keyReleased(java.awt.event.KeyEvent evt) {
                num =
Nodos.lienzo.superficie.nudo[Integer.valueOf(tSelecNudo.getText().trim
())].getOrden();
                lCoordenadas.setText("Coor= " + darCoordenada() + "
Cargas=" + darCargas());

tEtiqueta.setText(Nodos.lienzo.superficie.nudo[num].getEtiqueta());

tCargaX.setText(String.valueOf(Nodos.lienzo.superficie.nudo[num].carga
.getCompoX()));

tCargaY.setText(String.valueOf(Nodos.lienzo.superficie.nudo[num].carga
.getCompoY()));

tCargaZ.setText(String.valueOf(Nodos.lienzo.superficie.nudo[num].carga
.getCompoZ()));

tCoorX.setText(String.valueOf(Nodos.lienzo.superficie.nudo[num].getX()
));

tCoorY.setText(String.valueOf(Nodos.lienzo.superficie.nudo[num].getY()
));

tCoorZ.setText(String.valueOf(Nodos.lienzo.superficie.nudo[num].getZ()
));
            }
        });
        this.tEtiqueta.addKeyListener(new java.awt.event.KeyAdapter()
{
            public void keyReleased(java.awt.event.KeyEvent evt) {
                etiq = tEtiqueta.getText();
            }
        });
        /**
         * ******************************Confirmar
         * datos*****************************
         */
        this.btnOkNavegaNudo.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (chCargaMasa.isSelected()) {
                    Nodos.lienzo.superficie.isCarga = true;
                } else {
                    Nodos.lienzo.superficie.isCarga = false;
                }
                char ap = 'O';
                char ex = 'I';
```

```
                num =
Nodos.lienzo.superficie.nudo[Integer.valueOf(tSelecNudo.getText().trim
())].getOrden();

Nodos.lienzo.superficie.nudo[num].setX(Double.parseDouble(tCoorX.getTe
xt().trim()));

Nodos.lienzo.superficie.nudo[num].setY(Double.parseDouble(tCoorY.getTe
xt().trim()));

Nodos.lienzo.superficie.nudo[num].setZ(Double.parseDouble(tCoorZ.getTe
xt().trim()));
                etiq = tEtiqueta.getText();
                Nodos.lienzo.superficie.nudo[num].setEtiqueta(etiq);
                lCoordenadas.setText("Coor= " + darCoordenada() + "
Cargas=" + darCargas());
                Nodos.lienzo.repaint();
                // if (chGenerico.isSelected())
Nodos.lienzo.superficie.nudo[num].setTipoNudo('g');
                if (chApoyo.isSelected()) {
                    Nodos.lienzo.superficie.nudo[num].setApoyo(true);
                    ap = 'A';
                    if (chCargaMasa.isSelected()) {
                        Nodos.lienzo.superficie.isCarga = true;
                    } else {
                        Nodos.lienzo.superficie.isCarga = false;
                    }

Nodos.lienzo.superficie.setNumApoyos(Nodos.lienzo.superficie.getNumApo
yos() + 1);
                    //   Nodos.lienzo.pixelApo=new
Point[Nodos.lienzo.superficie.getNumApoyos()];
                    //     for(int
p=0;p<Nodos.lienzo.superficie.getNumApoyos();p++){
Nodos.lienzo.pixelApo[p]=new Point();  }
                } else {
                    Nodos.lienzo.superficie.nudo[num].setApoyo(false);
                    ap = 'O';
                }
                // if
(chExterno.isSelected()){Nodos.lienzo.superficie.nudo[num].setExterno(
true);ex='X';}else{Nodos.lienzo.superficie.nudo[num].setExterno(false)
;ex='I';}
                //
                etiq = "   ";
                char va = 'O';
                if (Nodos.lienzo.superficie.nudo[num].apoyo) {
                    va = 'A';
                } else {
                    va = 'O';
                }
                ex = 'I';
                if (Nodos.lienzo.superficie.nudo[num].externo) {
                    ex = 'X';
                } else {
                    ex = 'I';
                }
                int pos =
Nodos.lienzo.superficie.nudo[num].getOrden();
                String sOrden = String.format("%04d", pos);
```

```
                etiq = "N" + String.valueOf(va) + sOrden +
String.valueOf(ex) +
String.valueOf(Nodos.lienzo.superficie.nudo[pos].getTipoNudo()) +
String.valueOf(Nodos.lienzo.superficie.nudo[pos].getTipoEleFinito());
                Nodos.lienzo.superficie.nudo[num].setEtiqueta(etiq);

tEtiqueta.setText(Nodos.lienzo.superficie.nudo[num].getEtiqueta());

Nodos.lienzo.superficie.nudo[num].carga.setComponentes(Double.valueOf(
tCargaX.getText().trim()).doubleValue(),
Double.valueOf(tCargaY.getText().trim()).doubleValue(),
Double.valueOf(tCargaZ.getText().trim()).doubleValue());
                if (Nodos.lienzo.superficie.numNuNoExt == 0) {

Nodos.lienzo.superficie.setGeometria(Nodos.lienzo.superficie.tipoEleFi
nito);
                }
            }
        });

//////////////////////////////////////////CONTROL DE BOTONES DE
NAVEGACION ////////////777
        this.btnXmas.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

Nodos.lienzo.superficie.nudo[num].setX(Nodos.lienzo.superficie.nudo[nu
m].getX() + 1);
                lCoordenadas.setText("Coor= " + darCoordenada() + "
Cargas=" + darCargas());
                Nodos.lienzo.repaint();
            }
        });
        this.btnXmenos.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

Nodos.lienzo.superficie.nudo[num].setX(Nodos.lienzo.superficie.nudo[nu
m].getX() - 1);
                lCoordenadas.setText("Coor= " + darCoordenada() + "
Cargas=" + darCargas());
                Nodos.lienzo.repaint();
            }
        });
        this.btnYmas.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

Nodos.lienzo.superficie.nudo[num].setY(Nodos.lienzo.superficie.nudo[nu
m].getY() + 1);
                lCoordenadas.setText("Coor= " + darCoordenada() + "
Cargas=" + darCargas());
                Nodos.lienzo.repaint();
            }
        });
        this.btnYmenos.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

Nodos.lienzo.superficie.nudo[num].setY(Nodos.lienzo.superficie.nudo[nu
m].getY() - 1);
```

```
                lCoordenadas.setText("Coor= " + darCoordenada() + "
Cargas=" + darCargas());
                Nodos.lienzo.repaint();
            }
        });
        this.btnZmas.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

Nodos.lienzo.superficie.nudo[num].setZ(Nodos.lienzo.superficie.nudo[nu
m].getZ() + 1);
                lCoordenadas.setText("Coor= " + darCoordenada() + "
Cargas=" + darCargas());
                Nodos.lienzo.repaint();
            }
        });
        this.btnZmenos.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

Nodos.lienzo.superficie.nudo[num].setZ(Nodos.lienzo.superficie.nudo[nu
m].getZ() - 1);
                lCoordenadas.setText("Coor= " + darCoordenada() + "
Cargas=" + darCargas());
                Nodos.lienzo.repaint();

            }
        });

    }
////////////////////////////////////////////////////////////////////
///////

    void verNudos(Nudo[] nu) {
        if (chCargaMasa.isSelected()) {
            Nodos.lienzo.superficie.isCarga = true;
        } else {
            Nodos.lienzo.superficie.isCarga = false;
        }
        int j = 0;
        String linea = " nodo num\t" + " orden \t\t" + "etiquta\t" +
"Coordenadas \t\t\t\t" + "EleFinito\t" + "Nudo\t" + " VÃnculos\n";
        areaTexto.append(linea);
        String StringVinculos = "";

        for (int n = 0; n < nu.length; n++) {
            StringVinculos = "";
//for(j=0;j<nu[n].getNumBarrasT();j++) {
            for (j = 0; j < nu[n].vinculo.length; j++) {
                if (nu[n].vinculo[j] != null) {
                    StringVinculos = StringVinculos + "," +
nu[n].vinculo[j].getEtiqueta();
                } else {
                    StringVinculos = " n.a.";
                }
            }
            if (nu[n].tipoNudo == 'X') {
                linea = linea = "Elemento(" + String.valueOf(n) +
")\t" + nu[n].getOrden() + "\t" + " indice=" + nu[n].getIndice() +
"\t" + nu[n].getEtiqueta() + "\t\t" + nu[n].toPrint() + "\t" +
StringVinculos;
```

```
            } else {
                linea = "Elemento(" + String.valueOf(n) + ")\t" +
nu[n].getOrden() + "\t" + " indice=" + nu[n].getIndice() + "\t" +
nu[n].getEtiqueta() + "\t\t" + nu[n].toPrint2d() + "\t\t" +
nu[n].tipoEleFinito + "\t"
                        + nu[n].tipoNudo + "\t Vinculos S,E,T(" +
String.valueOf(nu[n].getNumBarrasS()) + "," +
String.valueOf(nu[n].getNumBarrasE()) + "," +
String.valueOf(nu[n].getNumBarrasT()) + ")\t" + StringVinculos;
                areaTexto.append("\n" + linea);
            }
        }
    }
//////////////////////////////VER
BARRAS//////////////////////////////////////

    void verBarras(Barra[] bar) {
        this.areaTexto.append("\n Listado de barras\n\n");
        this.areaTexto.append("NÃºmero de barras: " +
String.valueOf(Nodos.lienzo.superficie.getNumBarras()) + "\n");
        for (int j = 0; j < bar.length; j++) {
            this.areaTexto.append(String.valueOf(j) + "Orden=" +
String.valueOf(bar[j].getOrden()) + "\t" + bar[j].toPrint() + "\t
S(cm2)=" + String.valueOf(bar[j].getSeccion() + "\t D(" + j + ")" +
"DenFuerzas=" + String.valueOf(bar[j].getDensiFuerzaF()) + "\n"));
        }
    }

    void verEF(EleFinito[] ef) {
        this.areaTexto.append("\n Listado de Elementos Finitos\n\n");
        this.areaTexto.append("NÃºmero de EF: " +
String.valueOf(Nodos.lienzo.superficie.eleFinito.length) + "\n");
        for (int j = 0; j < ef.length; j++) {
            this.areaTexto.append(String.valueOf(j) + "Orden=" +
String.valueOf(ef[j].getOrden()) + "\t" +
String.valueOf(ef[j].getEtiqueta()) + "\t S(cm2)=" +
String.valueOf(ef[j].getArea() + "\t D(" + j + ")" + "Densidad=" +
String.valueOf(ef[j].getDensiMasa()) + "\n"));
        }
    }

    /**
     *
*****************************************************************
     */
    static String darCoordenada() {//Math.rint(numero*100)/100
        return "(" +
String.valueOf(Math.rint(Nodos.lienzo.superficie.nudo[num].getX() *
100) / 100) + "," +
String.valueOf(Math.rint(Nodos.lienzo.superficie.nudo[num].getY() *
100) / 100) + "," +
String.valueOf(Math.rint(Nodos.lienzo.superficie.nudo[num].getZ() *
100) / 100) + ")";
    }

    static String darCargas() { //Math.rint(numero*100)/100
        return "   (" +
String.valueOf(Math.rint(Nodos.lienzo.superficie.nudo[num].carga.getCo
mpoX() * 100) / 100) + "," +
String.valueOf(Math.rint(Nodos.lienzo.superficie.nudo[num].carga.getCo
mpoY() * 100) / 100) + "," +
```

```
String.valueOf(Math.rint(Nodos.lienzo.superficie.nudo[num].carga.getCo
mpoZ() * 100) / 100) + ")";
    }

    ////////////////////////////////////////////////////Funciones
con archivos y de volcado de
memoria/////////////////////////////////////////////////////////
    public void guardarArchivo(String titulo, String Directorio,
String tipoArchivo) {
        ObjectOutputStream bos = null;
        dialogo.setTitle(titulo);
        dialogo.setFile(tipoArchivo);
        dialogo.setLocation(50, 50);
        dialogo.setVisible(true);
        try {
            this.dialogo.getDirectory();
            bos = new ObjectOutputStream(new FileOutputStream((new
File(dialogo.getDirectory(), dialogo.getFile()))));
            System.out.println("guardando el archivo");
            bos.writeUnshared(Nodos.lienzo.superficie);
            bos.flush();
            // bos.writeObject(Nodos.lienzo); bos.flush();
            JOptionPane.showMessageDialog(null, "Archivo " +
dialogo.getFile() + " guardado"); // donde pone null podemos poner
cualquier componente desde donde se hace la llamada
            System.out.println("Archivo guardado");
            bos.close();
        } catch (FileNotFoundException fnfe) {
            System.out.println("No se encuentra el archivo " + fnfe);
        } catch (IOException ioe) {
            System.out.println("Error en la escritura del archivo " +
ioe);
        }
    }

    ;
////////////////////////////////////////////////////////////////////
//
public Superficie leerArchivo(String titulo, String Directorio, String
tipoArchivo) {
        Superficie memo = new Superficie();

        dialogo.setTitle(titulo);
        dialogo.setFile(tipoArchivo);
        dialogo.setDirectory(Directorio);
        dialogo.setLocation(50, 50);
        dialogo.setVisible(true);
        // String strFileName=dialogo.getFile();
        ObjectInputStream bis = null;
        try {
            dialogo.setDirectory(this.dialogo.getDirectory());
            bis = new ObjectInputStream(new FileInputStream((new
File(dialogo.getDirectory(), dialogo.getFile()))));
            System.out.println("Leyendo el archivo, " +
dialogo.getFile());
            memo = (Superficie) bis.readUnshared();
            JOptionPane.showMessageDialog(null, "Archivo leido");
            System.out.println("Archivo leÃdo");
            bis.close();
        } catch (Exception ec) {
            // ec.printStackTrace();
```

```
            JOptionPane.showMessageDialog(null, "Error en la lectura"
+ ec);
            System.out.println("Error en la lectura " + ec);
        }
        return memo;
    }
//////////////////////////COMPROBACION////////////////////////////////
//////////////////////////

    void test() {
        //comprobamos todos los Ãndices importantes como el numero de
apoyos y
        int NN = Nodos.lienzo.superficie.nudo.length;
        int numApoyos = 0;
        for (int n = 0; n < NN; n++) {
            if (Nodos.lienzo.superficie.nuOrdenado[n].getApoyo()) {
                numApoyos = numApoyos + 1;
            }
        }
        Nodos.lienzo.superficie.setNumApoyos(numApoyos);
        int nApoyos = numApoyos;
        int NRN = NN - nApoyos;
// 1Âº matriz de autovectores orlada
        Nodos.lienzo.superficie.MF = new Matriz(NN, NRN);
//lleno de ceros las filas aÃ±aÃ±didas a la matriz orlada
//llenamos todos los elementos como ceros
        for (int i = 0; i < NN; i++) {
            for (int j = 0; j < NRN; j++) {
                Nodos.lienzo.superficie.MF.setElemento(i, j, 0.0);
            }
        }
        for (int i = 0; i < NRN; i++) {
            for (int j = 0; j < NRN; j++) {
                Nodos.lienzo.superficie.MF.setElemento(i + nApoyos, j,
Nodos.lienzo.superficie.VP.getElemento(i, j));
            }
        }
        if (chMostrar.isSelected()) {
            areaTexto.append("\n   Matriz MF  de Autovectores orlada
\n  ");
        }
        if (chMostrar.isSelected()) {

areaTexto.append(Nodos.lienzo.superficie.MF.salidaMatriz());
        }
// 2Âº matriz de alargamiento
        int numBarras = Nodos.lienzo.superficie.barras.length;
        Nodos.lienzo.superficie.AL = new
Matriz(Nodos.lienzo.superficie.G.numFilas, NRN);//AL(nBarras* nudos
reducidos)
        Nodos.lienzo.superficie.AL =
Nodos.lienzo.superficie.AL.producto(Nodos.lienzo.superficie.G,
Nodos.lienzo.superficie.MF);
        if (chMostrar.isSelected()) {
            areaTexto.append("\n   Matriz AL de Alargamiento \n ");
        }
        if (chMostrar.isSelected()) {

areaTexto.append(Nodos.lienzo.superficie.AL.salidaMatriz());
        }
```

```
//  3Âº/////////////////crear la matriz de energÃa de alargamiento
EAL
        Nodos.lienzo.superficie.EAL =
Nodos.lienzo.superficie.AL.clone();
        double valor, cuadrado;
        for (int ff = 0; ff < numBarras; ff++) {
            for (int cc = 0; cc < NRN; cc++) {
                valor = Nodos.lienzo.superficie.AL.getElemento(ff,
cc);
                cuadrado = Math.pow(valor, 2);
                valor = cuadrado / 2;
                Nodos.lienzo.superficie.EAL.setElemento(ff, cc,
valor);
            }
        }
        if (chMostrar.isSelected()) {
            areaTexto.append("\n   Matriz EAL  de ENERGIA por
alargamiento\n  ");
        }
        if (chMostrar.isSelected()) {

areaTexto.append(Nodos.lienzo.superficie.EAL.salidaMatriz());
        }
// 4 Crear la matriz de energÃa potencial EP
        Matriz DDtemp = Nodos.lienzo.superficie.DD.clone();
        Matriz EALtemp = Nodos.lienzo.superficie.EAL.clone();
        Nodos.lienzo.superficie.EP = new Matriz(numBarras, NRN);
        Nodos.lienzo.superficie.EP =
Nodos.lienzo.superficie.EP.producto(DDtemp, EALtemp);
        if (chMostrar.isSelected()) {
            areaTexto.append("\n   Matriz EP, de ENERGIA POTENCIAL
\n");
        }
        if (chMostrar.isSelected()) {

areaTexto.append(Nodos.lienzo.superficie.EP.salidaMatriz());
        }
//creo el array de de eenrgia poencia elastica
//cada elemento es la suma de todos los elementos de cada columna de
EP
        Nodos.lienzo.superficie.EPE = new double[NRN];
        for (int c = 0; c < NRN; c++) {
            double suma = 0;
            for (int f = 0; f < numBarras; f++) {
                suma = suma +
Nodos.lienzo.superficie.EP.getElemento(f, c);
            }
            Nodos.lienzo.superficie.EPE[c] = suma;
        }
        areaTexto.append("\n EPE:  ENERGIA POTENCIAL ELASTICA DE CADA
FORMA DE OSCILACION \n");
        for (int c = 0; c < NRN; c++) {
            areaTexto.append("EPE (" + String.valueOf(c) + ")= " +
String.valueOf(Nodos.lienzo.superficie.EPE[c]) + "\n");
        }

areaTexto.append("_____________________________________________________
_____________________\n");
        areaTexto.append("\n EPE:  MATRICES CINETICAS  \n");
        /*
```

