CONTENIDO

Introducción

En cerámica tradicional, el diseño de productos se hace, en la mayoría de los casos, con métodos no rigurosos, por tanteo, reglas empíricas o con base en la heurística de un grupo de expertos. Este procedimiento puede tomar de 3 a 6 meses dependiendo de la cantidad de piezas que forman una familia y esto se traduce en pérdidas de competitividad de la empresa. Es así como el desarrollo de la tecnología en la industria cerámica está llamada a encontrar procedimientos más eficientes en tiempo y en costos, que permitan tener productos más competitivos en el mercado. En la industria de la vajillería se requieren cada vez más productos de moda, de alta calidad y bajo precio, con un desarrollo rápido de formas y decoraciones.

El método de los elementos finitos, (MEF), entendido como un método para encontrar una solución aproximada de un modelo simplificado, entra a jugar un papel importante en el diseño cerámico. El tratamiento numérico reduce el modelo simplificado a una forma soluble mediante un número finito de operaciones numéricas. Esta solución viene caracterizada por un número finito de parámetros, llamados grados de libertad, y al proceso se le conoce como discretización. Se espera que cuando el número de elementos sea grande, la solución por elementos finitos presente convergencia hacia la solución exacta, que es independiente de la selección de la discretización.

El proceso estándar de manufactura de porcelana por vía seca se compone de varias etapas: después de una preparación de la pasta, se seca en un atomizador y pasa al proceso de formación por prensado isostático, donde se compacta y queda la pieza con una densidad en crudo homogénea. Esta pieza pasa al proceso de cocción, donde ocurre la sinterización (densificación), con una contracción natural de la pieza y se presenta un cambio en su forma debido al comportamiento mecánico elástico y tipo “creep”.

Hay estudios previos que tratan el tema, aplicados en su mayoría a compuestos cerámicos de alta tecnología [2], [5], [36], [39] y algunos trabajos aplicados a la cerámica tradicional como los presenta Navarro en el tratamiento de baldosas cerámicas [31]. Para modelar la sinterización

se utilizó el procedimiento basado en el principio de conservación de la masa presentado por Aydin, Sanliturk y Briscoe [36] y la modelación mecánica se hizo con base en la teoria elástica y el modelo de Norton para el caso “creep” [5] y en su solución se empleó el principio de trabajo virtual [38]. Las propiedades del material y los parámetros necesarios para el modelo se obtuvieron a partir de mediciones directas en laboratorio, diseño de pruebas indirectas desarrolladas en la planta de producción y de referencias bibliográficas.

Agradecimientos y notas de aceptación

APLICACIÓN DEL ANALISIS DE ELEMENTOS FINITOS A LA PREDICCIÓN DE LA GEOMETRÍA FINAL DE PLATOS PROCESADOS POR MONOCOCCIÓN

Jurado 1

Jurado 2

UNIVERSIDAD EAFIT
POSGRADO EN MATEMÁTICAS APLICADAS
DEPARTAMENTO DE MATEMÁTICAS
MEDELLÍN 2005

Debemos agradecer a la biblioteca de la universidad EAFIT por su colaboración en la consecución de artículos relacionados con el tema de este trabajo.
De manera muy especial queremos agradecer al profesor Jorge Luis Restrepo Ochoa, director del departamento de Mecánica de la universidad EAFIT y asesor incondicional de esta tesis, por su dedicación, sus asertadas sugerencias, su apoyo continuo y sus grandes esfuerzos en procura de un trabajo de calidad.
La colaboración de la empresa "Locería Colombiana" ha sido realmente valiosa al prestarnos sus instalaciones para realizar las pruebas de laboratorio.

CAPÍTULO 1

Presentación del Problema

1.1 Proceso de Diseño y Producción Cerámica

La industria de producción de vajillas pertenece al sector de la cerámica blanca tradicional, que se caracteriza por ser blanca, de grano fino, esmaltada y fabricada a partir de pastas triaxiales, que son una mezcla de arcillas, cuarzo y feldespato, [23]. El proceso general de producción cerámica parte del beneficio de las materias primas, bien sea por dispersión para el caso de las arcillas o de molienda para los cuarzos y el feldespato. Una vez se tienen preparadas, se ensambla la pasta de acuerdo con la formulación establecida para cada tipo de producto, mediante un proceso de dispersión y/o molienda; de acá se obtiene una pasta en suspensión que puede usarse directamente o pasar a los procesos de: -Atomizado, del cual se obtiene una pasta granulada; -Filtroprensado, del cual se obtiene una pasta en galletas.