```
         E) Volviendo a la matriz de AUTOVECTORES (NRN por NRN)
generamos la matriz "EC" (NRN por NRN) tal que cada valor es el propio
valor de la matriz de autovectores al cuadrado por 0.5.
         F) Multiplicamos la matriz de masas reducidas factorizada
"MRF" por "EC" y obtenemos la matriz de energÃa cinÃ©tica "CIN" (NRN
por NRN).
         G) Obtenemos el array de energÃa cinÃ©tica "ACIN" (NRN)
resultante de sumar las columnas de la matriz "CIN"
         */
        //  3Âº/////////////////crear la matriz de energÃa de
alargamiento EAL
        Nodos.lienzo.superficie.EC = new Matriz(NRN, NRN);
        double v, cuadro;
        for (int ff = 0; ff < NRN; ff++) {
            for (int cc = 0; cc < NRN; cc++) {
                v = Nodos.lienzo.superficie.VP.getElemento(ff, cc);
                cuadro = Math.pow(v, 2);
                v = cuadro / 2;
                Nodos.lienzo.superficie.EC.setElemento(ff, cc, v);
            }
        }
        if (chMostrar.isSelected()) {
            areaTexto.append("\n   Matriz EC   ");
        }
        if (chMostrar.isSelected()) {

areaTexto.append(Nodos.lienzo.superficie.EC.salidaMatriz());
        }
//4Âº Creamos una matriz llamada CIN
        Nodos.lienzo.superficie.CIN = new Matriz(NRN, NRN);
        Nodos.lienzo.superficie.CIN =
Nodos.lienzo.superficie.CIN.producto(Nodos.lienzo.superficie.MasaR,
Nodos.lienzo.superficie.EC);
        if (chMostrar.isSelected()) {
            areaTexto.append("\n   Matriz CIN    ");
        }
        if (chMostrar.isSelected()) {

areaTexto.append(Nodos.lienzo.superficie.CIN.salidaMatriz());
        }
// 5Âº CReo el array ACIN
        Nodos.lienzo.superficie.ACIN = new double[NRN];
        for (int c = 0; c < NRN; c++) {
            double suma = 0;
            for (int f = 0; f < NRN; f++) {
                suma = suma +
Nodos.lienzo.superficie.CIN.getElemento(f, c);
            }
            Nodos.lienzo.superficie.ACIN[c] = suma;
        }
        areaTexto.append("\n EPE: ARRAY DE VALOREA ACIN \n");
        for (int c = 0; c < NRN; c++) {
            areaTexto.append("ACIN (" + String.valueOf(c) + ")= " +
String.valueOf(Nodos.lienzo.superficie.ACIN[c]) + "\n");
        }
        double error, vAngular, periodo, frecuencia;
        String lema;
        areaTexto.append("\nPorcentaje de ERROR para los modos de
vibraciÃ³n, velocidad angular, periodo y frecuencia\n");
        for (int s = 0; s < NRN; s++) {
```

```
            vAngular =
Math.sqrt(Nodos.lienzo.superficie.valoresPropios[s]);
            periodo = Math.PI * 2 / vAngular;
            frecuencia = 1 / periodo;
            error = 100 * (Nodos.lienzo.superficie.EPE[s] /
Nodos.lienzo.superficie.ACIN[s] -
Nodos.lienzo.superficie.valoresPropios[s]) /
Nodos.lienzo.superficie.valoresPropios[s];
            if (Math.abs(error) < 5) {
                lema = " comprobaciÃ³n OK ";
            } else {
                lema = "error superior al 5%";
            }
            areaTexto.append("Nivel de error para modo vibraciÃ³n(" +
s + ")=" + error + " % " + lema + " . Velocida angular=" + vAngular +
" s(-1).  Periodo=" + periodo + " s.  Frecuencia =" + frecuencia + "
Hz  \n");

        }

    }

////////////////////////////////////////////////////////////////////
/////////////
    void calculoDinamico() {
        if (chCargaMasa.isSelected()) {
            Nodos.lienzo.superficie.isCarga = true;
        } else {
            Nodos.lienzo.superficie.isCarga = false;
        }
        Nodos.lienzo.superficie.FactorMasa =
Double.valueOf(tFactorMasa.getText().trim()).doubleValue();
        int numApoyos = 0; //calculamos de nuevo los apoyos
        int numNuTotal = Nodos.lienzo.superficie.nuOrdenado.length;

        for (int n = 0; n < numNuTotal; n++) {
            if (Nodos.lienzo.superficie.nuOrdenado[n].getApoyo()) {
                numApoyos = numApoyos + 1;
            }
        }
        Nodos.lienzo.superficie.setNumApoyos(numApoyos);

        //creamos la matriz de densidades calculadas
        Nodos.lienzo.superficie.DD = new
Matriz(Nodos.lienzo.superficie.barras.length,
Nodos.lienzo.superficie.barras.length);
        for (int i = 0; i < Nodos.lienzo.superficie.barras.length;
i++) {
            Nodos.lienzo.superficie.DD.setElemento(i, i,
Nodos.lienzo.superficie.barras[i].getDensiFuerzaC());
        }
        if (chMostrar.isSelected()) {
            areaTexto.append("Matriz de Densidad de Fuerzas
DinÃ¡micas, DD");
        }
        if (chMostrar.isSelected()) {

areaTexto.append(Nodos.lienzo.superficie.DD.salidaMatrizInt());
        }
```

```
        Nodos.lienzo.superficie.MDE = new Matriz(numNuTotal,
numNuTotal);  //
        Nodos.lienzo.superficie.tempo = new
Matriz(Nodos.lienzo.superficie.transG.numFilas,
Nodos.lienzo.superficie.DD.numColumnas); //inicialmente ernesto lo
puso al revÃ©s
        Nodos.lienzo.superficie.MDE =
Nodos.lienzo.superficie.MDE.producto(Nodos.lienzo.superficie.tempo.pro
ducto(Nodos.lienzo.superficie.transG,
Nodos.lienzo.superficie.DD.clone()), Nodos.lienzo.superficie.G);
        if (chMostrar.isSelected()) {
            areaTexto.append("Matriz MDE= transG x DD x G");
        }
        if (chMostrar.isSelected()) {

areaTexto.append(Nodos.lienzo.superficie.MDE.salidaMatriz());
        }
//Creo y lleno  MasaR = matriz de masas reducida, sin los puntos de
apoyo. Es cuadrada.
        int nFyC = Nodos.lienzo.superficie.Masa.numFilas - numApoyos;
        Nodos.lienzo.superficie.MasaR = new Matriz(nFyC, nFyC);
        for (int f = numApoyos; f < numNuTotal; f++) {
            Nodos.lienzo.superficie.MasaR.setElemento(f - numApoyos, f
- numApoyos, Nodos.lienzo.superficie.FactorMasa *
Nodos.lienzo.superficie.Masa.getElemento(f, f));
        }
        if (chMostrar.isSelected()) {
            areaTexto.append("\n\n Se reducirÃ¡n las matrices un
nÃºmero de Apoyos " + numApoyos + "\n");
        }
        if (chMostrar.isSelected()) {
            areaTexto.append("Matriz de Masas Reducidas MasaR x el
factor de masas " + Nodos.lienzo.superficie.FactorMasa + "\n");
        }
        if (chMostrar.isSelected()) {

areaTexto.append(Nodos.lienzo.superficie.MasaR.salidaMatriz());
        }
//Creo y lleno MDER que es el resultado de quitarle los apoyos a MDE;
Es cuadrada
        nFyC = numNuTotal - Nodos.lienzo.superficie.numApoyos;
        Nodos.lienzo.superficie.MDER = new Matriz(nFyC, nFyC);
        for (int f = numApoyos; f < numNuTotal; f++) {
            for (int c = numApoyos; c < numNuTotal; c++) {
                Nodos.lienzo.superficie.MDER.setElemento(f -
numApoyos, c - numApoyos, Nodos.lienzo.superficie.MDE.getElemento(f,
c));
            }
        }
        if (chMostrar.isSelected()) {
            areaTexto.append("Matriz de MDER ");
        }
        if (chMostrar.isSelected()) {

areaTexto.append(Nodos.lienzo.superficie.MDER.salidaMatriz());
        }
// Calculamos la inversa de la Matriz de MasaR
        nFyC = Nodos.lienzo.superficie.MasaR.numFilas; //es cuadrada
        Nodos.lienzo.superficie.MasaRI = new Matriz(nFyC, nFyC);
        Nodos.lienzo.superficie.MasaRI =
Nodos.lienzo.superficie.MasaRI.inversa(Nodos.lienzo.superficie.MasaR);
```

```
        if (chMostrar.isSelected()) {
            areaTexto.append("Matriz inversa de Masas Reducidas ");
        }
        if (chMostrar.isSelected()) {

areaTexto.append(Nodos.lienzo.superficie.MasaRI.salidaMatriz());
        }
        Nodos.lienzo.superficie.MA = new Matriz(nFyC, nFyC);
        Nodos.lienzo.superficie.MA =
Nodos.lienzo.superficie.MA.producto(Nodos.lienzo.superficie.MasaRI,
Nodos.lienzo.superficie.MDER);
        if (chMostrar.isSelected()) {

areaTexto.append("_____________________________________________\n");
        }
        if (chMostrar.isSelected()) {
            areaTexto.append("   Matriz MA para los autovalores");
        }
        if (chMostrar.isSelected()) {

areaTexto.append(Nodos.lienzo.superficie.MA.salidaMatriz());
        }
        Nodos.lienzo.superficie.valoresPropios = new
double[Nodos.lienzo.superficie.MA.numFilas];
//Nodos.lienzo.superficie.valoresPropios=new
double[Nodos.lienzo.superficie.MESiste.numFilas];
        for (int v = 0; v <
Nodos.lienzo.superficie.valoresPropios.length; v++) {
            Nodos.lienzo.superficie.valoresPropios[v] = 1.0;
        }
//Nodos.lienzo.superficie.VP=new
Matriz(Nodos.lienzo.superficie.MA.numFilas,Nodos.lienzo.superficie.MA.
numFilas);
//Nodos.lienzo.superficie.VP= new
Matriz(Nodos.lienzo.superficie.MA.numFilas,Nodos.lienzo.superficie.MA.
numColumnas );
//for(int f=0;f<Nodos.lienzo.superficie.VP.numFilas;f++){for(int
c=0;c<Nodos.lienzo.superficie.VP.numColumnas;c++){
//Nodos.lienzo.superficie.VP.x[f][c]=1.0;
//}}
        Nodos.lienzo.superficie.VP =
Nodos.lienzo.superficie.MA.clone();
        try {
            Nodos.lienzo.superficie.VP =
Nodos.lienzo.superficie.MA.valoresPropios(Nodos.lienzo.superficie.valo
resPropios, Nodos.lienzo.superficie.interacciones);
            areaTexto.append("===============Matriz de valores
AUTOVECTORES ======================== ");
            areaTexto.append("  " +
Nodos.lienzo.superficie.VP.salidaMatriz());
            areaTexto.append("==============Valores propios\n");
            for (int v = 0; v <
Nodos.lienzo.superficie.valoresPropios.length; v++) {
                areaTexto.append("Auto valor" + String.valueOf(v) +
"=" + String.valueOf(Nodos.lienzo.superficie.valoresPropios[v]) +
"\n");
            }
            btnComprueba.setEnabled(true);
        } catch (ValoresExcepcion exc) {
```

```
            // JOptionPane.showMessageDialog(  null, "Error en el
cÃ¡lculo de la matriz de Densidades de Fuerza. Se asignara una
densidad por defecto para todas las barras");
            JOptionPane.showMessageDialog(null, exc);
        }
    }
////////////////////////////////////////////////////////////////////
////////

////////////////////////////////////////////////////////////////////
//////////////
    void calculoFormas() {
        int numNuTotal = Nodos.lienzo.superficie.nudo.length;
        double numBa = 0;
        int numApoyos = 0;
        Nodos.lienzo.superficie.isCarga = chCargaMasa.isSelected();
        if (chAlerta.isSelected()) {
            Nodos.bAlerta = true;
        } else {
            Nodos.bAlerta = false;
        }
        //if(chMostrar.isSelected())Nodos.bMostrar=true;else
Nodos.bMostrar=false;
        Nodos.lienzo.superficie.nuOrdenado = new
Nudo[Nodos.lienzo.superficie.nudo.length];
        Nodos.lienzo.superficie.nuOrdenado = (Nudo[])
Nodos.lienzo.superficie.nudo.clone();
        Nodos.lienzo.superficie.nuSalida = (Nudo[])
Nodos.lienzo.superficie.nudo.clone();
        Nodos.lienzo.superficie.nuOrdenado =
Nodos.lienzo.superficie.OrdenarNudos(Nodos.lienzo.superficie.nuOrdenad
o); //}   //este corchete es el resultado del if. Ahora siempre
ordeno.
//calculos los parametros de numNudos, numApoyos, numBarras;

        for (int n = 0; n < numNuTotal; n++) {
            if (Nodos.lienzo.superficie.nuOrdenado[n].getApoyo()) {
                numApoyos = numApoyos + 1;
            }
        }
        Nodos.lienzo.superficie.setNumApoyos(numApoyos);

//for(int n=0; n<numNuTotal;n++){
numBa=numBa+Nodos.lienzo.superficie.nudo[n].getNumBarrasT(); }
        for (int n = 0; n < numNuTotal; n++) {
            numBa = numBa +
Nodos.lienzo.superficie.nuOrdenado[n].vinculo.length;
        }
        int numBarras = (int) (numBa / 2);
        Nodos.lienzo.superficie.setNumBarras(numBarras);

        if (chMostrar.isSelected()) {
            areaTexto.append("\n NÃºmero total de barras de la
superficie = (" + String.valueOf(numBarras) + ") " +
String.valueOf(Nodos.lienzo.superficie.getNumBarras()) + "\n" +
"NÃºmero de nudos=" +
String.valueOf(Nodos.lienzo.superficie.nudo.length) + "(" +
String.valueOf(numNuTotal) + "), Apoyos=" +
String.valueOf(Nodos.lienzo.superficie.getNumApoyos()) + " \n");
        }
```

```
        /**
         * **************MAtriz Or original o desordenada***********
         */
        // 1Âº Se crea la matirz
        Nodos.lienzo.superficie.Origen = new
Matriz(Nodos.lienzo.superficie.getNumBarras(), numNuTotal); //se puede
crear la matriz pues ya se conocen las barras totales de la
superficie.
//2Âº se le pone todos los valores a cero
        for (int i = 0; i < Nodos.lienzo.superficie.getNumBarras();
i++) {
            for (int j = 0; j < Nodos.lienzo.superficie.numNudos; j++)
{
                Nodos.lienzo.superficie.Origen.x[i][j] = 0;
            }
        }//iniciamos todos los elementos=0;/
this.numBarras=this.getNumeroBarras();
//3Âº le asigno el valor de sus elementos (-1,0,1) a partir del array
de nudos original.
        Nodos.lienzo.superficie.Origen =
Nodos.lienzo.superficie.asignaMatrizVinculoOrigen(Nodos.lienzo.superfi
cie.Origen, Nodos.lienzo.superficie.nudo);
        if (chMostrar.isSelected()) {
            areaTexto.append("Matriz Original sin orden  Origen \n");
        }
        if (chMostrar.isSelected()) {

areaTexto.append(Nodos.lienzo.superficie.Origen.salidaMatrizInt());
//solo se imprimen enteros pero la matriz es de dobles
        }
        /**
         * ********MATRIZ ORDENADA G********************
         */
//1Âº Se crea la matriz que tiene que tener las mismas dimensioes que
la OR
        Matriz veriOrigen = new Matriz(1, numNuTotal);
        //creo la matriz de verificaciÃ³n de la matriz inicial
        veriOrigen =
veriOrigen.verificaMatriz(Nodos.lienzo.superficie.Origen);
        if (chMostrar.isSelected()) {
            areaTexto.append(veriOrigen.salidaMatrizInt());
        }
        //se crea la matriz G que va a ser ordanada. La funciÃ³n de
creaciÃ³n le asigna a todos los valores ceros.
        Nodos.lienzo.superficie.G = new
Matriz(Nodos.lienzo.superficie.Origen.numFilas,
Nodos.lienzo.superficie.Origen.numColumnas); //El texto este es
equivalente al anterior
//3Âº se le asigna a sus elementos los valores segÃºn el array  de
nudos ordenados,
        Nodos.lienzo.superficie.G =
Nodos.lienzo.superficie.asignaMatrizVinculoG(Nodos.lienzo.superficie.G
, Nodos.lienzo.superficie.nuOrdenado);
//Creamos el vector de verificaciÃ³n,
        Matriz veriG = new Matriz(1, numNuTotal);
        veriG = veriG.verificaMatriz(Nodos.lienzo.superficie.G);
        if (chMostrar.isSelected()) {
            areaTexto.append("        Matriz DE vÃncuos ordenada de
G\n");
        }
        if (chMostrar.isSelected()) {
```

```
            areaTexto.append("   " +
Nodos.lienzo.superficie.G.salidaMatrizInt()); //solo se imprimen
enteros pero la matriz es de dobles
        }
        if (chMostrar.isSelected()) {
            areaTexto.append("   " + veriG.salidaMatrizInt());
        }
//4Âº creo la matriz traspuesta de G
        Nodos.lienzo.superficie.transG = new Matriz(numNuTotal,
Nodos.lienzo.superficie.getNumBarras());
        Nodos.lienzo.superficie.transG =
Nodos.lienzo.superficie.transG.traspuesta(Nodos.lienzo.superficie.G);
//traspuesta de la matriz con nudos ordenados
        if (chMostrar.isSelected()) {
            areaTexto.append("    Matriz transpuesta de G de Nodos y
vÃ-
nculos\n.....................................................\
n");
        }
        if (chMostrar.isSelected()) {
            areaTexto.append("   " +
Nodos.lienzo.superficie.transG.salidaMatrizInt()); //solo se imprimen
enteros pero la matriz es de dobles
        }//5Âº Se crea la matriz de densidades D y se le asigna
valores a la diagonal, caso de densidad constante.
////////////////////////////////////////////////////////
        this.btnCalBarras.doClick(); //la puse aqui originalmente esta
mÃ¡s adelante
        if (Nodos.lienzo.superficie.DFF == null ||
Nodos.lienzo.superficie.DFF.getNumFilas() <
Nodos.lienzo.superficie.getNumBarras()) {
            Nodos.lienzo.superficie.DFF = new
Matriz(Nodos.lienzo.superficie.getNumBarras(),
Nodos.lienzo.superficie.getNumBarras());
            for (int i = 0; i <
Nodos.lienzo.superficie.getNumBarras(); i++) {
                Nodos.lienzo.superficie.DFF.setElemento(i, i,
Nodos.lienzo.superficie.densiFuerzaBarraForma);
            }
        } else {
            for (int t = 0; t <
Nodos.lienzo.superficie.DFF.getNumFilas(); t++) {
                Nodos.lienzo.superficie.DFF.setElemento(t, t,
Nodos.lienzo.superficie.barras[t].getDensiFuerzaF());