Partiendo de estas pastas, se pasa al proceso de formación, que tiene varias alternativas según el tipo de producto a desarrollar, estos procesos pueden ser: Prensado, Forjado y Vaciado tradicional. En ésta etapa se obtiene un cuerpo blando que aún no tiene las propiedades adecuadas para el uso, como la resistencia, color, brillo, higiene entre otras. Una vez se tiene la pieza en éste estado, pasa al esmaltado (si se trata de piezas tipo monococción), o a una primera quema en el horno (si se trata de piezas tipo bicocción). En la cocción se obtiene un cuerpo con las propiedades características de la cerámica: resistencia, color y forma deseadas.

Para el caso de bicocción, las piezas pasan a un proceso de esmaltado y luego son sometidas a un segundo tratamiento térmico, para lograr cocer el esmalte y darles las características finales de acabado. A partir de éste estado, las piezas pueden decorarse o ir directamente al empaque y distribución del producto en el mercado. En la figura (1.1) se esquematiza el proceso productivo.

Para el caso particular de la fabricación de platos de porcelana blanca, objeto de estudio de este

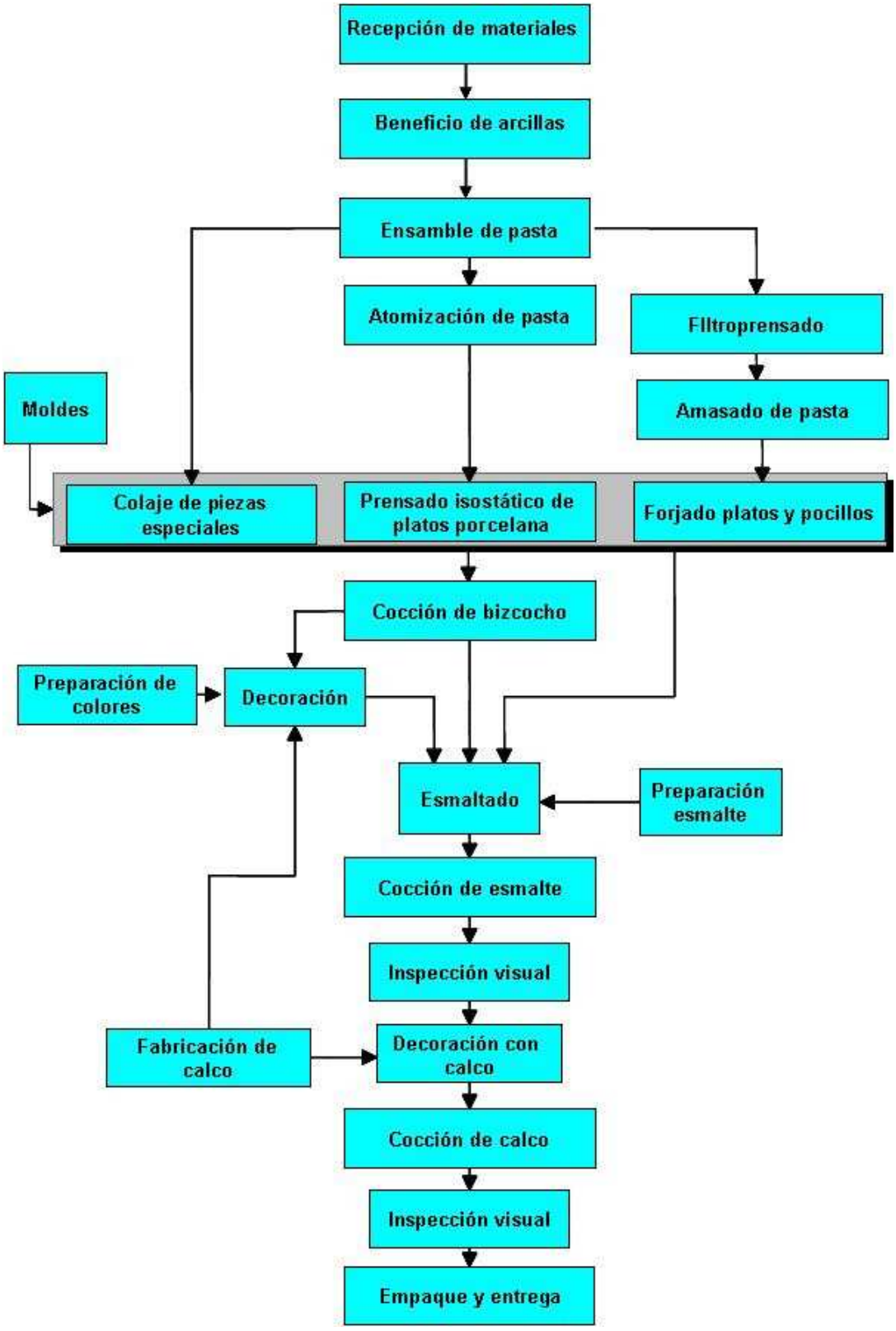


Figura 1.1: Proceso cerámico

trabajo, el proceso estándar de manufactura se realiza por vía seca. Después de una preparación de la pasta, ésta se seca en un atomizador del cual se obtiene una pasta granulada con máximo un 4 % de humedad y una granulometría determinada, luego pasa al proceso de formación por prensado isostático, para el caso especial de estudio se utilizó una Prensa PHO600, ver figura (1.2).

Figura 1.2: Prensa isostática PHO600

En esta prensa, la pasta granulada se alimenta a un molde metálico recubierto en poliuretano. Una vez se llena el molde, la prensa sella y se somete a un ciclo de compactación, del cual se obtiene una pieza cruda con densidad homogénea. A ésta pieza se le aplica un esmalte, compuesto de materiales vitreos, que le daran el brillo, estética e higiene necesarios; después pasa al proceso de cocción en carga individual, soportada en refractario, en un horno túnel, el cual opera a 18 horas de ciclo y una temperatura máxima de 1200^oC. En ésta etapa, ocurren dos procesos claves que determinan