}////////////////////////////////////////////////////////////////
///////
        }
        mostrarParametroBarra(1);     //introducido 22102013
////////////////////////////////////////////////////////////////
//////////////////////
        if (chMostrar.isSelected()) {
            areaTexto.append("   Matriz de Densidad");
        }
        if (chMostrar.isSelected()) {
            areaTexto.append("   " +
Nodos.lienzo.superficie.DFF.salidaMatrizInt());
        }

//////////////////////////////ME//////////////////////////////////
///////
```

```
        // Creamos la matriz ME, de riguidez orifinal,  que es
cuadrada de dimensiÃ³n nuNudo X numNudo
        Nodos.lienzo.superficie.ME = new Matriz(numNuTotal,
numNuTotal);  //

//Nodos.lienzo.superficie.ME=Nodos.lienzo.superficie.ME.producto(Nodos
.lienzo.superficie.ME.producto(Nodos.lienzo.superficie.transG,
Nodos.lienzo.superficie.D),Nodos.lienzo.superficie.G);
        Nodos.lienzo.superficie.tempo = new
Matriz(Nodos.lienzo.superficie.transG.numFilas,
Nodos.lienzo.superficie.DFF.numColumnas); //inicialmente ernesto lo
puso al revÃ©s
//Nodos.tempo=Nodos.tempo.producto(Nodos.lienzo.superficie.transG,Nodo
s.lienzo.superficie.D);
        Nodos.lienzo.superficie.ME =
Nodos.lienzo.superficie.ME.producto(Nodos.lienzo.superficie.tempo.prod
ucto(Nodos.lienzo.superficie.transG, Nodos.lienzo.superficie.DFF),
Nodos.lienzo.superficie.G);
        // temp=temp.producto(Nodos.lienzo.superficie.D,
Nodos.lienzo.superficie.G);  //NOTA:: DE debe ordenarse para
materiales de densidad variable.
        //
Nodos.lienzo.superficie.ME=Nodos.lienzo.superficie.ME.producto(Nodos.l
ienzo.superficie.transG, temp);
        //temp=null; //elimino la matriz temporal
        if (chMostrar.isSelected()) {
            areaTexto.append("   Matriz ME= transG x D x G");
        }
        if (chMostrar.isSelected()) {
            areaTexto.append("   " +
Nodos.lienzo.superficie.ME.salidaMatriz());
        }

////////////////////////////////////////////////////////////////////
//////////////////////////////
// Creo la matriz de coordenadas de los aporyos MapXYZ
        Nodos.lienzo.superficie.MapXYZ = new Matriz(numNuTotal, 3);
        for (int k = 0; k < Nodos.lienzo.superficie.getNumApoyos();
k++) {
            Nodos.lienzo.superficie.MapXYZ.x[k][0] =
Nodos.lienzo.superficie.nuOrdenado[k].coordenada.X;
            Nodos.lienzo.superficie.MapXYZ.x[k][1] =
Nodos.lienzo.superficie.nuOrdenado[k].coordenada.Y;
            Nodos.lienzo.superficie.MapXYZ.x[k][2] =
Nodos.lienzo.superficie.nuOrdenado[k].coordenada.Z;
        }
        if (chMostrar.isSelected()) {
            areaTexto.append("   Matriz de coordenadas X,Y, Z de los
apoyos");
        }
        if (chMostrar.isSelected()) {
            areaTexto.append("   " +
Nodos.lienzo.superficie.MapXYZ.salidaMatriz());
        }
        // Creo la matriz MEC, a partir de ME en la que los valores
que no son de apoyos se hacen ceros. Por defecto todos los valores son
ceros.
        Nodos.lienzo.superficie.MEC = new
Matriz(Nodos.lienzo.superficie.ME.numFilas,
Nodos.lienzo.superficie.ME.numColumnas);
        //asigno los valores a MEC de ME
```

```
        Nodos.lienzo.superficie.MEC =
Nodos.lienzo.superficie.ME.clone();
        for (int i = 0; i < Nodos.lienzo.superficie.getNumApoyos();
i++) {
            for (int j = 0; j <
Nodos.lienzo.superficie.MEC.numColumnas; j++) {
                Nodos.lienzo.superficie.MEC.x[i][j] = 0;
            }
        }
        for (int l = 0; l < Nodos.lienzo.superficie.MEC.numFilas; l++)
{
            for (int m = Nodos.lienzo.superficie.getNumApoyos(); m <
Nodos.lienzo.superficie.MEC.numColumnas; m++) {
                Nodos.lienzo.superficie.MEC.x[l][m] = 0;
            }
        }
        if (chMostrar.isSelected()) {
            areaTexto.append("\n   Matriz ME modifica con ceros en los
nudos que no son apoyos");
        }
        if (chMostrar.isSelected()) {

areaTexto.append(Nodos.lienzo.superficie.MEC.salidaMatriz());
        }
        // realizo las operaciones para obtener FuApo multiplicando
MEC x MAPXYZ
        Nodos.lienzo.superficie.FuApo = new Matriz(numNuTotal, 3);
        Nodos.lienzo.superficie.FuApo =
Nodos.lienzo.superficie.FuApo.producto(Nodos.lienzo.superficie.MEC,
Nodos.lienzo.superficie.MapXYZ);
        if (chMostrar.isSelected()) {
            areaTexto.append("\n   Matriz de Fuerzas desde los apoyos
");
        }
        if (chMostrar.isSelected()) {
            areaTexto.append("   " +
Nodos.lienzo.superficie.FuApo.salidaMatriz());
        }
        Nodos.lienzo.superficie.FuSiste = new Matriz(numNuTotal, 3);
        for (int i = 0; i < Nodos.lienzo.superficie.getNumApoyos();
i++) {
            Nodos.lienzo.superficie.FuSiste.x[i][0] =
Nodos.lienzo.superficie.MapXYZ.x[i][0];
            Nodos.lienzo.superficie.FuSiste.x[i][1] =
Nodos.lienzo.superficie.MapXYZ.x[i][1];
            Nodos.lienzo.superficie.FuSiste.x[i][2] =
Nodos.lienzo.superficie.MapXYZ.x[i][2];
        }
        for (int i = Nodos.lienzo.superficie.getNumApoyos(); i <
numNuTotal; i++) {

//Nodos.lienzo.superficie.FuSiste.x[i][0]=Nodos.lienzo.superficie.nuOr
denado[i].carga.getCompoX();
            Nodos.lienzo.superficie.FuSiste.setElemento(i, 0,
Nodos.lienzo.superficie.nuOrdenado[i].carga.getCompoX());

//Nodos.lienzo.superficie.FuSiste.x[i][1]=Nodos.lienzo.superficie.nuOr
denado[i].carga.getCompoY();
            Nodos.lienzo.superficie.FuSiste.setElemento(i, 1,
Nodos.lienzo.superficie.nuOrdenado[i].carga.getCompoY());
```

```
//Nodos.lienzo.superficie.FuSiste.x[i][2]=Nodos.lienzo.superficie.nuOr
denado[i].carga.getCompoZ();
            Nodos.lienzo.superficie.FuSiste.setElemento(i, 2,
Nodos.lienzo.superficie.nuOrdenado[i].carga.getCompoZ());
        }
        Nodos.lienzo.superficie.FuSiste =
Nodos.lienzo.superficie.FuSiste.resta(Nodos.lienzo.superficie.FuSiste,
Nodos.lienzo.superficie.FuApo);
        if (chMostrar.isSelected()) {
            areaTexto.append("\n   (A1) Matriz de Fuerzas del Sistema,
FUSiste ");
        }
        if (chMostrar.isSelected()) {
            areaTexto.append("   " +
Nodos.lienzo.superficie.FuSiste.salidaMatriz());
        }
        // Creo la matriz MESiste, y la clono con ME;
        Nodos.lienzo.superficie.MESiste = new
Matriz(Nodos.lienzo.superficie.ME.numFilas,
Nodos.lienzo.superficie.ME.numColumnas);
        Nodos.lienzo.superficie.MESiste =
Nodos.lienzo.superficie.ME.clone();
        for (int i = 0; i < Nodos.lienzo.superficie.getNumApoyos();
i++) {
            for (int j = 0; j <
Nodos.lienzo.superficie.getNumApoyos(); j++) {
                if (i == j) {
                    Nodos.lienzo.superficie.MESiste.x[i][j] = 1;
                } else {
                    Nodos.lienzo.superficie.MESiste.x[i][j] = 0;
                }
            }
        }
        for (int k = Nodos.lienzo.superficie.getNumApoyos(); k <
Nodos.lienzo.superficie.MESiste.numFilas; k++) {
            for (int l = 0; l <
Nodos.lienzo.superficie.getNumApoyos(); l++) {
                Nodos.lienzo.superficie.MESiste.x[k][l] = 0;
            }
        }

        if (chMostrar.isSelected()) {
            areaTexto.append("\n   Matriz de ResoluciÃ³n del sistema,
MESiste ");
        }
        if (chMostrar.isSelected()) {
            areaTexto.append("   " +
Nodos.lienzo.superficie.MESiste.salidaMatriz());
        }
// Calcular la inversa de la matriz MESiste
        Nodos.lienzo.superficie.MESisteInv = new
Matriz(Nodos.lienzo.superficie.MESiste.numFilas,
Nodos.lienzo.superficie.MESiste.numColumnas);
        Nodos.lienzo.superficie.MESisteInv =
Nodos.lienzo.superficie.MESisteInv.inversa(Nodos.lienzo.superficie.MES
iste);
        if (chMostrar.isSelected()) {
            areaTexto.append("\n    Matriz inversa de la funciÃ³n
MESiste, MESisteInv ");
        }
```

```
        if (chMostrar.isSelected()) {
            areaTexto.append("   " +
Nodos.lienzo.superficie.MESisteInv.salidaMatriz());
        }
        //Crearemos la matruz resultao FormaOut es ordenada, las
filas, hasta el nÃºmero de apoyos son las reacciones en los apoyos.
        //Los elementos desde el nÃºmero de apoyos al final son las
coordenadas de las formas:
        Nodos.lienzo.superficie.FormaOut = new
Matriz(Nodos.lienzo.superficie.MESiste.numFilas, 3);
        Nodos.lienzo.superficie.FormaOut =
Nodos.lienzo.superficie.FormaOut.producto(Nodos.lienzo.superficie.MESi
steInv, Nodos.lienzo.superficie.FuSiste);
        if (chMostrar.isSelected()) {
            areaTexto.append("\n   Matriz de Resultados de la forma de
la superficie ");
        }
        if (chMostrar.isSelected()) {
            areaTexto.append("   " +
Nodos.lienzo.superficie.FormaOut.salidaMatriz());
        }
        ////////////////////////////cumplimentaciÃ³n del array de la
forma de salida//////////////////
        //Ahora hay que llenar los puntos de nuSalida que son los
mismos que los de NuOrdenado pero en los que cambian
        //las coordenadas. El cambio lo abtengo a parir de la matriz
FormaOut, a partir de la fina i=numApoyo
        for (int k = 0; k < Nodos.lienzo.superficie.nudo.length; k++)
{
            Nodos.lienzo.superficie.nuSalida[k] =
Nodos.lienzo.superficie.nuOrdenado[k];
        }
        for (int k = Nodos.lienzo.superficie.getNumApoyos(); k <
numNuTotal; k++) {

Nodos.lienzo.superficie.nuSalida[k].setX(Nodos.lienzo.superficie.Forma
Out.x[k][0]);

Nodos.lienzo.superficie.nuSalida[k].setY(Nodos.lienzo.superficie.Forma
Out.x[k][1]);

Nodos.lienzo.superficie.nuSalida[k].setZ(Nodos.lienzo.superficie.Forma
Out.x[k][2]);
        }
        //llamamos la funciÃ³n que genera el array de Rracciones
        Nodos.lienzo.superficie.reacciones();
        if (chMostrar.isSelected()) {
            areaTexto.append("\n REACCIONES EN LOS APOYOS\n ");
        }
        for (int r = 0; r < Nodos.lienzo.superficie.numApoyos; r++) {
            if (chMostrar.isSelected()) {
                areaTexto.append("(" +
Nodos.lienzo.superficie.aReac[r].X + "," +
Nodos.lienzo.superficie.aReac[r].Y + "," +
Nodos.lienzo.superficie.aReac[r].Z + ")\n");
            }
            btnMatrizMasas.doClick();
            btnAjusteMasas.setEnabled(true);
        }
    }
```

```
////////////////////////////////////////////////////////////////////
////////////////////

    void ventEleFinito() {
        cbMaterialEF = new JComboBox(this.sMaterial);
        JLabel lAviso = new JLabel("Elementos Finitos por
nÃºmero,inicio o final\n");
        JPanel pNavegaEF = new JPanel();
        pNavegaEF.setBackground(Color.ORANGE);
        pMuestraEF = new JPanel();
        pMuestraEF.setBackground(Color.CYAN);
        pNavegaEF.add(new JLabel("Num.E.Finito(n,or) "));
        tNumEF = new JTextField(" ", 4);
        tEFEtiqueta = new JTextField(" ", 6);
        tEFSuperficie = new JTextField(" ", 4);
        tEFDensiMasa = new JTextField(" ", 4);
        tEFMasa = new JTextField(" ", 4);
        tEFGrosor = new JTextField(" ", 4);
        btnEFInicio = new JButton("0<<");
        btnEFMenos = new JButton("<=");
        btnEFMas = new JButton("=>");
        btnEFFinal = new JButton(">>" +
String.valueOf(Nodos.lienzo.superficie.eleFinito.length - 1));
        btnEFAceptarCambios = new JButton("OK");
        pNavegaEF.add(tNumEF);
        tNumEF.setEditable(true);
        pNavegaEF.add(btnEFInicio);
        pNavegaEF.add(btnEFMenos);
        pNavegaEF.add(btnEFMas);
        pNavegaEF.add(btnEFFinal);
        pNavegaEF.add(cbMaterialEF);

        pMuestraEF.setLayout(new GridLayout(1, 9, 2, 2)); //filas,
columnas,separaciÃ³n columnas ,separaciÃ³n filas
        pMuestraEF.add(new JLabel("Etiquetq Elemento Finito "));
        pMuestraEF.add(tEFEtiqueta);
        tEFEtiqueta.setEditable(false);
        pMuestraEF.add(new JLabel("Superficie"));
        pMuestraEF.add(tEFSuperficie);
        pMuestraEF.add(new JLabel("Grosor"));
        pMuestraEF.add(tEFGrosor);//tEFGrosor.setEditable(false);
        pMuestraEF.add(new JLabel("Densidad Masa del Elemento Finito
"));
        pMuestraEF.add(tEFDensiMasa);
        pMuestraEF.add(new JLabel("Masa del Elemento Finito "));
        pMuestraEF.add(tEFMasa);
        pMuestraEF.add(btnEFAceptarCambios);

        Nodos.ventEleFinito.setLayout(new BorderLayout());
        Nodos.ventEleFinito.add(pNavegaEF, BorderLayout.NORTH);
        Nodos.ventEleFinito.add(pMuestraEF, BorderLayout.SOUTH);
        Nodos.ventEleFinito.setIconImage(new
javax.swing.ImageIcon(getClass().getResource("imagenes/nodos2.png")).g
etImage());
        Nodos.ventEleFinito.setTitle("Nodos: Navegacion elementos
finitos ");
        Nodos.ventEleFinito.setSize(600, 125);
        Nodos.ventEleFinito.setResizable(true);
        Nodos.ventEleFinito.setVisible(true);
////////////////////////////////////////
```

```
        this.tNumEF.addKeyListener(new java.awt.event.KeyAdapter() {
            @Override
            public void keyReleased(java.awt.event.KeyEvent evt) {
                int t = Integer.valueOf(tNumEF.getText().trim());
                //
tEFEtiqueta.setText(Nodos.lienzo.superficie.eleFinito[t].getEtiqueta()
);
                numEF = Nodos.lienzo.superficie.mostrarParametroEF(t);
                Lienzo.nEFActiva = t;
                Nodos.lienzo.repaint();
            }
        });

        ////////////////////////////////////
        this.btnEFAceptarCambios.addActionListener(new
ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                int k =
Nodos.lienzo.superficie.eleFinito[numEF].getOrden();
                Nodos.lienzo.superficie.eleFinito[k].setGrosor(0.0001
* Double.parseDouble(tEFGrosor.getText()));
                //void setParametrosMaterial(double den, double lE,
double mE, double coeSeg){
                Nodos.lienzo.superficie.eleFinito[k].densiMasa =
Double.parseDouble(tEFDensiMasa.getText());

Nodos.lienzo.superficie.eleFinito[k].setMaterial(Nodos.lienzo.superfic
ie.dMaterial);
                numEF = Nodos.lienzo.superficie.mostrarParametroEF(k);
            }
        });
////////////////////////////////////////////
        this.btnEFMas.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (numEF < Nodos.lienzo.superficie.eleFinito.length -
1 && numEF > -1) {
                    numEF++;
                    numEF =
Nodos.lienzo.superficie.mostrarParametroEF(numEF);
                }
            }
        });
        ////////////////////////////////////////////
        this.btnEFMenos.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (numEF > 0 && numEF <
Nodos.lienzo.superficie.eleFinito.length) {
                    numEF--;
                    numEF =
Nodos.lienzo.superficie.mostrarParametroEF(numEF);
                }
            }
        });
        ////////////////////////////////////////////
        this.btnEFInicio.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                numEF = Nodos.lienzo.superficie.mostrarParametroEF(0);
```

```
            }
        });
        ////////////////////////////////////////////////
        this.btnEFFinal.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                numEF =
Nodos.lienzo.superficie.mostrarParametroEF(Nodos.lienzo.superficie.ele
Finito.length - 1);
            }
        });

        // String[] sMaterial={"Acero B500","Acero S235","Acero
S275","Acero S355","Acero Y1760","Acero Y1830","Aluminio","Textil
T502","Textil T1202", "HA 30","HA 40","HA 50","DEFINIDO USUARIO"};
        cbMaterialEF.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String str = "                                    ";
                cbMaterial = (JComboBox) e.getSource();
//TipoEleFinito);
                str = (String) cbMaterial.getSelectedItem();
                if (str == "Acero B500") {
                    Nodos.lienzo.superficie.dMaterial[0] = 7850.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 500e6;
                    Nodos.lienzo.superficie.dMaterial[2] = 19e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 2.5;
                }
                if (str == "Acero S235") {
                    Nodos.lienzo.superficie.dMaterial[0] = 7850.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 235e6;
                    Nodos.lienzo.superficie.dMaterial[2] = 21e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 2.5;
                };
                if (str == "Acero S275") {
                    Nodos.lienzo.superficie.dMaterial[0] = 7850.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 2750e6;
                    Nodos.lienzo.superficie.dMaterial[2] = 21e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 2.5;
                };
                if (str == "Acero S355") {
                    Nodos.lienzo.superficie.dMaterial[0] = 7850.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 355e6;
                    Nodos.lienzo.superficie.dMaterial[2] = 21e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 2.5;
                };
                if (str == "Acero Y1760") {
                    Nodos.lienzo.superficie.dMaterial[0] = 7850.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 1760e6;
                    Nodos.lienzo.superficie.dMaterial[2] = 21e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 2.5;
                };
                if (str == "Acero Y1830") {
                    Nodos.lienzo.superficie.dMaterial[0] = 7850.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 1830e6;
                    Nodos.lienzo.superficie.dMaterial[2] = 21e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 2.5;
                };
                if (str == "Aluminio") {
                    Nodos.lienzo.superficie.dMaterial[0] = 2700.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 70e6;
```

```
                    Nodos.lienzo.superficie.dMaterial[2] = 2.1e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 2.5;
                };
                if (str == "Textil T502") {
                    Nodos.lienzo.superficie.dMaterial[0] = 7850.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 5.0e8;
                    Nodos.lienzo.superficie.dMaterial[2] = 21e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 2.5;
                };
                if (str == "Textil T1202") {
                    Nodos.lienzo.superficie.dMaterial[0] = 7850.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 5.0e8;
                    Nodos.lienzo.superficie.dMaterial[2] = 21e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 2.5;
                };
                if (str == "HA 30") {
                    Nodos.lienzo.superficie.dMaterial[0] = 2500.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 5.0e8;
                    Nodos.lienzo.superficie.dMaterial[2] = 30e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 1.5;
                };
                if (str == "HA 40") {
                    Nodos.lienzo.superficie.dMaterial[0] = 2500.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 5.0e8;
                    Nodos.lienzo.superficie.dMaterial[2] = 40e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 1.5;
                };
                if (str == "HA 50") {
                    Nodos.lienzo.superficie.dMaterial[0] = 2500.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 5.0e8;
                    Nodos.lienzo.superficie.dMaterial[2] = 50e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 1.5;
                };
                if (str == "DEFINIDO USUARIO") {
                    Nodos.lienzo.superficie.dMaterial[0] = 2500.0;
                    Nodos.lienzo.superficie.dMaterial[1] = 5.0e8;
                    Nodos.lienzo.superficie.dMaterial[2] = 21e10;
                    Nodos.lienzo.superficie.dMaterial[3] = 2.5;
                };

Nodos.lienzo.superficie.eleFinito[numEF].setMaterial(Nodos.lienzo.supe
rficie.dMaterial);

Nodos.lienzo.superficie.eleFinito[numEF].setGrosor(es);
            }
        });

    }

}
//////////////////////////////////////////FIN//////////////////
/////////////
```

Clase Punto

```
/**
 *
 * @author julio MuÃ±iz pADILLA Y Ernesto MuÃ±iz MartÃn
 */
package nodos;

import java.io.Serializable;


public final class Punto implements Serializable {
    protected double X;
    protected double Y;
    protected double Z;
 void   Punto(){};
 public void Punto(double x, double y, double z){
    setPunto(x,y,z);
    }

  public void setPunto(double coordx, double coordy, double coordz) {
          X = coordx;
          Y = coordy;
          Z= coordz;    }

public double getX(){ return X;}
public double getY(){ return Y;}
public double getZ(){ return Z;}

public Punto escalar(double escala){
 Punto salida=new Punto();
salida.X=this.X*escala;
salida.Y=this.Y*escala;
salida.Z=this.Z*escala;
return salida;
}
public Punto restar(Punto A,Punto B){
Punto salida= new Punto();
salida.X=A.X-B.X;
salida.Y=A.Y-B.Y;
salida.Z=A.Z-B.Z;
return salida;
}

public String toPrint() {return "[" + X + "," + Y +","+Z+ "]"; }
}
```

Clase Superficie

```
/*/*
()Julio MuÃ±iz Padilla
para una tesis doctoral del ingeniero Ernesto MuÃ±iz MartÃn
 2014
 */

package nodos;

//import java.awt.Point;

import java.io.Serializable;
//import java.util.Formatter;
import javax.swing.JOptionPane;
//import static nodos.Paneli.tEFGrosor;
import static nodos.Paneli.tNumEF;
/**
 *
 * @author julio
 */
public class Superficie  implements Serializable {
 static  boolean isCarga=false;
static double[] dMaterial=new double[4];
double[] EPE; // Array Suma de todos los elementos de la columna de EP
double[] ACIN; // Array Suma de todos los elementos de la columna de
CIN
double[] valoresPropios;
Nudo[] nudo;
Nudo[] nuOrdenado;
Nudo[] nuSalida;
Nudo[] NudoO;
Nudo[] apoyo;
Nudo[] esquina;
Barra[] barras; //.
Nudo[] vibra;
static Punto[] aReac; //array de reacciones
static Nudo[] temporal;
double FactorMasa=3.0;

double Ancho=0, Alto=0;
double dX=1, dY=1; //hay que impedir la divisiÃ³n por cero.
double dZ=0;
int numEleFinito=0;
char tipoEleFinito;
double grosorEF=0; //grosor por defecto de los elementos finitos.
double densiEF=0;// densiedad por defecto
 EleFinito[] eleFinito;
int numNudosX = 0;
int numNudosY=0;
int numNudos=0;;
int numApoyos=0;
int numEsquinas=0;
int numBarras=0;
double seccionBarra=0.0001;
int numBarrasEx=0;
```

```
int numNuNoExt=0; //son los nudos que se generan inicialmente. Se
utilizarÃ¡n para listar por separado los externos de los de la
superficie.
static  Nudo nuExterno;

//Point[] pixelApo;
static Barra[] barraTemp;
double densiFuerzaBarraForma=1; //se trata de la densidad genÃ©rica de
fuerza de cada barra. Este valor se tomarÃ¡ por defecto
private double masa=1; //este valor serÃ¡ un resultado final
Vinculo[] vinculoInterno;
static Matriz tempo;
static   Matriz Origen;
Matriz G;
Matriz transG;
Matriz DFF;
Matriz DD; //matriz densidades de fuerza calculada. Densidad Dinamica
Matriz ME;
Matriz MDE; //equivalente a ME pero con DD en vez de D
Matriz MapXYZ; // declaraciÃ³n de la matriz de coordenadas de los
apoyos de (numero de nudos X 3)
Matriz MEC; //matriz ME en la que se hacen cero los valores
correspondientes a nudos que no son apoyos ni externos.
Matriz FuApo; // matriz de las fuerzas desde los puntos de apoyos
(numero de nudos X 3)
Matriz FuNudo;
Matriz FuSiste; // matriz para el calculo (numApoyox X 3) se obtiene
por combinaciÃ³n de FuNudo , MapXYZ y fu Apo
Matriz MESiste;
Matriz MESisteInv;
Matriz FormaOut;
Matriz Masa; //matriz diagonal que asigna las masas a los nudos
ordenados, empezando por apoyos.
Matriz MasaR;//matriz cuadrada de masas sin los apoyos
Matriz MDER;//matriz MDE sin elementos apoyos
Matriz MasaRI;//matriz inversa de masas sin los apoyos
Matriz MA;// Matriz de la que vamos a botener los autovalor

Matriz VP; //autvectores
Matriz MF; //matriz de autovectores orlada con tantas filas como
apoyos
Matriz AL; //Matriz alargamiento
Matriz EAL; //matriz de energÃa de alargamiento obtenida a partir de
la aenterior 1/2 por el cuadrado
Matriz EP;//matriz de energÃa potencial EP=DD x EAL
Matriz EC;// matriz VP con elementos al cuadrado /2;
Matriz CIN; // MasaRxEC

static int interacciones=1000;

public void setNumApoyos(int apo){this.numApoyos=apo;}
public void setNumBarras(int bar){this.numBarras=bar;}
public void setNumBarrasEX(int barEx){this.numBarrasEx=barEx;}
public void setTipoEleFinito(char ef){this.tipoEleFinito=ef;}
char getTipoEleFinito(){return this.tipoEleFinito;}
int getNumNudo(){return this.numNudos;}
int getNumEleFinito(){return this.numEleFinito;}
double getAncho(){return this.Ancho;}
double getAlto(){return this.Alto;}
int getNumNudosX(){return this.numNudosX;}
int getNumNudosY(){return this.numNudosY;}
```

```
int getNumApoyos( ){return this.numApoyos;}
int getNumBarras( ){return this.numBarras;} //da las barras de la
superficie
int getNumBarrasEx( ){return this.numBarrasEx;} //da las barras
externas
//////////////////////geometrÃa inicial//////////////////////////
public void setGeoInicio(double anchoRect, double altoRect,double
anchoEleFinitoX, double altoEleFinitoY,char tipMall)
{
this.tipoEleFinito=tipMall;
this.dX=anchoEleFinitoX;this.dY=altoEleFinitoY;
//this.dX=this.dY=this.dX; // esta lÃnea hace que los elementos
infinitesimales sean cuadrados, no rectÃ¡ngulos.
this.numNudosX=1+(int)Math.round(anchoRect/this.dX);
this.numNudosY=1+(int)Math.round(altoRect/this.dY);
if(this.numNudosX % 2 !=0){this.numNudosX=this.numNudosX+1;
this.dX=anchoRect/(this.numNudosX-1);}
if(this.numNudosY % 2 !=0){this.numNudosY=this.numNudosY+1;
this.dY=altoRect/(this.numNudosY-1);}
this.Ancho=(this.numNudosX-1)*this.dX;  //se redimenciona el ancho
para que no ha
this.Alto=(this.numNudosY-1)*this.dY;  //
this.numNudos=this.numNudosX*this.numNudosY;
this.nudo =new Nudo[this.numNudos]; //creo el array de nudos, todos
consecutivos
this.numNuNoExt=this.numNudos;
for(int n=0;n<this.nudo.length;n++){
 nudo[n]=new Nudo();this.nudo[n].setNudo(0, 0, 0, tipMall);
this.nudo[n].setOrden(n);nudo[n].carga=new JuVector(0,0,0);} //creo
todos los elementos del array nudos y los inicio y le doy el orden de
la figura.

int l=-1;  //no modificar debe ser -1 ya lo he intentado otras veces.
//cuando se llame a la funcion ordenar ese orden se mantendrÃ¡ aunque
no coicidirÃ¡ con el de la posiciÃ³n en la matriz ordenada
   for(int i=0;i<this.numNudosX;i++){
        for(int j=0;j<this.numNudosY;j++){if(l<this.nudo.length) l++ ;
//el primer n es cero
         this.nudo[l].setNudo(i*this.dX, j*this.dY, 0, tipMall);
          } }
 asignaNudosEleFinitos(tipMall,this.numNudosX, this.numNudosY);
setGeometria(tipMall);
}

////////////////////////////////////////////////////////////////////
//////////
public void   setGeometria(char tipMall)
{
this.tipoEleFinito=tipMall;
for(int
n=0;n<this.numNuNoExt;n++){this.nudo[n].settipoEleFinito(tipMall);}
//se actualiza el tipo de malla.

for(int k=0;k<this.numNuNoExt;k++){
 //this.numBarras=this.numBarras + this.asignaVinculo(k,tipMall,
this.numNudosY);
 this.numBarras=this.numBarras + this.asignaVinculo(k,tipMall,
this.numNudosY);//lamada a la funciÃ³n de asignaciÃ³n de la
}
}
```

```
//*****************************************************************
***************//

  Matriz asignaMatrizVinculoB(Matriz Ma,Nudo[] nud){ //funcion
original sin la BB
  int n=0;
//Matriz Mout=new Matriz(Ma.numFilas,Ma.numColumnas); //julio 04072013
16:47

for(int i=0;i< nud.length  ;i++){
       //for(int j=0;j<this.nudo[i].nBarras;j++){
      for(int j=0;j<nud[i].vinculo.length ;j++){
       if(Ma.x[n][nud[i].vinculo[j].getOrden()]==0 &&
nud[i].vinculo[j].getOrden()>i  ){
           Ma.x[n][nud[i].vinculo[j].getOrden()]=-1;
           if(Ma.x[n][i]==0)  Ma.x[n][i]=1;
          n++;}
            }
       }
   return Ma;
  };

  ////////////////////////////
   Matriz asignaMatrizVinculoOrigen(Matriz Ma,Nudo[] nud){  //funciÃ³n
modificada
  int n=0;
Matriz Mout=new Matriz(Ma.numFilas,Ma.numColumnas); //julio 04072013
16:47
for(int h=0;h<Mout.numFilas ;h++){for(int
j=0;j<Mout.numColumnas;j++){Mout.x[h][j]=0;}}//iniciamos todos los
elementos=0;/ this.numBarras=this.getNumeroBarras();
for(int i=0;i< nud.length  ;i++){
    for(int j=0;j<nud[i].vinculo.length ;j++){
 //System.out.print("n="+String.valueOf(n)+", i="+String.valueOf(i)+",
j="+String.valueOf(j)+" 0rj="+
String.valueOf(nud[i].vinculo[j].getOrden())+"\n");
   if(nud[i].vinculo[j].getOrden()-i<0 ){ }
  else if(Ma.x[n][nud[i].vinculo[j].getOrden()]==0 &&
nud[i].vinculo[j].getOrden()>i     )
    {  Mout.x[n][nud[i].vinculo[j].getOrden()]=-1;
       if(Ma.x[n][i]==0)  Mout.x[n][i]=1;
       n++;
    }
    }
     }
   return Mout;
  };


////////////////////////////////////////////////////////////////////
//////
 Matriz asignaMatrizVinculoG(Matriz Ma,Nudo[] nud){  //funciÃ³n
modificada
  int n=0;
Matriz Mout=new Matriz(Ma.numFilas,Ma.numColumnas); //julio 04072013
16:47
for(int i=0;i<Mout.numFilas ;i++){for(int
j=0;j<Mout.numColumnas;j++){Mout.x[i][j]=0;}}//iniciamos todos los
elementos=0;/ this.numBarras=this.getNumeroBarras();
for(int i=0;i< nud.length  ;i++){
```

```
  int ii=nud[i].getOrden();       //for(int
j=0;j<this.nudo[i].nBarras;j++){

    for(int j=0;j<nudo[ii].vinculo.length ;j++){  //me da el tamaÃ±o
del vinculo en la array origina desordenada
  int jj=nudo[ii].vinculo[j].getIndice();
 // System.out.print("n="+String.valueOf(n)+",
i="+String.valueOf(i)+", ii="+String.valueOf(nud[i].getOrden())+",
j="+String.valueOf(j)+" 0rj="+ String.valueOf(jj)+"\n");

  if(jj-i<1){ System.out.print( "para i="+String.valueOf(i)+ "  y
j="+String.valueOf(i)+" salto"); }
  else //if(Ma.x[n][nudo[nud[i].getOrden()].vinculo[j].getIndice()]==0
&& nudo[nud[i].getOrden()].vinculo[j].getIndice()>nud[i].getIndice()
)
    {
           Mout.x[n][jj]=-1;
         //  if(Ma.x[n][nud[i].getOrden()]==0)
           Mout.x[n][i]=1;
          n++;    }
    }
     }
   return Mout;
  };


////////////////////////////////////////////////////////////////////
//////////
void romperVinculo(){};  //no aisgnada;

////////////////////////////////////////////////////////////////////
////////77

int  asignaVinculo( int h, char EleFinito,  int nXc){
      //funcion  IMPORTANTISIMA
        int nX= (int)nXc;  //int nY=(int)nYc;
        int n=this.nudo[h].getOrden();
        //se inicia la cuenta en el sentido horario a las 9 menos
cuarto,  // las 4 esquinas deben ir al final
if(this.nudo[n].externo){}
 else if((n>0 && n< nX-1) && EleFinito=='t' ){ //borde superior
quitando las esquinas
 this.nudo[n].tipoNudo='l';this.nudo[n].setNumBarrasS(4);
this.nudo[n].vinculo=new Nudo[this.nudo[n].getNumBarrasT()];
  this.nudo[n].vinculo[0]= new
Nudo();this.nudo[n].vinculo[0]=this.nudo[n-1];
  this.nudo[n].vinculo[1]= new
Nudo();this.nudo[n].vinculo[1]=this.nudo[n+1];
  this.nudo[n].vinculo[2]= new
Nudo();this.nudo[n].vinculo[2]=this.nudo[n+nX+1];
  this.nudo[n].vinculo[3]= new
Nudo();this.nudo[n].vinculo[3]=this.nudo[n+nX];//Or.x[n][n+1]=1;
 }
  else if((n>this.numNudos-nX && n< this.nudo.length-1) &&
EleFinito=='t' ) //borde inferior quitando las esquinas

{this.nudo[n].tipoNudo='l';this.nudo[n].setNumBarrasS(4);this.nudo[n].
vinculo=new Nudo[this.nudo[n].getNumBarrasT()];
   this.nudo[n].vinculo[0]= new
Nudo();this.nudo[n].vinculo[0]=this.nudo[n-1];//Or.x[n][n-1]=1;
```

```
   this.nudo[n].vinculo[1]= new
Nudo();this.nudo[n].vinculo[1]=this.nudo[n-nX-1];//Or.x[n][n+1]=1;
   this.nudo[n].vinculo[2]= new
Nudo();this.nudo[n].vinculo[2]=this.nudo[n-nX];//Or.x[n][n-nX]=1;
     this.nudo[n].vinculo[3]= new
Nudo();this.nudo[n].vinculo[3]=this.nudo[n+1];//Or.x[n][n-nX]=1;
  }

 else if((n>0 && n< nX-1) && EleFinito=='c' ) //borde inferior
quitando las esquinas
  {this.nudo[n].tipoNudo='l';this.nudo[n].setNumBarrasS(3);
this.nudo[n].vinculo=new Nudo[this.nudo[n].getNumBarrasT()];
   this.nudo[n].vinculo[0]= new
Nudo();this.nudo[n].vinculo[0]=this.nudo[n-1];//Or.x[n][n-1]=1;
   this.nudo[n].vinculo[1]= new
Nudo();this.nudo[n].vinculo[1]=this.nudo[n+nX];//Or.x[n][n+nX]=1;
   this.nudo[n].vinculo[2]= new
Nudo();this.nudo[n].vinculo[2]=this.nudo[n+1];//Or.x[n][n+1]=1;
    }
  else if((n>this.numNudos-nX && n< this.nudo.length-1) &&
EleFinito=='c' ) //borde superior quitando las esquinas