la geometría final de la pieza, el primero es la sinterización (densificación), que conlleva una contracción de la pieza y el segundo un desplazamiento mecánico de naturaleza elástica y otro de fluencia (creep) que le cambian su forma. En las figuras (1.3) y (1.4) se puede apreciar un semiplano de un plato de porcelana crudo versus uno cocido. Se puede apreciar la contracción de la pieza durante el proceso de cocción y el leve desplazamiento de este, especialmente en dirección vertical.

Sinterización en Estado Sólido

Los cambios que ocurren durante el proceso de cocción, especificamente con la sinterización, están relacionados con los cambios en la forma y tamaño de los poros, es decir con los cambios en porosidad. Durante la sinterización se pasa de un plato poroso, a una cerámica densa y fuerte.

Figura 1.3: Plato crudo versus plato quemado vista perfil

Figura 1.4: Plato crudo versus plato quemado vista superior

Un plato en crudo (sin cocer) está compuesto por granos individuales, separados entre un 25 y 60 % en volumen, dependiendo de las propiedades del material y del método de formación utilizado en la producción. Para maximizar las propiedades de resistencia, traslucidez y conductividad térmica, es deseable eliminar tanto como sea posible esta porosidad. Estos resultados se obtienen durante la cocción, mediante la trasferencia de material de una parte de la estructura a otra. El tipo de cambios están ilustrados en la figura (1.5)

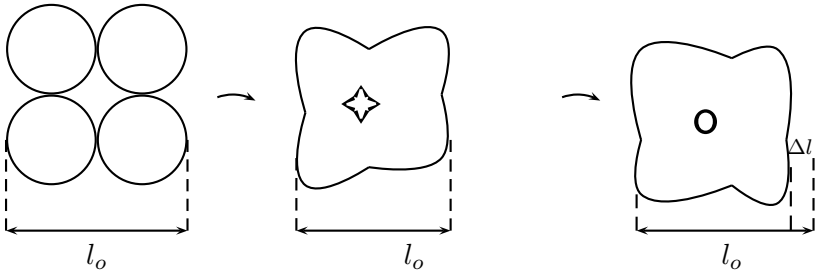


Figura 1.5: Proceso de sinterización

En la figura 1.5 a) se aprecia una ampliación de como está conformada la estructura del plato en crudo. Los granos de pasta se pueden representar por esferas, ligadas mecánicamente por un

aditivo químico. Una vez empieza el proceso de sinterización, se inician cambios en la forma de los poros, figura (1.5) b) convirtiendosen en canales o esferas aisladas pero sin necesariamente presentarse cambios en tamaño. Más adelante, figura (1.5) c) se presentan cambios en el tamaño y en la forma, los poros empiezan a volverse más esféricos y pequeños, lo que da paso a la contracción de la pieza y su estado final dependerá del material utilizado y del tratamiento térmico en cuestión.