{this.nudo[n].tipoNudo='l';this.nudo[n].setNumBarrasS(3);this.nudo[n].
vinculo=new Nudo[this.nudo[n].getNumBarrasT()];
   this.nudo[n].vinculo[0]= new
Nudo();this.nudo[n].vinculo[0]=this.nudo[n-1];//Or.x[n][n-1]=1;
   this.nudo[n].vinculo[1]= new
Nudo();this.nudo[n].vinculo[1]=this.nudo[n+1];//Or.x[n][n+1]=1;
   this.nudo[n].vinculo[2]= new
Nudo();this.nudo[n].vinculo[2]=this.nudo[n-nX];//Or.x[n][n-nX]=1;
  }
    else if(( n != 0 && n!= this.nudo.length-nX)&&( n%nX==0 &&
EleFinito=='t')) {//borde izquierdo quitando las esquinas
   this.nudo[n].setNumBarrasS(4);this.nudo[n].vinculo=new
Nudo[this.nudo[n].getNumBarrasT()];this.nudo[n].tipoNudo='l';
   this.nudo[n].vinculo[0]= new
Nudo();this.nudo[n].vinculo[0]=this.nudo[n-nX];//Or.x[n][n+nX]=1;
   this.nudo[n].vinculo[1]= new
Nudo();this.nudo[n].vinculo[1]=this.nudo[n+1];//Or.x[n][n+1]=1;
   this.nudo[n].vinculo[2]= new
Nudo();this.nudo[n].vinculo[2]=this.nudo[n+nX+1];//Or.x[n][n-nX]=1;
   this.nudo[n].vinculo[3]= new
Nudo();this.nudo[n].vinculo[3]=this.nudo[n+nX];//Or.x[n][n-nX]=1;

    }
  else if(( n != 0 && n!= this.nudo.length-nX)&&( n%nX==0 &&
EleFinito=='c')) { //borde izquierdo quitando las esquinas
   this.nudo[n].setNumBarrasS(3);this.nudo[n].vinculo=new
Nudo[this.nudo[n].getNumBarrasT()];this.nudo[n].tipoNudo='l';
   this.nudo[n].vinculo[0]= new
Nudo();this.nudo[n].vinculo[0]=this.nudo[n+nX];//Or.x[n][n+nX]=1;
   this.nudo[n].vinculo[1]= new
Nudo();this.nudo[n].vinculo[1]=this.nudo[n+1];//Or.x[n][n+1]=1;
   this.nudo[n].vinculo[2]= new
Nudo();this.nudo[n].vinculo[2]=this.nudo[n-nX];//Or.x[n][n-nX]=1;
  }
   else if((  (n+1)%nX==0 && n !=nX-1 && n !=this.nudo.length-1 &&
EleFinito=='t')) {//borde derecho quitando las esquinas
   this.nudo[n].setNumBarrasS(4);this.nudo[n].vinculo=new
Nudo[this.nudo[n].getNumBarrasT()];this.nudo[n].tipoNudo='l';
```

```
   this.nudo[n].vinculo[0]= new
Nudo();this.nudo[n].vinculo[0]=this.nudo[n-1];//Or.x[n][n+nX]=1;
   this.nudo[n].vinculo[1]= new
Nudo();this.nudo[n].vinculo[1]=this.nudo[n-nX-1];//Or.x[n][n+1]=1;
   this.nudo[n].vinculo[2]= new
Nudo();this.nudo[n].vinculo[2]=this.nudo[n-nX];//Or.x[n][n-nX]=1;
   this.nudo[n].vinculo[3]= new
Nudo();this.nudo[n].vinculo[3]=this.nudo[n+nX];//Or.x[n][n-nX]=1;

    }

  else if( (n+1)%nX==0 && n !=nX-1 && n !=this.nudo.length-1 &&
EleFinito=='c' ) //borde derecho quitando las esquinas
  {this.nudo[n].setNumBarrasS(3);this.nudo[n].vinculo=new
Nudo[this.nudo[n].getNumBarrasT()];this.nudo[n].tipoNudo='l';
   this.nudo[n].vinculo[0]= new
Nudo();this.nudo[n].vinculo[0]=this.nudo[n-nX];//Or.x[n][n-nX]=1;
   this.nudo[n].vinculo[1]= new
Nudo();this.nudo[n].vinculo[1]=this.nudo[n-1];//Or.x[n][n-1]=1;
   this.nudo[n].vinculo[2]= new
Nudo();this.nudo[n].vinculo[2]=this.nudo[n+nX];//Or.x[n][n+nX]=1;
  }

  else if((n>0 && n< nX-1) && EleFinito=='d' ){ //borde inferior
quitando las esquinas
   this.nudo[n].setNumBarrasS(5); this.nudo[n].vinculo=new
Nudo[this.nudo[n].getNumBarrasT()]; this.nudo[n].tipoNudo='l';
   this.nudo[n].vinculo[0]= new Nudo();this.nudo[n].vinculo[0]=
this.nudo[n-1];//Or.x[n][n-1]=1;
   this.nudo[n].vinculo[1]= new Nudo(); this.nudo[n].vinculo[1]=
this.nudo[n+nX-1];//Or.x[n][n+nX-1]=1;
   this.nudo[n].vinculo[2]= new Nudo();
this.nudo[n].vinculo[2]=this.nudo[n+nX];//Or.x[n][n+nX]=1;
   this.nudo[n].vinculo[3]= new Nudo();
this.nudo[n].vinculo[3]=this.nudo[n+nX+1];//Or.x[n][n+nX+1]=1;
   this.nudo[n].vinculo[4]= new Nudo();
this.nudo[n].vinculo[4]=this.nudo[n+1];//Or.x[n][n+1]=1;
                              }

  else if((n>this.numNudos-nX && n< this.nudo.length-1) &&
EleFinito=='d' ){ //borde superior quitando las esquinas
  this.nudo[n].setNumBarrasS(5);this.nudo[n].vinculo=new
Nudo[this.nudo[n].getNumBarrasT()];  this.nudo[n].tipoNudo='l';
   this.nudo[n].vinculo[0]= new Nudo();this.nudo[n].vinculo[0]=
this.nudo[n-1];//Or.x[n][n-1]=1;
   this.nudo[n].vinculo[1]= new Nudo(); this.nudo[n].vinculo[1]=
this.nudo[n+1];//Or.x[n][n+1]=1;
   this.nudo[n].vinculo[2]= new Nudo();
this.nudo[n].vinculo[2]=this.nudo[n-nX+1];//Or.x[n][n-nX+1]=1;
   this.nudo[n].vinculo[3]= new Nudo();
this.nudo[n].vinculo[3]=this.nudo[n-nX];//Or.x[n][n-nX]=1;
   this.nudo[n].vinculo[4]= new Nudo();
this.nudo[n].vinculo[4]=this.nudo[n-nX-1];// Or.x[n][n-nX-1]=1;
                              }
  else if(( n != 0 && n!= this.nudo.length-nX)&&( n%nX==0 &&
EleFinito=='d')) //borde izquierdo quitando las esquinas
  {this.nudo[n].setNumBarrasS(5);this.nudo[n].vinculo=new
Nudo[this.nudo[n].getNumBarrasT()];
   this.nudo[n].vinculo[0]= new
Nudo();this.nudo[n].vinculo[0]=this.nudo[n+nX];//Or.x[n][n+nX]=1;
```

```
   this.nudo[n].vinculo[1]= new
Nudo();this.nudo[n].vinculo[1]=this.nudo[n+nX+1];//Or.x[n][n+nX+1]=1;
   this.nudo[n].vinculo[2]= new
Nudo();this.nudo[n].vinculo[2]=this.nudo[n+1];//Or.x[n][n+1]=1;
   this.nudo[n].vinculo[3]= new Nudo();
this.nudo[n].vinculo[3]=this.nudo[n-nX+1];//Or.x[n][n-nX+1]=1;
   this.nudo[n].vinculo[4]= new Nudo();
this.nudo[n].vinculo[4]=this.nudo[n-nX];//Or.x[n][n-nX]=1;
  }
  else if( (n+1)%nX==0 && n !=nX-1 && n !=this.nudo.length-1 &&
EleFinito=='d' ) //borde derecho  quitando las esquinas
  {this.nudo[n].setNumBarrasS(5);this.nudo[n].vinculo=new
Nudo[this.nudo[n].getNumBarrasT()];
   this.nudo[n].vinculo[0]= new
Nudo();this.nudo[n].vinculo[0]=this.nudo[n-1];//Or.x[n][n-1]=1;
   this.nudo[n].vinculo[1]= new
Nudo();this.nudo[n].vinculo[1]=this.nudo[n+nX-1];//Or.x[n][n+nX-1]=1;
   this.nudo[n].vinculo[2]= new
Nudo();this.nudo[n].vinculo[2]=this.nudo[n+nX];//Or.x[n][n+nX]=1;
    this.nudo[n].vinculo[3]= new Nudo();
this.nudo[n].vinculo[3]=this.nudo[n-nX];//Or.x[n][n-nX]=1;
   this.nudo[n].vinculo[4]= new Nudo();
this.nudo[n].vinculo[4]=this.nudo[n-nX-1]; //Or.x[n][n-nX-1]=1;
  }

 ////////////////////////////////////////esquinas de las
superficie
  else if(n==0 && EleFinito=='c'){  //esquina origen, inferiior
izquierda, n=0


this.nudo[n].tipoNudo='e';this.nudo[n].setNumBarrasS(2);this.nudo[n].v
inculo=new Nudo[this.nudo[n].getNumBarrasT()];
   this.nudo[n].vinculo[0]= new
Nudo();this.nudo[n].vinculo[0]=this.nudo[n+nX];//Or.x[n][n+nX]=1;
   this.nudo[n].vinculo[1]= new
Nudo();this.nudo[n].vinculo[1]=this.nudo[n+1];//Or.x[n][n+1]=1;
  }

  else if(n==0 && EleFinito=='d'){  //esquina origen, inferiior
izquierda, n=0 malla diagonada


this.nudo[n].tipoNudo='e';this.nudo[n].setNumBarrasS(3);this.nudo[n].v
inculo=new Nudo[this.nudo[n].getNumBarrasT()];
   this.nudo[n].vinculo[0]= new
Nudo();this.nudo[n].vinculo[0]=this.nudo[n+nX];//Or.x[n][n+nX]=1;
   this.nudo[n].vinculo[1]= new
Nudo();this.nudo[n].vinculo[1]=this.nudo[n+nX+1];//Or.x[n][n+nX+1]=1;
   this.nudo[n].vinculo[2]= new
Nudo();this.nudo[n].vinculo[2]=this.nudo[n+1];//Or.x[n][n+1]=1;
  }
  //
   else if(n==this.nudo.length-nX && EleFinito=='t'){  // esquina
inferior izquierda, triÃ¡ngulo


this.nudo[n].tipoNudo='e';this.nudo[n].setNumBarrasS(2);this.nudo[n].v
inculo=new Nudo[this.nudo[n].getNumBarrasT()];
   this.nudo[n].vinculo[0]= new
Nudo();this.nudo[n].vinculo[0]=this.nudo[n-nX];//Or.x[n][n+1]=1;
```

```
   this.nudo[n].vinculo[1]= new
Nudo();this.nudo[n].vinculo[1]=this.nudo[n+1];//Or.x[n][n-nX]=1;
  }
    else if(n==0 && EleFinito=='t'){  //esquina origen,  triÃ¡ngulo


this.nudo[n].tipoNudo='e';this.nudo[n].setNumBarrasS(3);this.nudo[n].v
inculo=new Nudo[this.nudo[n].getNumBarrasT()];
   this.nudo[n].vinculo[0]= new
Nudo();this.nudo[n].vinculo[0]=this.nudo[n+1];//Or.x[n][n+1]=1;
   this.nudo[n].vinculo[1]= new
Nudo();this.nudo[n].vinculo[1]=this.nudo[n+nX+1];//Or.x[n][n-nX]=1;
   this.nudo[n].vinculo[2]= new
Nudo();this.nudo[n].vinculo[2]=this.nudo[n+nX];//Or.x[n][n-nX]=1;
  }

  else if(n==nX-1 && EleFinito=='t'){  //esquina superior derecha,
triÃ¡ngulo


this.nudo[n].tipoNudo='e';this.nudo[n].setNumBarrasS(2);this.nudo[n].v
inculo=new Nudo[this.nudo[n].getNumBarrasT()];
   this.nudo[n].vinculo[0]= new
Nudo();this.nudo[n].vinculo[0]=this.nudo[n-1];//Or.x[n][n+1]=1;
   this.nudo[n].vinculo[1]= new
Nudo();this.nudo[n].vinculo[1]=this.nudo[n+nX];//Or.x[n][n-nX]=1;
   }

      else if(n==this.nudo.length-1 && EleFinito=='t'){  //esquina
inferior derecha,  triÃ¡ngulo


this.nudo[n].tipoNudo='e';this.nudo[n].setNumBarrasS(3);this.nudo[n].v
inculo=new Nudo[this.nudo[n].getNumBarrasT()];
   this.nudo[n].vinculo[0]= new
Nudo();this.nudo[n].vinculo[0]=this.nudo[n-1];//Or.x[n][n+1]=1;
   this.nudo[n].vinculo[1]= new
Nudo();this.nudo[n].vinculo[1]=this.nudo[n-nX-1];//Or.x[n][n-nX]=1;
   this.nudo[n].vinculo[2]= new
Nudo();this.nudo[n].vinculo[2]=this.nudo[n-nX];//Or.x[n][n-nX]=1;
  }


 //
  else if(n==this.nudo.length-nX && EleFinito=='d'){  //esquina
origen, superior izquierda,  malla diagonada


this.nudo[n].tipoNudo='e';this.nudo[n].setNumBarrasS(3);this.nudo[n].v
inculo=new Nudo[this.nudo[n].getNumBarrasT()];
   this.nudo[n].vinculo[0]= new
Nudo();this.nudo[n].vinculo[0]=this.nudo[n+1];//Or.x[n][n+1]=1;
   this.nudo[n].vinculo[1]= new
Nudo();this.nudo[n].vinculo[1]=this.nudo[n-nX+1];//Or.x[n][n-nX+1]=1;
   this.nudo[n].vinculo[2]= new
Nudo();this.nudo[n].vinculo[2]=this.nudo[n-nX];//Or.x[n][n-nX]=1;
  }
  else if(n==this.nudo.length-1 && EleFinito=='d' ){  //esquina
origen, superior derecha,  malla diagonada
```

```
this.nudo[n].tipoNudo='e';this.nudo[n].setNumBarrasS(3);this.nudo[n].v
inculo=new Nudo[this.nudo[n].getNumBarrasT()];
   this.nudo[n].vinculo[0]= new
Nudo();this.nudo[n].vinculo[0]=this.nudo[n-1];//Or.x[n][n-1]=1;
   this.nudo[n].vinculo[1]= new
Nudo();this.nudo[n].vinculo[1]=this.nudo[n-nX];//Or.x[n][n-nX]=1;
   this.nudo[n].vinculo[2]= new
Nudo();this.nudo[n].vinculo[2]=this.nudo[n-nX-1];//Or.x[n][n-nX-1]=1;
  }
  else if(n==nX-1 && EleFinito=='d'){  //esquina origen, inferior
derecha,  malla diagonada


this.nudo[n].tipoNudo='e';this.nudo[n].setNumBarrasS(3);this.nudo[n].v
inculo=new Nudo[this.nudo[n].getNumBarrasT()];
   this.nudo[n].vinculo[0]= new
Nudo();this.nudo[n].vinculo[0]=this.nudo[n-1];//Or.x[n][n-1]=1;
   this.nudo[n].vinculo[1]= new
Nudo();this.nudo[n].vinculo[1]=this.nudo[n+nX-1];//Or.x[n][n+nX-1]=1;
   this.nudo[n].vinculo[2]= new
Nudo();this.nudo[n].vinculo[2]=this.nudo[n+nX];//Or.x[n][n+nX]=1;
  }

  else if(n==nX-1 && EleFinito=='c') //esquina inferiro derecha

{this.nudo[n].tipoNudo='e';this.nudo[n].setNumBarrasS(2);this.nudo[n].
vinculo=new Nudo[this.nudo[n].getNumBarrasT()];
   this.nudo[n].vinculo[0]= new
Nudo();this.nudo[n].vinculo[0]=this.nudo[n-1];//Or.x[n][n-1]=1;
   this.nudo[n].vinculo[1]= new
Nudo();this.nudo[n].vinculo[1]=this.nudo[n+nX];//Or.x[n][n+nX]=1;
  }
  else if(n==this.nudo.length-nX && EleFinito=='c') // esquina
superior izquierda

{this.nudo[n].tipoNudo='e';this.nudo[n].setNumBarrasS(2);this.nudo[n].
vinculo=new Nudo[this.nudo[n].getNumBarrasT()];
   this.nudo[n].vinculo[0]= new
Nudo();this.nudo[n].vinculo[0]=this.nudo[n+1];//Or.x[n][n+1]=1;
   this.nudo[n].vinculo[1]= new
Nudo();this.nudo[n].vinculo[1]=this.nudo[n-nX];//Or.x[n][n-nX]=1;
  }
  else if(n==this.nudo.length-1 && EleFinito=='c' ) //esquina superior
derecha

{this.nudo[n].tipoNudo='e';this.nudo[n].setNumBarrasS(2);this.nudo[n].
vinculo=new Nudo[this.nudo[n].getNumBarrasT()];
   this.nudo[n].vinculo[0]= new
Nudo();this.nudo[n].vinculo[0]=this.nudo[n-1];//Or.x[n][n-1]=1;
   this.nudo[n].vinculo[1]= new
Nudo();this.nudo[n].vinculo[1]=this.nudo[n-nX];//Or.x[n][n-nX]=1;
  }

  else {
 if(EleFinito=='t' ){ //punto generico del interior
  this.nudo[n].setNumBarrasS(6); this.nudo[n].vinculo=new
Nudo[this.nudo[n].getNumBarrasT()];
  this.nudo[n].vinculo[0]= new Nudo(); this.nudo[n].vinculo[0]=
this.nudo[n-1];//Or.x[n][n-1]=1;
```

```
  this.nudo[n].vinculo[1]= new Nudo(); this.nudo[n].vinculo[1]=
this.nudo[n-nX-1];//Or.x[n][n+nX]=1;
    this.nudo[n].vinculo[2]= new Nudo(); this.nudo[n].vinculo[2]=
this.nudo[n-nX];//Or.x[n][n+nX]=1;
  this.nudo[n].vinculo[3]= new Nudo();
this.nudo[n].vinculo[3]=this.nudo[n+1];//Or.x[n][n+1]=1;
  this.nudo[n].vinculo[4]= new Nudo();
this.nudo[n].vinculo[4]=this.nudo[n+nX+1];//Or.x[n][n-nX]=1;
    this.nudo[n].vinculo[5]= new Nudo();
this.nudo[n].vinculo[5]=this.nudo[n+nX];//Or.x[n][n-nX]=1;
                              }


if(EleFinito=='c' ){
  this.nudo[n].setNumBarrasS(4); this.nudo[n].vinculo=new
Nudo[this.nudo[n].getNumBarrasT()];
  this.nudo[n].vinculo[0]= new Nudo(); this.nudo[n].vinculo[0]=
this.nudo[n-1];//Or.x[n][n-1]=1;
  this.nudo[n].vinculo[1]= new Nudo(); this.nudo[n].vinculo[1]=
this.nudo[n+nX];//Or.x[n][n+nX]=1;
  this.nudo[n].vinculo[2]= new Nudo();
this.nudo[n].vinculo[2]=this.nudo[n+1];//Or.x[n][n+1]=1;
  this.nudo[n].vinculo[3]= new Nudo();
this.nudo[n].vinculo[3]=this.nudo[n-nX];//Or.x[n][n-nX]=1;
                              }
  if(EleFinito=='d' ){
  this.nudo[n].setNumBarrasS(8); this.nudo[n].vinculo=new
Nudo[this.nudo[n].getNumBarrasT()];
  this.nudo[n].vinculo[0]= new Nudo(); this.nudo[n].vinculo[0]=
this.nudo[n-1];//Or.x[n][n-1]=1;
  this.nudo[n].vinculo[1]= new Nudo(); this.nudo[n].vinculo[1]=
this.nudo[n+nX-1];//Or.x[n][n+nX-1]=1;
  this.nudo[n].vinculo[2]= new Nudo();
this.nudo[n].vinculo[2]=this.nudo[n+nX];//Or.x[n][n+nX]=1;
  this.nudo[n].vinculo[3]= new Nudo();
this.nudo[n].vinculo[3]=this.nudo[n+nX+1];//Or.x[n][n+nX+1]=1;
  this.nudo[n].vinculo[4]= new Nudo();
this.nudo[n].vinculo[4]=this.nudo[n+1];//Or.x[n][n+1]=1;
  this.nudo[n].vinculo[5]= new Nudo();
this.nudo[n].vinculo[5]=this.nudo[n-nX+1];//Or.x[n][n-nX+1]=1;
  this.nudo[n].vinculo[6]= new Nudo();
this.nudo[n].vinculo[6]=this.nudo[n-nX];//Or.x[n][n-nX]=1;
  this.nudo[n].vinculo[7]= new Nudo();
this.nudo[n].vinculo[7]=this.nudo[n-nX-1];//Or.x[n][n-nX-1]=1;
  }
  this.nudo[n].tipoNudo='s';
  }////
  String eti= new String();
   eti="   ";
  char va='O'; if(this.nudo[n].apoyo)va='A';else va='O';
  char ex='I'; if(this.nudo[n].externo)ex='X';else ex='I';
  int pos= nudo[n].getOrden();
  String sOrden =String.format("%04d", pos);

eti="N"+String.valueOf(va)+sOrden+String.valueOf(ex)+String.valueOf(th
is.nudo[pos].getTipoNudo())+String.valueOf(this.nudo[pos].getTipoEleFi
nito());
//eti=String.valueOf(va)+String.valueOf(ex)+String.valueOf(pos)+String
.valueOf(this.nudo[pos].getTipoNudo())+String.valueOf(this.nudo[pos].g
etTipoEleFinito())+String.valueOf(this.nudo[pos].getOrden());
  this.nudo[n].setEtiqueta(eti);
```

```
//return this.nudo[n].nBarrasS ;
//  for(int
k=this.nudo[n].getNumBarrasS();k<nudo[n].vinculo.length;k++){this.nudo
[n].vinculo[k]=new Nudo();}//21012013
  return this.nudo[n].getNumBarrasT() ;
  }

/*////////////////////////////////////////////////////////FINAL///
//////////////////////////////////////////////////////////*/

Matriz OrdenarMatrizNudos(Nudo[] nu){
int Fil=nu.length;
int Col=nu.length;
Matriz MaOr=new Matriz(Fil, Col);
Nudo t=new Nudo();
for( int i=0; i < Fil; i++){//ordena la matriz de abajo hacia arriba
for(int x=0; x < Fil; x++){
if(nu[i].etiqueta.compareTo(nu[x].etiqueta)<0){
 t = nu[i];
 nu[i]=nu[x];
 nu[x] = t;
}
}}
for(int h=0;h<Fil;h++){for(int
v=0;v<Fil;v++){MaOr.x[h][v]=0;}}//iniciamos todos los elementos=0;
for(int j=0;j<Fil;j++){for(int
v=0;v<nu[j].getNumBarrasT();v++){for(int c=0;c<Fil;c++){
    if (nu[j].vinculo[v]==nu[c]){ MaOr.x[j][c]=1;} else{
MaOr.x[j][c]=0;
    }}}}

return MaOr;

}



Nudo[] OrdenarNudos(Nudo[] nu){
int K=nu.length; //dimensiones del array de entrada que debe ser igual
que la del array de salida.

Nudo[] salida;
salida=new Nudo[K];
Nudo t=new Nudo(); // me hace falta una variable temporal para
realizar la comparaciÃ³an delante versus detras
//ordenamiento por tipo de nudo
for( int i=0; i < nu.length; i++){//ordena la matriz de abajo hacia
arriba
for(int x=0; x < nu.length; x++){
if(nu[i].etiqueta.compareTo(nu[x].etiqueta)<0){
 t = nu[i];
 nu[i]=nu[x];
 nu[x] = t;
}
}}

for(int i=0;i<nu.length;i++){
salida[i]= nu[i];
nu[i].setOrdenIndice(i);}
return salida;
}
```

```
protected void finaliza(Nudo[] nd){ for(int n=0;n<nd.length;n++)
{nd[n].finaleze();}  nd=null;}
/////////////////Esta funcion devuelbe una arry con barras diferentes
en ambos
senidos//////////////////////////////////////////////////////////

Matriz AsignaMatrizDensidad(Barra[] bar, Matriz Mat ){
    int n=0;
    for(int h=0;h<bar.length;h++){
    n=h;
bar[n].setOrdenMatrizDensidad(OrdenBarraMatrizDensidad( bar[n], Mat)
);

}

Matriz MB=new Matriz(this.getNumBarras(), this.getNumBarras());
  MB.x=new double[this.getNumBarras()][this.getNumBarras()];
 for(int i=0;i<this.getNumBarras();i++){ for(int
j=0;j<this.getNumBarras();j++){
         MB.setElemento(i, j, 0);
       }}
for(int i=0;i<this.getNumBarras();i++){
    int p=bar[i].getOrdenMatrizDensidad();
   MB.setElemento( p,p,bar[i].getDensiFuerzaF());
}

return MB;
}
//////////////////////////////////////////////////////
Matriz _matrizDensidad(Barra[] bar, Matriz Mat){
int a=0,b=0, salida=0;
int filas=bar.length; //que coincidira con
lienzo.superficie.gerNumeroBarras();
Matriz Den=new Matriz(filas, filas);
Den.x=new double[this.getNumBarras()][filas];
 for(int i=0;i<filas;i++){ for(int j=0;j<filas;j++){ //el numero de
filas y columnas es el mismo
        Den.setElemento(i, j, 0);  //se le asigna a todos los valores
cero
   }}
 if(filas==Mat.getNumFilas()){  //se hace una comprobaciÃ³n
 for(int i=0;i<filas;++i){ //barrido por el array de barras
     a=bar[i].getOrigen().getOrden(); //se localiza el orden del punto
que origina la barra segun el orden de la matriz de nudos ordenados
     b=bar[i].getFin().getOrden();   //se localiza el orden del
elemento que finaliza la barra segÃºn su posiciÃ³n en el array de
nudos ordenado
  //  a=bar[i].getOrigen().getOrden();

     for(int j=0;j<filas;++j){ //barrido por las fila de la  matiz Mat

if (Mat.getElemento(j, a)==1){if (Mat.getElemento(j, b)==-1){
//sustitui el && por el doble bucle
    bar[i].setOrdenMatrizDensidad(j);
Den.setElemento(bar[i].getOrdenMatrizDensidad(),bar[i].getOrdenMatrizD
ensidad(), bar[i].getDensiFuerzaF());}
                                 }
                                }
}} else{ //en caso que de error se asigna el valor de densidad por
defecto
```

```
if(Nodos.bAlerta) JOptionPane.showMessageDialog(  null, "Error en el
cÃ¡lculo de la matriz de Densidades de Fuerza. Se asignara una
densidad por defecto para todas las barras");
     for(int i=0;i<filas;i++){ Den.setElemento(i, i,
this.densiFuerzaBarraForma   );
  }}
 return Den;
}
//////////////////////////////////////


int OrdenBarraMatrizDensidad(Barra bar, Matriz Mat){
int salida=0;
int a=bar.getOrdenOrigen(); //como digo yo
int b=bar.getOrdenFin(); //como digo yo
for(int j=0;j<Mat.getNumFilas();j++){
    if ((Mat.getElemento(j, a)==1 && Mat.getElemento(j, b)==-
1)){{salida=j; }}
}

return salida;
}

// funciÃ³n que asigna las masas por las barras a la matriz nudo.
bCarga indica si se suma o no (-)cargaZ/g
Nudo[] masaBarraToNudo(Barra[] bar, Nudo[] nud){
Nudo[] nd = new Nudo[nud.length];
nd=nud.clone();
//la primera vez limpiamos las masas y asignamos la mitad de la masa
de la barra al origen.
//la segunda vez no se limpia y asignamos la mitas de la masa al fin
//la tercera vez le asignamos, segun bCarga, la masa de carga Z
for (int
b=0;b<bar.length;b++){bar[b].origen.setMasa(0);bar[b].fin.setMasa(0);}
for (int n=0;n<nd.length;n++){nd[n].setMasa(0);}
for (int b=0;b<bar.length;b++){
 double ma = bar[b].origen.getMasa(false);
bar[b].origen.setMasa(ma+bar[b].getMasa()/2);
}
for (int b=0;b<bar.length;b++){
double ma = bar[b].fin.getMasa(false);
bar[b].fin.setMasa( ma+bar[b].getMasa()/2);
}

return nd;}

double escalaAmplitud(Barra[] bar,int modo, Matriz al ){
double cociente=0;
double temp=0;
for(int b=0;b<bar.length;b++){
if(bar[b].acortaInicial!=0)
cociente=al.getElemento(b,modo)/bar[b].acortaInicial;else
JOptionPane.showMessageDialog(  null, "DivisiÃ³n por cero: la barra no
se ha acrotado");
cociente=Math.abs(cociente);
if (cociente>temp)temp=cociente;
}
return temp;
}
double amplitudMax(Matriz av,int modo, double factor){
double maximo=0;
```

```
double cociente=0;
double temp=0;
for(int b=0;b<av.getNumFilas();b++){
maximo=av.getElemento(b,modo);
maximo=Math.abs(maximo);
if (maximo>temp)temp=maximo;
}
if(factor!=0) cociente=temp/factor; else
JOptionPane.showMessageDialog(  null, "DivisiÃ³n por cero: factor de
amplitud es nulo");
return cociente;
}
////////////////////////////////////////////////////////////////////
//////////////////////
 Nudo[] restaura(){
   Nodos.lienzo.boolBarras=true;
   Nudo[] nuVtemp; //= new Nudo[this.nuSalida.length];
   nuVtemp=null;
    nuVtemp=this.nuSalida.clone();
    int nA=this.numApoyos;
  // Nodos.lienzo.boolVibra=false;
   int  numNuTotal= this.nudo.length;
  for (int k=0;k< nA;k++)

for(int a=0;a<nA;a++){
nuVtemp[a].setX(this.nuSalida[a].getX());
nuVtemp[a].setY(this.nuSalida[a].getY());
nuVtemp[a].setZ(this.nuSalida[a].getZ());
}

  for (int k=nA;k<numNuTotal;k++)
  {this.nuSalida[k].setX(this.FormaOut.x[k][0]);
   this.nuSalida[k].setY(this.FormaOut.x[k][1]);
   this.nuSalida[k].setZ(this.FormaOut.x[k][2]);
  }
 return nuVtemp;
}
////////////////////////////////////////////////////////////////////
/////////////
///////DEFORMA/////////DEFORMA////////////DEFORMA/////////////////
////////////////////////////////
 Nudo[] deforma(int mod,double escV, double escX, double escY, double
escZ){

Nudo[] nuVtemp;//=new Nudo[this.nudo.length];
nuVtemp=null;
nuVtemp=this.nuSalida.clone();
           int nA=this.numApoyos;
for(int a=0;a<nA;a++){
nuVtemp[a].setX(this.nuSalida[a].getX());
nuVtemp[a].setY(this.nuSalida[a].getY());
nuVtemp[a].setZ(this.nuSalida[a].getZ());
}
for(int i=nA;i<this.nuSalida.length;i++){
nuVtemp[i].setX(nuVtemp[i].getX()+escV*escX*this.VP.getElemento(i-nA,
mod));
nuVtemp[i].setY(nuVtemp[i].getY()+escV*escY*this.VP.getElemento(i-nA,
mod));
nuVtemp[i].setZ(nuVtemp[i].getZ()+escV*escZ*this.VP.getElemento(i-nA,
mod));
}
```

```
//for(int n=0;n<this.nudo.length;n++){ this.vibra[n].equals(tempV[n]);
}
 return nuVtemp;
  }
////////REACCIONES
///////REACCIONES///////REACCIONES/////////////////////////////////
/////////////////////////////
void reacciones(){
int nN=this.nudo.length;
Matriz Mcoord= new Matriz(nN,3);
Nudo[] coorXYZ = this.nuSalida.clone();
coorXYZ=this.OrdenarNudos(coorXYZ);
for(int i=0;i<nN;i++){
Mcoord.setElemento(i, 0,coorXYZ[i].getX() );
Mcoord.setElemento(i, 1,coorXYZ[i].getY() );
Mcoord.setElemento(i, 2,coorXYZ[i].getZ() );
}
Matriz tempReac = new Matriz(nN,3);
tempReac=tempReac.producto( this.ME,  Mcoord);
this.aReac= new Punto[this.numApoyos];
 for(int r=0;r<this.numApoyos;r++){
 this.aReac[r]=new Punto();
this.aReac[r].X= tempReac.getElemento(r,0);
this.aReac[r].Y= tempReac.getElemento(r,1);
this.aReac[r].Z= tempReac.getElemento(r,2);
}
}
//////////////////////////ASIGNACION DE
ELEMENETOSFINITOS/////////////////////////////////////////////////////
/
void asignaNudosEleFinitos(char tipoMall, int nX, int nY){
 int nEleFinito=0;
 int n=0;
 int k=0;
 Barra ladoA,ladoB;
 JuVector vectorTemporal=new JuVector(0,0,0);
 if(tipoMall=='t'){
 nEleFinito=2*(nX-1)*(nY-1);//el nÃºmero de elementos finitos es el
doble que en casos rectangulares
 this.eleFinito=new EleFinito[nEleFinito];
 k=0;
for(int i=0;i<nX-1; i++){ //barrido por filas de nudos de la
superficie  hasta la penultima
  for(int j=0;j<nY-1; j++){ //barrido por columnas de nudos de la
superficie hasta la penultima
      n=j+i*nY; //cada rectÃ¡nfulo tiene dos elementos triangulares
     //contador del elemento finito
  ////////////////////////////////////////////k
//////////////////////////////
   this.eleFinito[k]=new EleFinito('t');
  // this.eleFinito[k].arista=new Barra[3];
 //  this.eleFinito[k].nudo= new Nudo[3]; //el orden de los nudos es
el sentido antihorario empezando por las menos cuarto
   this.eleFinito[k].nudo[0]=new
Nudo();this.eleFinito[k].nudo[0]=this.nudo[n];
   this.eleFinito[k].nudo[1]=new
Nudo();this.eleFinito[k].nudo[1]=this.nudo[n+nY];
   this.eleFinito[k].nudo[2]=new
Nudo();this.eleFinito[k].nudo[2]=this.nudo[n+nY+1];
    this.eleFinito[k].arista[0]=new
Barra(this.eleFinito[k].nudo[0],this.eleFinito[k].nudo[1]);
```

```
     this.eleFinito[k].arista[1]=new
Barra(this.eleFinito[k].nudo[1],this.eleFinito[k].nudo[2]);
      this.eleFinito[k].arista[2]=new
Barra(this.eleFinito[k].nudo[2],this.eleFinito[k].nudo[0]);
   this.eleFinito[k].setOrden(k);
   this.eleFinito[k].setTipo(tipoMall);

this.eleFinito[k].setEtiqueta("EF"+String.valueOf(tipoMall)+String.val
ueOf(this.eleFinito[k].getOrden()));

//this.eleFinito[k].actualizar('t');
//////////////////////////////////////////k+1
//////////////////////////////////////
   this.eleFinito[k+1]=new EleFinito('t');
 //  this.eleFinito[k+1].arista=new Barra[3];
 //   this.eleFinito[k+1].nudo= new Nudo[3]; //el orden de los nudos
es el sentido horario empezando por las menos cuarto
   this.eleFinito[k+1].nudo[0]=new
Nudo();this.eleFinito[k+1].nudo[0]=this.nudo[n];
   this.eleFinito[k+1].nudo[1]=new
Nudo();this.eleFinito[k+1].nudo[1]=this.nudo[n+1];
   this.eleFinito[k+1].nudo[2]=new
Nudo();this.eleFinito[k+1].nudo[2]=this.nudo[n+nY+1];
   this.eleFinito[k+1].arista[0]=new
Barra(this.eleFinito[k+1].nudo[0],this.eleFinito[k+1].nudo[1]);
   this.eleFinito[k+1].arista[1]=new
Barra(this.eleFinito[k+1].nudo[1],this.eleFinito[k+1].nudo[2]);
   this.eleFinito[k+1].arista[2]=new
Barra(this.eleFinito[k+1].nudo[2],this.eleFinito[k+1].nudo[0]);
k=k+2;
  }}
////////////////////////////elementos finitos rectangulares
}

//////////////////////////////////////////////////////////////////////
//////
 if( tipoMall=='c'){
 nEleFinito=((nX-1)*(nY-1));
 this.eleFinito=new EleFinito[nEleFinito];
 k=0;
for(int i=0;i<nX-1; i++){ //barrido por filas de nudos de la
superficie  hasta la penultima
  for(int j=0;j<nY-1; j++){ //barrido por columnas de nudos de la
superficie hasta la penultima
      n=j+i*nY;
     //contador del elemento finito
this.eleFinito[k]=new EleFinito();
this.eleFinito[k].arista=new Barra[4];

this.eleFinito[k].nudo= new Nudo[4]; //el orden de los nudos es el
sentido horario empezando por las menos cuarto
this.eleFinito[k].nudo[0]=new
Nudo();this.eleFinito[k].nudo[0]=this.nudo[n];
this.eleFinito[k].nudo[1]=new
Nudo();this.eleFinito[k].nudo[1]=this.nudo[n+nY];
this.eleFinito[k].nudo[2]=new
Nudo();this.eleFinito[k].nudo[2]=this.nudo[n+nY+1];
this.eleFinito[k].nudo[3]=new
Nudo();this.eleFinito[k].nudo[3]=this.nudo[n+1];
this.eleFinito[k].arista[0]=new
Barra(this.eleFinito[k].nudo[0],this.eleFinito[k].nudo[1]);
```