El cambio en energía libre que da paso a la densificación es el decrecimiento en el área superficial y la disminución de la energía libre superficial mediante la eliminación de las interfases sólido-vapor. Esto usualmente toma lugar con la formación de una nueva interfase sólido-sólido de menor energía libre, la red decrece en energía libre, del orden de 1 cal/g por partícula de un micrómetro. En una escala microscópica, la transferencia de material es afectada por la diferencia de presión y los cambios en energía libre a través de las superficies curvadas. Para profundizar en éste tema ver [8], [23], [36] y [40].

Deformación Mecánica

La cerámica normalmente no exhibe una deformación plástica apreciable, por tanto es frágil y se fractura con pequeña o ninguna deformación. Su uso en muchas aplicaciones es limitado por sus propiedades mecánicas relativamente pobres; sin embargo, en procesos que requieran altas temperaturas, resultan ser buenos competidores por sus propiedades de baja fluencia (comportamiento creep) a estas condiciones.

Existen muchas aplicaciones en ingeniería donde es necesario manipular y controlar la deformación mecánica elástica de la cerámica, como es el caso de la producción de vajillas, en la cual es necesario garantizar una forma final diseñada, una vez la pieza cerámica ha pasado por las diferentes etapas industriales. Hasta unos límites dados, la tensión (σ) es directamente proporcional a la deformación unitaria ϵ de acuerdo con la ley de Hooke.

$$\sigma = E * \epsilon \tag{1.1}$$

donde E es el modulo de Young, cuyos valores más comunes oscilan entre 5 y 90×10^6 psi $\equiv 3.4 \times 10^4 Pa$ y $6.2 \times 10^{11} Pa$, ver [23]. Similarmente el esfuerzo de corte τ es directamente proporcional a la deformación de corte γ mediante la ecuación.

$$\tau = G * \gamma$$

donde G es el módulo de rigidez o módulo de elasticidad en corte. Cuando una muestra es extendida en tensión, hay un decrecimiento del espesor; la relación entre la disminución del espesor d y el crecimiento de la longitud l se conoce como el coeficiente de Poisson (ν) y está dado por

$$\nu = \frac{\Delta d/d}{\Delta l/l}$$

Para flujo plástico, flujo viscoso y fluencia (creep) el volumen pemanece constante, por lo tanto ν es igual a 0.5.

Adicional al comportamiento elástico, puede presentarse deformación bajo esfuerzos aplicados a una temperatura t, que pueden inducir cambios permanentes en la forma de la pieza. Los procesos implicados en este comportamiento, son variados y complejos y aún no se tiene una comprensión total de los detalles a escala atómica; sin embargo, cuando se trata del flujo viscoso, la deformación plástica o el comportamiento creep, en aplicaciones a altas temperaturas es clave entender los modelos a nivel macroscópico. A continuación se da este acercamiento para el comportamiento creep, proceso de gran influencia en la determinación de la geometría final de platos cerámicos.

El comportamiento creep aparece cuando se somete el espécimen cerámico a un esfuerzo constante, durante un periodo t de tiempo a elevada temperatura. Esta deformación dependiente del tiempo, es causada por el movimiento de las vacantes activadas térmicamente y el movimiento de las dislocaciones bajo carga. Para entender más el fenómeno remitamonos a la figura (1.6):