```
this.eleFinito[k].arista[1]=new
Barra(this.eleFinito[k].nudo[1],this.eleFinito[k].nudo[2]);
this.eleFinito[k].arista[2]=new
Barra(this.eleFinito[k].nudo[2],this.eleFinito[k].nudo[3]);
this.eleFinito[k].arista[3]=new
Barra(this.eleFinito[k].nudo[3],this.eleFinito[k].nudo[0]);
this.eleFinito[k].setOrden(k);
this.eleFinito[k].setTipo(tipoMall);

this.eleFinito[k].setEtiqueta("EF"+String.valueOf(tipoMall)+String.val
ueOf(this.eleFinito[k].getOrden()));

//this.eleFinito[k].actualizar(tipoMall);
k++;

  }}
}

////////////////////////////////////////////////////////////////////
//
if( tipoMall=='d'){
 nEleFinito=((nX-1)*(nY-1));
 this.eleFinito=new EleFinito[nEleFinito];
 k=0;
for(int i=0;i<nX-1; i++){ //barrido por filas de nudos de la
superficie  hasta la penultima
  for(int j=0;j<nY-1; j++){ //barrido por columnas de nudos de la
superficie hasta la penultima
      n=j+i*nY;
     //contador del elemento finito
this.eleFinito[k]=new EleFinito();
this.eleFinito[k].arista=new Barra[6];
this.eleFinito[k].nudo= new Nudo[4]; //el orden de los nudos es el
sentido horario empezando por las menos cuarto
this.eleFinito[k].nudo[0]=new
Nudo();this.eleFinito[k].nudo[0]=this.nudo[n];
this.eleFinito[k].nudo[1]=new
Nudo();this.eleFinito[k].nudo[1]=this.nudo[n+nY];
this.eleFinito[k].nudo[2]=new
Nudo();this.eleFinito[k].nudo[2]=this.nudo[n+nY+1];
this.eleFinito[k].nudo[3]=new
Nudo();this.eleFinito[k].nudo[3]=this.nudo[n+1];
this.eleFinito[k].arista[0]=new
Barra(this.eleFinito[k].nudo[0],this.eleFinito[k].nudo[1]);
this.eleFinito[k].arista[1]=new
Barra(this.eleFinito[k].nudo[1],this.eleFinito[k].nudo[2]);
this.eleFinito[k].arista[2]=new
Barra(this.eleFinito[k].nudo[2],this.eleFinito[k].nudo[3]);
this.eleFinito[k].arista[3]=new
Barra(this.eleFinito[k].nudo[3],this.eleFinito[k].nudo[0]);
this.eleFinito[k].arista[4]=new
Barra(this.eleFinito[k].nudo[0],this.eleFinito[k].nudo[2]);
this.eleFinito[k].arista[5]=new
Barra(this.eleFinito[k].nudo[1],this.eleFinito[k].nudo[3]);
this.eleFinito[k].setOrden(k);
this.eleFinito[k].setTipo(tipoMall);

this.eleFinito[k].setEtiqueta("EF"+String.valueOf(tipoMall)+String.val
ueOf(this.eleFinito[k].getOrden()));
//this.eleFinito[k].actualizar(tipoMall);
k++;
```

```
  }}
}
//ahora introducimios las superficies iniciales llamando a la funciÃ³n
actualizar
for(int i=0;i<this.eleFinito.length;i++){
   this.eleFinito[i].setGrosor(this.grosorEF);
   this.eleFinito[i].setDensiMasa(this.densiEF);
   this.eleFinito[i].centroGeometrico=new Punto();
   this.eleFinito[i].carga=new JuVector();
   this.eleFinito[i].peso=new JuVector(0,0,0);
   this.eleFinito[i].superficieA=new JuVector(0,0,0);
   this.eleFinito[i].superficieB=new JuVector(0,0,0);
   this.eleFinito[i].peso=new JuVector(0,0,0);
this.eleFinito[i].actualizar();
this.eleFinito[i].setOrden(i);
}

////////////////////////////////////////////////////////////////////
///
}
public int  mostrarParametroEF(int k){
   // Nodos.lienzo.superficie.eleFinito[k].actualizar();
    this.eleFinito[k].actualizar();
  // int t= Nodos.lienzo.superficie.eleFinito[k].getOrden();
     int t= this.eleFinito[k].getOrden();
   // int t=k;
          //  Nodos.lienzo.superficie.eleFinito[t].actualizar();
           tNumEF.setText(String.valueOf(t));

Paneli.tEFEtiqueta.setText(Nodos.lienzo.superficie.eleFinito[t].getEti
queta());

Paneli.tEFDensiMasa.setText(String.valueOf(Nodos.lienzo.superficie.ele
Finito[t].densiMasa));

Paneli.tEFGrosor.setText(String.valueOf(Nodos.lienzo.superficie.eleFin
ito[t].getGrosor()));

Paneli.tEFSuperficie.setText(String.valueOf(0.01*Math.round(100*Nodos.
lienzo.superficie.eleFinito[t].area)));

Paneli.tEFMasa.setText(String.valueOf(Nodos.lienzo.superficie.eleFinit
o[t].masa));
           Nodos.lienzo.nEFActiva=t;
           Nodos.lienzo.repaint();
            return t;
}


}
```

Clase Ventana

```
/**
 *
 * @author julio MuÃ±iz Padilla y Erensto MuÃ±iz MartÃn
 */
package nodos;


import javax.swing.JFrame;
import java.awt.event.*;
import java.io.Serializable;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JLabel;
import java.awt.Image;
import java.awt.Toolkit;


public class Ventana extends JFrame implements Serializable {
private String aviso=" ";
 Ventana(String titulo, int Ancho, int Alto,boolean bEnd){
 super();
 this.setIconImage(new
javax.swing.ImageIcon(getClass().getResource("imagenes/nodos2.png")).g
etImage());


  this.setTitle(titulo);
 this.setSize(Ancho,Alto);
  if(bEnd) {
       this.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e){System.exit(0);}}
             );  }
 }
 Ventana(String titulo, int Ancho, int Alto, Paneli panel,boolean
bEnd){
  panel.setSize(Ancho-2,Alto-2);
 add(panel);
 this.setTitle(titulo); this.setSize(Ancho,Alto);

 if(bEnd) {
       this.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e){System.exit(0);}}
             );  }

 this.setIconImage(new
javax.swing.ImageIcon(getClass().getResource("imagenes/nodos2.png")).g
etImage());


 }


}
```

Clase Vinculo

```
/**
 *
 * @author Julio MuÃ±iz Padilla, Erensto MuÃ±iz MartÃn
   2014
 */
package nodos;

import java.io.Serializable;


public class Vinculo implements Serializable   {
int orden;
int nudoOrigen;
int nudoFinal;
char tipoNudoOrigen='x';
char tipoNudoFinal;
//public Nudo nuExterno;
Barra barra;
JuVector vector;
String etiqueta;

Vinculo(){
barra = new Barra(null,null);
vector = new JuVector(3);