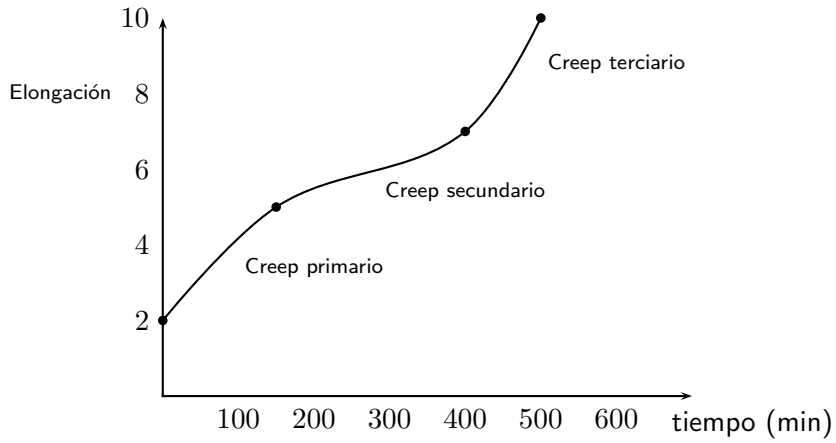


Figura 1.6: Curva creep caraterística

Después de una extensión elástica, hay un periodo durante el cual la velocidad de deformación decrece, etapa conocida como creep primario o transiente. Esta es seguida por un periodo de velocidad constante de deformación conocido como creep secundario o de estado estable; y finalmente aparece una tercera etapa cuando la velocidad de deformación creep incrementa a causa de la fractura inminente. La forma de la curva de deformación creep presentada en la figura (1.6) varía dependiendo de las condiciones particulares de las pruebas y del material ensayado. Frecuentemente la parte inicial de la curva o creep primario se puede representar por una expresión de la forma

$$\dot{\epsilon}_c = k\sigma^{-n}$$

donde
$\dot{\epsilon}_c$: Velocidad de deformación unitaria n y k constantes del material y σ: Tensión.
A bajas temperaturas $n = 1$; y la deformación es $\epsilon_c = k' \log(\sigma)$ siendo k' una constante del modelo.

Ambos, la temperatura y el esfuerzo afectan la forma de la curva de deformación; cuando se alcanza la temperatura deseada, la deformación es más rápida y se acorta el periodo de velocidad constante de deformación. El mismo comportamiento se observa al aumentar el esfuerzo aplicado. Cuando se pasa a la segunda etapa, frecuentemente la velocidad de deformación es proporcional a alguna potencia del esfuerzo aplicado.

$$\dot{\epsilon}_c = k\sigma^{n}$$

Donde n varía entre 2 - 10 y un valor típico puede ser 4, ver [23].

La fuerte dependencia de la velocidad de deformación a esfuerzo constante con la temperatura, ha llevado a desarrollar una variedad de ensayos que permitan modelar el proceso; en estos ensayos, se aplica una carga constante al espécimen cerámico, y la temperatura deseada se alcanza mediante incrementos a velocidad constante. La deformación resultante es una suma compleja de expansiones térmicas y fluencias. En forma más general, la deformación unitaria (ϵ) total, no sólo es función del esfuerzo o tensión (σ), sino también del tiempo (t), la temperatura (T), la estructura (s).

$$\epsilon = f(\sigma, t, s, T)$$

Para profundizar en la termodinámica de la deformación creep ver [5] y [23]

Diseño y Desarrollo de Productos

Actualmente, el proceso de diseño y desarrollo de nuevos productos requiere una mayor innovación en la industria cerámica debido a la fuerte competencia de productores de China y de los paises del este y por una tendencia mundial de diferenciación y rápida respuesta frente a cambios de moda y tendencias. Frente a esta dinámica de mercado externa, las empresas cerámicas deben optimizar sus procesos de desarrollo para lograr obtener productos de mejor calidad, mas diferenciados y de rápida respuesta. Tradicionalmente el diseño de formas, obedece a la experticia de un grupo de técnicos que, trabajan con un método de tanteo, elaboran un prototipo, observan su forma final y determinan que correcciones hay que hacer para que se obtenga la pieza final deseada. Esto conlleva a realizar varios prototipos que toman un tiempo promedio de 3 a 6 meses y unos costos de producción elevados, en muchos casos no evaluados.

Este diseño y desarrollo tradicional de productos cerámicos parte de la investigación de mercados para determinar las especificaciones del producto. Con ella, se realiza un diseño conceptual,

unos cálculos preliminares y se plantea una propuesta de producto que involucra la estética, la selección de materiales y la ingeniería del proceso y del producto. Esta propuesta se analiza para realizar un prototipo el cual pasará a ensayo semi-industrial y validación en el campo. Este ciclo de desarrollo se aprecia en la figura (1.7).