}

}
```

Anexo II- Bibliografía

Bibliografía.

Benavent-Climent, A. (2010). *Estructuras Sismorresistentes*. Madrid: Maia Editores.

Chopra, A.K. (2007). *Dynamics of Structures*. *Theory and Applications to Earthquake Engineering (3ª Ed.* Upper Saddle River, New Jersey, NJ: Prentice Hall.

Colegio de Ingenieros de Caminos Canales y Puertos y Ache Geo-ATEP (2001). *Problemas de vibraciones en estructuras*. Madrid: Colegio de Ingenieros de Caminos Canales y Puertos.

Foster, B. y Mollaert, M. (2009). Arquitectura Textil. Guía Europea de Diseño de Estructuras Superficiales Tensadas. Madrid: Munilla-Leria.

Frei, O. (1962). *Cubiertas Colgantes*. Barcelona: Labor.

Gimsing, N.J. y Gergakis,C.T. (2012). *Cable Supported Bridges: Concept and Desing* (3ª Ed). New York, NY: John Wiley &Sons.

Hernandez-Montes, E., Jurado-Piña, R. y Bayo, E. (2006). *Topological Mapping for Tension Structures. Journal of Structural Engineering*. 132 (6), 970-977.

Holzner S. (2000). *La Biblia de Java 2*. Madrid: Anaya.

Inman, D.J. (2006). *Vibration with Control*. London: John Wiley &Sons.

Leonard, J.W. (1988). *Tension Structures: Behaviour and analysis*. New York, NY: Mc Graw-Hill.

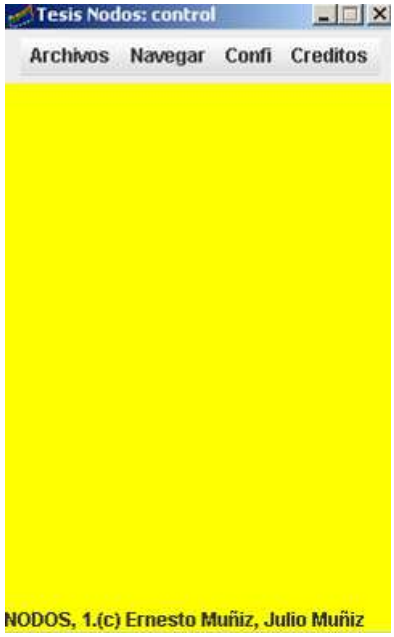
Majowiecki, M. (1994). *Tensostrutture: Progetto e Verifica*. Genova, Italia: Crea Edizioni.

Seidel, M. (2007). *Tensile Surface Structure: A Practical Guide to Cable and Membrane Construction*. Berlin: Ernest & Sohn.

Anexo III- Manual del programa Nodos

1- Menú desplegable control:

Al arrancarse el programa Nodos la pantalla que inmediatamente visualizamos es el menú de **control**, que contiene los desplegables Archivos, Navegar, Confi y Créditos.

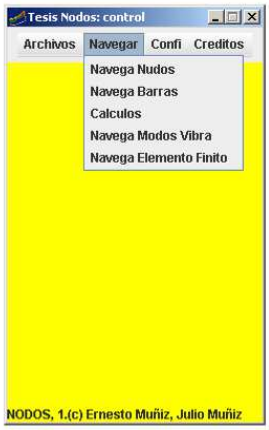


Este menú es siempre accesible cuando el programa está en ejecución, y deberemos dirigirnos al mismo cuando necesitemos **realizar o recuperar** alguna actividad del programa que en ese momento no esté visualizada en pantalla. Los contenidos del menú contextual son:

Archivos:

Donde se realiza la gestión a nivel de archivos de trabajo (la exportación DXF aún no está habilitada) así como la inicialización de una nueva estructura. Más delante de desarrollarán los contenidos pertinentes al inicio de un nuevo proyecto.

Navegar:



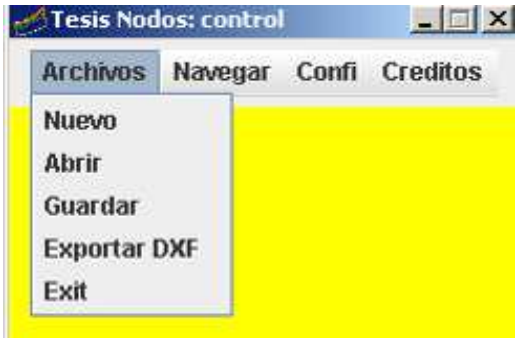
Este desplegable es quizá el de mayor importancia para el manejo del programa, dado que desde él se pueden llamar a los “motores” del programa que permiten navegar por la estructura analizada en el caso que no estén en ese momento visualizándose o en la barra de ejecución al pie de la pantalla. Más adelante se expondrán los contenidos distintos navegadores.

Confi: Opción aún no habilitada del programa cuyo objeto será el guardar las preferencias de ejecución: los valores de inicialización de parámetros, etc.

Créditos: Información sobre los autores del programa.

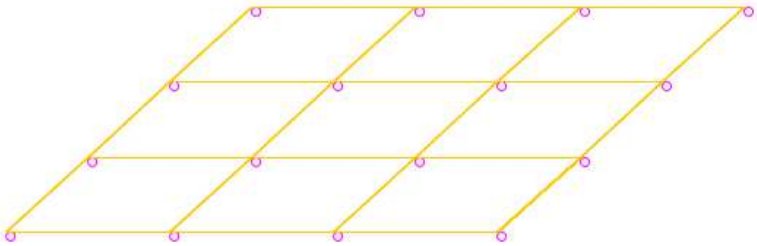
1.1 Menú desplegable Control/Archivos/Nuevos

Dentro del menú desplegable Archivos tenemos como primera opción “nuevo”



Al pinchar sobre “Nuevo” se activa el siguiente desplegable, cuyas casillas contienen una serie de valores dispuestos por defecto pero que el usuario deberá ajustar a sus necesidades conforme al proyecto:

A la hora de comprender dichos valores hemos de entender que por defecto el sistema parte de una “malla” rectangular plana en “X,Y”



Dicha malla se muestra en perspectiva “caballera”, siendo el plano de fondo el constituido por los ejes “Y,Z” quedando en perspectiva el eje “X”.

Los valores de partida que requiere el programa a la hora de definir una estructura nueva son:

- Ancho del Rectángulo: dimensión en metros de la malla de partida en la dirección “X” (es decir en perspectiva)

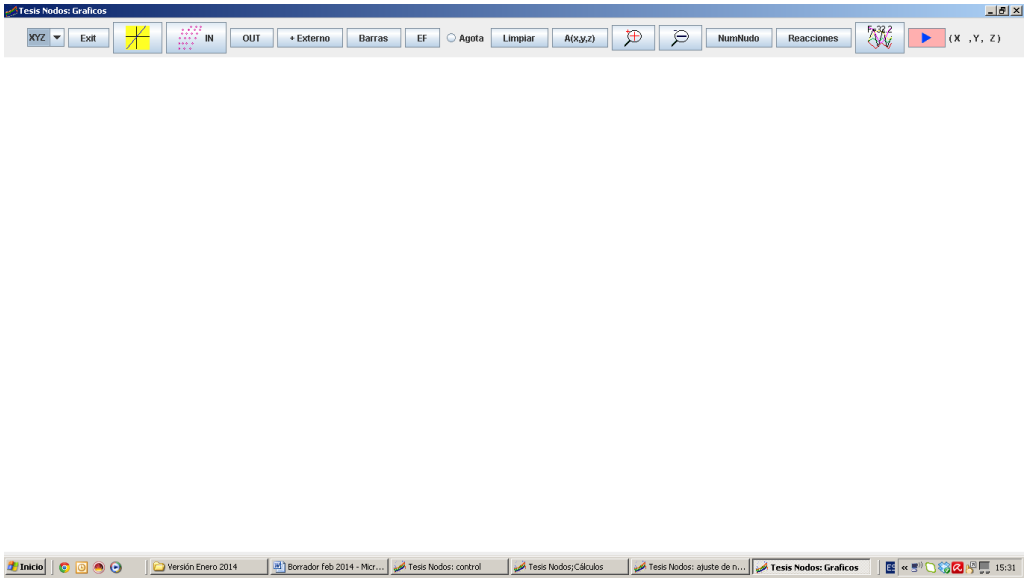
- Alto del Rectángulo: dimensión en metros de la malla de partida en la dirección "Y"
- Tipo de elemento finito: Rectángulo, Rectángulo Diagonado, Triángulo. Todas las formas parten de una malla basada en rectángulos si bien el rectángulo diagonado es aquel que contiene dos barras en su interior conformando sus diagonales y el triángulo es aquel caso en que el elemento sólo contiene una diagonal (no activado). Como es sabido la forma de partida de una malla es intrascendente, lo importante es cómo se establecen los vínculos entre nudos.
- Grosor de elemento finito: Representa el espesor de la malla.
- Densidad General de Fuerza: Valor inicial de la Densidad de Fuerza con el que las barras son inicializadas. Es un parámetro importante dado que guarda relación con el grado de pretensado del sistema.
- Sección barra: Es el valor con el que por defecto se inicializan todas las barras
- Densidad EF: Es el valor de densidad inicial con el que se calcularán las masas del sistema
- Material: Es el valor con el que por defecto se inicializarán las barras del sistema.

Finalmente hay dos botones "inicia" y "actualizar". El primero se emplea cuando se comienza un problema nuevo -con los datos que haya introducido el usuario según se muestran en pantalla- y el segundo es cuando se desea solamente actualizar algún parámetro. (En desarrollo)

2- Pantalla Gráficos:

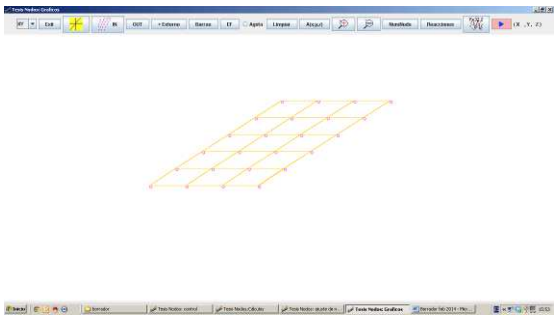
Cuando se inicia/carga un archivo nuevo/existente aparece inmediatamente la pantalla "Gráficos". Normalmente aparecerá "en blanco" la parte principal de la imagen, mostrando una barra de iconos superior y una serie de ventanas emergentes que podrán estar bien desplegadas (normalmente aparece activado el navegador de cálculos) o bien minimizadas en el pie de la pantalla

Pantalla Gráficos:



Véase en la parte superior los iconos propios de esta pantalla y al pie de la pantalla las ventanas emergentes de los diferentes navegadores, que en este caso están replegadas.

PARA VISUALIZAR LA MALLA INTRODUCIDA AL INICIAR/ABRIR UNA ESTRUCTURA DEBERÁ HACERSE “DOBLE CLICK” CON EL RATON EN LA PANTALLA EN BLANCO.



En ese momento se visualizará la estructura, pudiéndose actuar sobre la imagen mostrada mediante el uso de los iconos de la parte superior de la pantalla:

El menú que aparece en la anterior imagen desplegado pre-establece una serie de visualizaciones, o bien “Tridimensional” (X,Y,Z), o bien “Aplanando” alguno de los ejes.

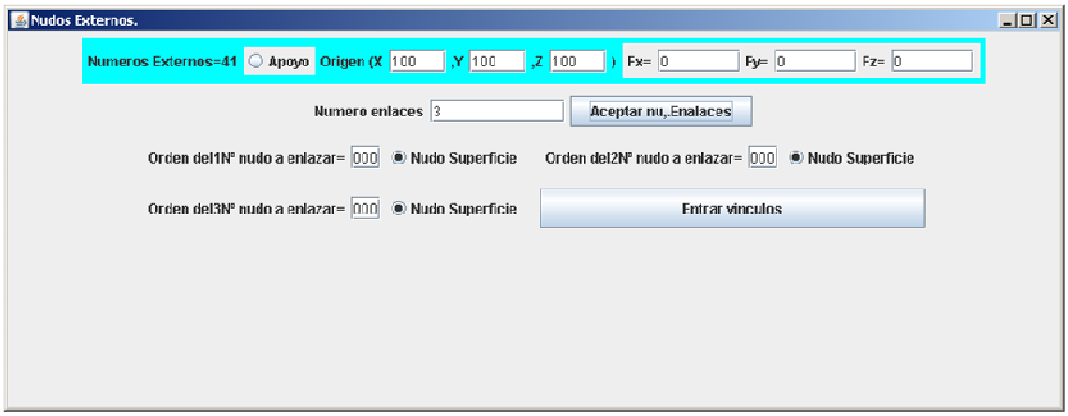
IMPORTANTE: Véase que presionando los botones del ratón y moviendo este por la pantalla la figura mostrada cambia de perspectiva, siempre partiendo de una caballera (tridimensional o con un eje aplanado). El manejo de las perspectivas requiere algo de práctica –no es sencillo- si bien recomendamos el mantenerse normalmente en "X,Y,Z" hasta que se tenga práctica.

Junto al anterior menú vemos el botón de "Exit". Dicho botón se emplea para abandonar el programa de forma inmediata.

Vemos con fondo amarillo el botón de "Ejes": Nos representa, -en la perspectiva caballera según la visualización "X,Y,Z"- unos ejes "Y,Z" que pasan por el origen de coordenadas. Se recomienda su uso conjuntamente con el manejo dinámico de las perspectivas con el ratón, dado que el origen de rotaciones coincide con el de coordenadas por lo que facilita la orientación de la perspectiva el ubicar el origen de coordenadas.

El botón "In" y "Out" introduce (o elimina) en la imagen la visualización de los nudos.

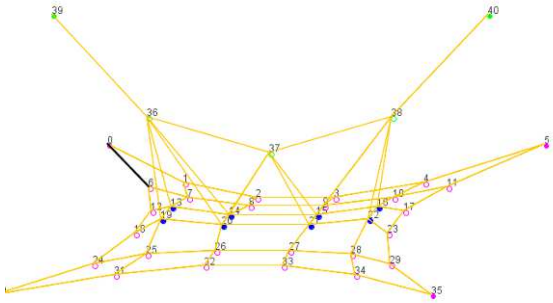
El Botón +Externo INTRODUCE UN NUDO EXTERNO A LA MALLA.



Cuando dicho botón se presiona aparece un display emergente que se va introduciendo poco a poco (arriba aparece en su totalidad). En él se establece si el nudo externo es un apoyo o el libre, sus coordenadas (lo cual es importante en el caso de que sea apoyo), qué cargas actúan sobre dicho nudo y se establece con cuántos

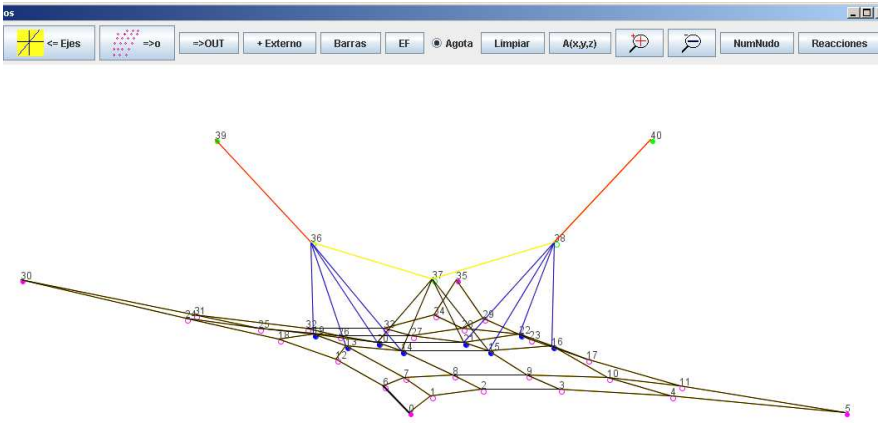
nudos se vincula dicho nudo EN EL MOMENTO DE INTRODUCCIÓN DEL MISMO así como cuales son dichos nudos con los que se relaciona. Una vez creado el nudo siempre se podrán modificar sus características actuando en el navegador de nudos.

Véase a continuación una estructura a la cual se le han creado vínculos externos.

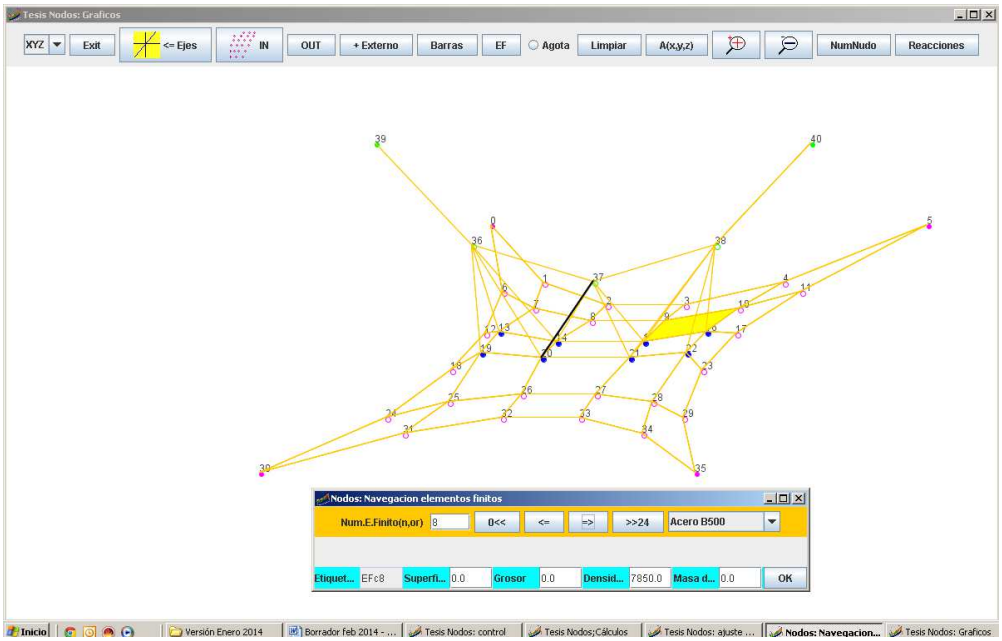


Apreciemos como los nudos originarios de la malla están en rojo, mientras que los nudos que se han ido creando como vínculos externos están en verde. Los que aparecen sombreados en azul es porque se les ha introducido una carga.

El icono Barras se maneja conjuntamente con el pulsador Agota, y nos permite ver el estado de agotamiento de la estructura:



Las barras disponen de un proceso de verificación que determina –para un coeficiente de seguridad dado- en que rango de aprovechamiento está, mostrándose en una escala gráfica (de negro a rojo) su situación.
El icono EF se utiliza para visualizar los Elementos Finitos, desplegándose una ventana que muestra sus propiedades (en desarrollo)



A continuación tenemos los iconos:

Dibuja/Borra: Muestra los números de nudos, las coordenadas de los apoyos, etc… en general activa o desactiva los textos que se estén visualizando en el dibujo.

A<=(X,Y,Z) Muestra en pantalla las coordenadas de los apoyos.

Lupas +/- : Sirven de Zoom.

NumNudo: Muestra la numeración de los nudos.

Reacciones: Muestra el vector de reacciones aplicado sobre los apoyos.

Por último tenemos los iconos:

El primero muestra la ventana de análisis de vibraciones y el segundo (una flecha) activa el cálculo. Centrándonos en el primero:

Esta ventana expone los principales resultados del programa desde el punto de vista dinámico, así muestra:

- El listado de modos de vibración, ordenadamente según el cuadrado de la velocidad angular de oscilación.
- La amplitud máxima del movimiento (en metros) para mantener la validez de la matriz de rigidez de cálculo así como la escala de amplitud de cálculo (es un factor que relaciona la amplitud posible con la amplitud “sin escala” del resultado del problema matricial)
- La energía asociada a un ciclo de oscilación (ajustadamente a la amplitud máxima posible)
- La frecuencia, adaptadamente a los factores de masa en cada dirección.

Los parámetros que el usuario podrá introducir son:

- La escala visual para apreciar el movimiento.
- Los factores de masa en cada dirección (variará la frecuencia y la forma de oscilación respetando el modo de oscilación). Es importante que el usuario maneje los factores de masa y vea como la frecuencia se adapta a los diferentes factores. Quizá para que se “active” el cambio de masas deba cambiar de modo de vibración y volver al que desea analizar para que se actualicen los valores.

Por último las flechas permiten cambiar entre la forma original y la forma con la oscilación superpuesta (aplicado el factor de escala visual). La animación hace que se pase automáticamente de un estado al otro. Los factores X10 y /10 alteran la escala

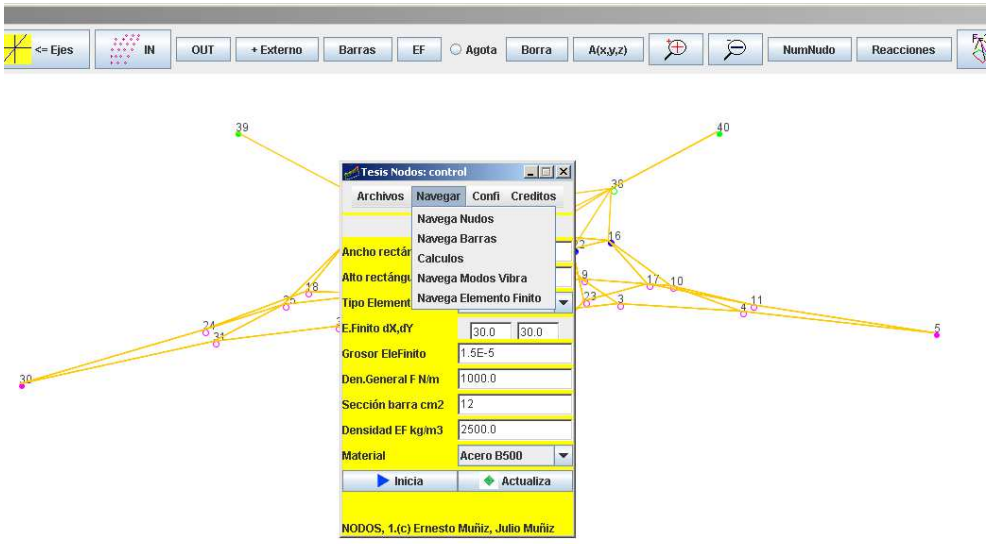
en el tiempo para que se visualice la oscilación más rápido o más lento (manéjese con precaución pues puede estar largo tiempo oscilando con el programa bloqueado.

Hasta este punto hemos visto el menú control y la pantalla gráfica. A continuación procede pasar a los navegadores.

IMPORTANTE: PARA PODER NAVEGAR LAS FRECUENCIAS SE DEBERÁ HABER REALIZADO PREVIAMENTE EL CALCULO DINÁMICO Y SU VERIFICACIÓN. Para ello se deberá ir al Navegador de Cálculos.

3- Navegadores.

Los navegadores permiten visualizar y actuar sobre los parámetros de diseño y cálculo de la estructura.



Así tenemos navegadores de Nudos, de Barras, de Cálculos, de Modos de Vibración y de Elementos Finitos. Algunos de ellos –como el de modos de vibración o el de los elementos finitos- son accesibles desde la ventana gráfica y por ello ya han sido expuestos.

3.1 Navegador de nudos:

Pulsadores +-X/Y/Z: Situados en la parte superior permiten mover el nudo seleccionado gráficamente con un salto unidad.

Etiqueta de nudo: es un identificador a nivel de cálculo. Contiene implícita codificación necesaria a nivel de ordenamiento interno, permitiendo identificar al usuario la posición del nudo en determinados listados de cálculo, independientemente de la numeración del nudo, pues ésta bajo determinadas circunstancias puede variar (numeración re-ordenada).

Nudo seleccionado: El usuario seleccionará el nudo sobre el que desea actuar escribiendo su número según se asigna en la pantalla gráfica.

Apoyo: El usuario determina si el nudo es apoyo o no.

Carga X/Y/Z: Valores de la acción en Newtons.

+ Masa Carga: Si se activa la carga negativa en el eje Z (-Z) se entenderá como masa.

X/Y/Z = Coordenadas del nudo según se introducen por el usuario (tiene sentido cuando es apoyo).

Confirmar: Validar los datos introducidos.

Véase que la ventana gráfica muestra la posición y las cargas del nudo en el momento que es seleccionado.

3.2 Navegador de Barras:

Permite discurrir por las barras de la estructura. La selección de la barra se realiza bien según su numeración, bien por los botones de avance/retroceso (que irán sombreando la barra en la estructura) o bien seleccionando su nudo inicio o fin (con el botón de avance se resuelven los casos en los que se produce coincidencia de barras sobre un mismo nudo, dado que al poner un valor automáticamente se actualiza el otro)

Seleccionada la barra se muestran los parámetros esenciales de la barra:

Etiqueta: Al igual que los nudos es un identificador interno que permite reconocer la barra en determinados listados.

Tracción: Cuando está activado ello implica que salvo que se imponga manualmente lo contrario la densidad de fuerza es positiva. Si fuera compresión sería negativa.

Material: En la imagen figura Acero B500, siendo un desplegable en el que el usuario introduce los valores por defecto existentes en base de datos. Manualmente podrá alterar los mismos en las casillas posteriores.

Densidad de Fuerza de Forma: Valor de Densidad de fuerza para obtener la forma de equilibrio. Es manejable por el usuario a efectos de alcanzar un cierto Form Finding o imponer un determinado nivel de pretensado.

Densidad de Fuerza de Calculo: En función del dimensionamiento es un valor deducido de cálculo correspondiente con la Densidad Dinámica de Fuerza según se ha descrito en la memoria justificativa del método.

Longitud: Medida de la barra en la situación de equilibrio descrita.

Fuerza Axial: Solicitación de la barra en la situación de equilibrio descrita.

Densidad (Kg/m3): Densidad del material a efectos de determinar la masa de la barra.

Módulo elástico: Propiedad del material de constituyente

Límite elástico: Propiedad del material constituyente.

Coeficiente de seguridad: Valor considerado a efectos de determinar el ratio de aprovechamiento de la barra.

Sección cm2: Sección con la que se dota la barra en estudio conforme al material seleccionado o descrito.

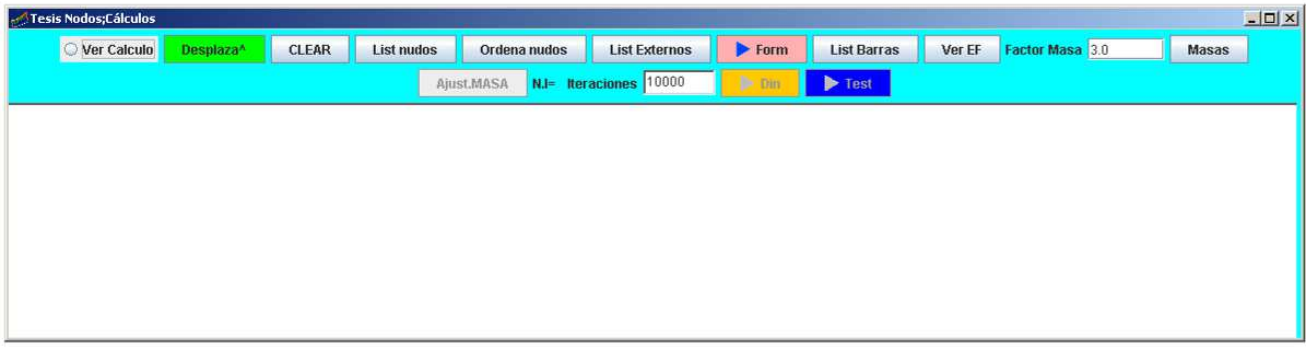
Sección de cálculo: Valor que sería necesario para estrictamente cumplir con la solicitación de equilibrio para el material y coeficiente de seguridad dado.

Ratio de aprovechamiento: Relación entre la sección necesaria y la dispuesta.

Masa (Kg): Peso de la barra.

Acortamiento Inicial: Longitud que tendría que deducirse de la longitud de equilibrio de la barra para que al entrar ésta en carga adoptara el tamaño de compatibilidad.

3.3 Navegador de Cálculo.



Las opciones que muestra son:

Ver cálculo: Al activarse se visualiza la totalidad del proceso de cálculo, lo cual puede ser verdaderamente farragoso.

Desplazar: Permite avanzar en la pantalla de visualización dado que en determinados casos pude terminar el proceso de cálculo sin que se terminen de ver los datos, siendo necesario avanzar los listados haciendo uso de dicho botón.

Clear: Pone en blanco la pantalla.

List nudos: Lista los nudos numeradamente conforme la pantalla gráfica

Ordena nudos: Lista los nudos ordenadamente conforme el proceso de cálculo (apoyos primero)

Lista externos: Lista los nudos introducidos manualmente por el usuario.

Form: Botón de cálculo para buscar la forma de equilibrio.

List Barras: Lista las barras constituyentes con sus propiedades.

Ver EF: Lista los Elementos Finitos con sus propiedades (En desarrollo)

Factor de Masa: Factor de masas base de cálculo (por defecto 3: Importante respetar dicho valor salvo que se tenga MUY CLARO lo contrario)

Masas: Lista las masas aplicadas sobre los nudos, incluyendo la parte correspondiente de la masa de las barras tributarias sobre cada nudo.

Ajuste Masas: Realiza un proceso iterativo de cálculo que converge con la geometría de equilibrio una vez se estabiliza ésta con el valor de masa de las barras para esa geometría. (N.I. = muestra el número de ciclos de cálculo hasta llegar a la convergencia)

Iteraciones: Número Máximo de ciclos de carga para la resolución del problema de autovalores.

Din: Botón que se ha de presionar para que se resuelva el cálculo dinámico.

Test: Verificación energética del citado cálculo.

ES NECESARIO EJECUTAR “Din” y “Test” PARA PODER NAVEGAR LAS FRECUENCIAS.

Firma:

Ernesto Muñiz Martín

Septiembre de 2014.