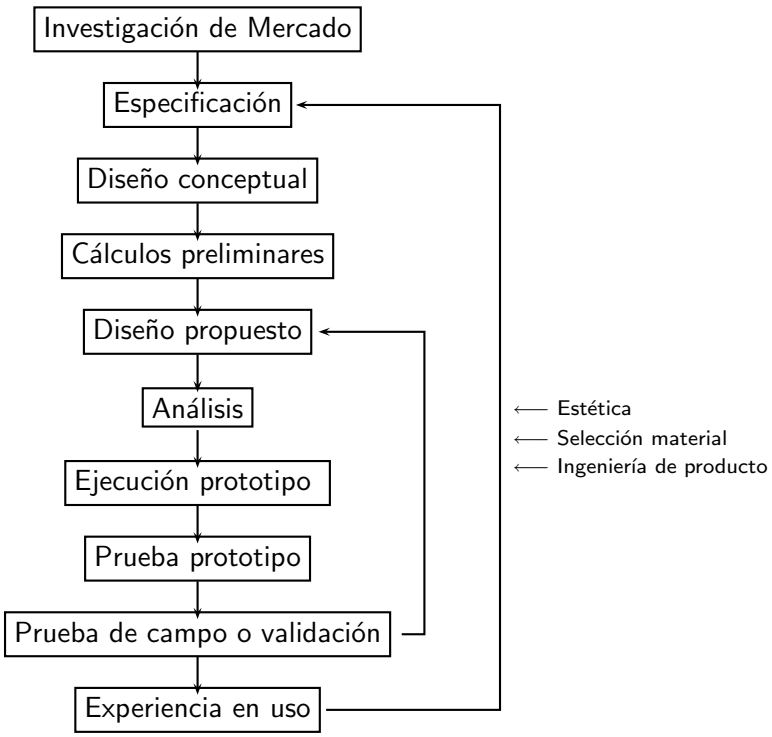


Figura 1.7: Ciclo de Diseño

Este proceso conlleva una serie de ajustes en el prototipo que implican tiempo y dinero para la empresa. Este ciclo puede optimizarse usando modelación matemática y simulación numérica con el método de los elementos finitos, como se esquematiza en la figura (1.8).

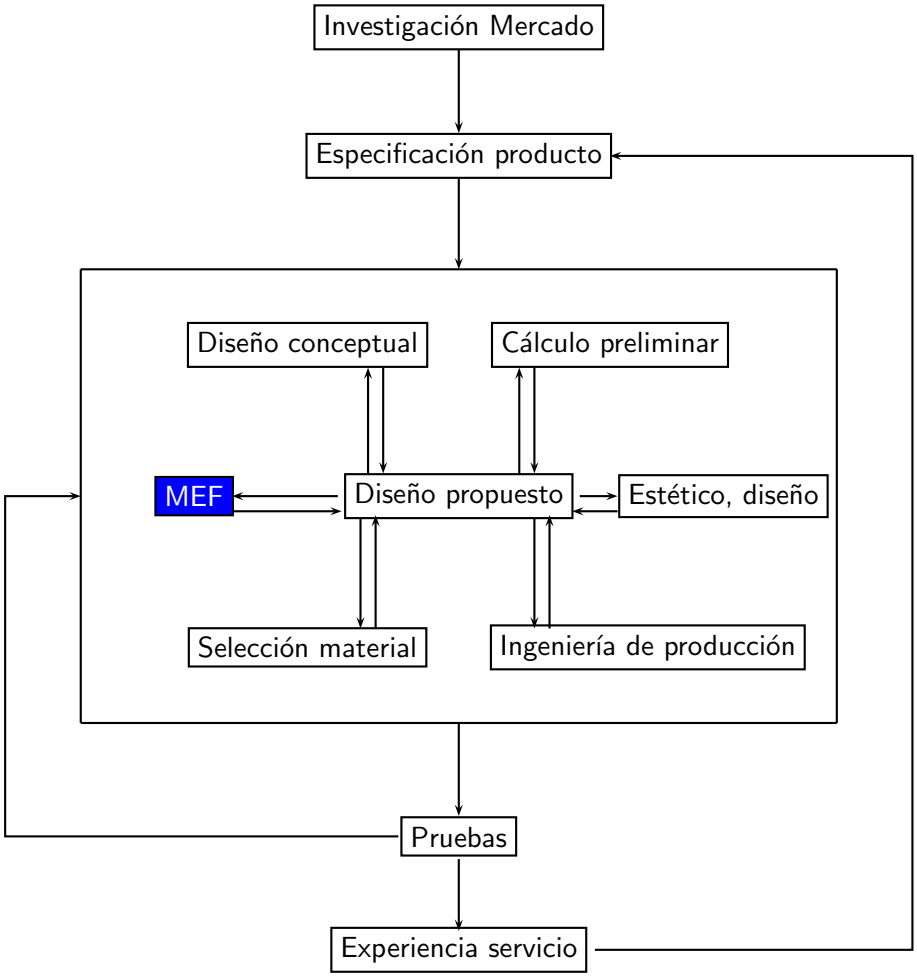


Figura 1.8: Ciclo de Diseño optimizado

Este ciclo permite simular los comportamientos de las diferentes propuestas de diseño, optimizando el ciclo; principalmente cuando hay pruebas destructivas y fallas durante el servicio del producto. Operar con este ciclo presenta los siguientes beneficios, ver [20] y [41]:

- Mejorar el desempeño del producto.
- Disminución del costo del producto.
- Reducción del tiempo del desarrollo.
- Eliminación o reducción de las pruebas.
- Logro de estándares de calidad a la primera vez.
- Satisfación de los clientes.
- Información para la toma de decisiones.

Es precisamente en este proceso de diseño y desarrollo de productos cerámicos donde está enmarcado este trabajo. La propuesta es desarrollar una metodología para optimizar el diseño de vajillas para la mesa, mediante la modelación matemática y simulación numérica, que permita reducir los tiempos de diseño y desarrollo. En el caso de estudio, se modelará y simulará el comportamiento en cocción de un plato de porcelana vitrificada, proveniente de un proceso de prensado isostático, y cocción en horno túnel con carga individul, a las condiciones estables de temperatura y tiempo del proceso industrial operante.

1.2 Modelos Cerámicos de Solución

Prácticamente no se conocen modelos cerámicos para modelar estos comportamientos en cerámica tradicional. En las principales publicaciones cerámicas seriadas se concentran en aplicaciones de alta tecnología tales como compuestos cerámicos, que sirven como base para plantear un acercamiento de la modelación de piezas cerámicas tradicionales como platos de uso doméstico.

Existen diferentes acercamientos para modelar tanto el proceso de sinterización de piezas cerámicas, asi como su comportamiento mecánico durante la cocción. Taheri [39] nos presenta un desarrollo basado en el - Standar Linear Solid Viscoelastic Model (SLSVM)- el cual calcula los esfuerzos τ y deformaciones mecánicas ϵ, caracterizado por dos constantes del material k_1 y k_2 y el coeficiente de viscosidad n, su formulación básica es:

$$\sigma + \tau_f.\sigma = k_2.\epsilon + \tau_f(k_1 + k_2)\dot{\epsilon}$$

donde $\tau_f = n/k$ tiempo de retardo.

Este modelo es capaz de dar cuenta de la respuesta instantanea de material una vez terminada la sinterización.

Taheri [39] tambien presenta un modelo de sinterización basado en la ley de viscosidad lineal y el comportamiento de fluencia de los materiales porosos a nivel micromecánico. El modelo está caracterizado por la viscosidad como una función de la densidad relativa y el tamaño del grano, su formulación básica, es:

$$\dot{\rho} \sim ng(\rho)D\left(1 - \frac{\sigma_m}{\sum(\rho)}\right) l^{-a}$$

donde:
l : longitud grano, n : poros promedio / grano, σ_m : esfuerzo medio, $\sum$: potencial de sinterización, D : difusividad, g : función de densidad y a : constante que oscila entre 3 y 4.

Kraft y Coube [24] utilizan modelos basados en leyes viscosas lineales caracterizado por la viscosidad aparente, la viscosidad de corte y el esfuerzo de sinterización. La velocidad de densificación se calcula como la traza del vector de velocidad de deformación.

Tsvelikh [40] presenta un modelo basado en funciones de contracción del material con densidad heterogénea.

Aydin y otros [2] presentan un modelo de sinterización basado en el principio de conservación de la masa. Este modelo consta de dos etapas básicas: Determinación de los elementos masicos individuales (elemento finitos) y la solución iterativa para determinar una constante de proporcionalidad que garantice la conservación de la masa al darse la densificación en el proceso de cocción; esta segunda fase implica la solución de un sistema no lineal.

Navarro [31] presenta un modelo basado en la ley constitutiva del material, expresada como una de las ecuaciones más ampliamente utilizadas en cerámica técnica.

$$\dot{\epsilon}_c = A'\left(\frac{\sigma}{\sigma_o}\right)^n$$

donde
$\dot{\epsilon}_c$ velocidad de deformación creep, A' constante experimental σ_o esfuerzo de referencia y n exponente de esfuerzo.
Para modelar el fenómeno de sinterización se adoptará el modelo de Aydin y otros [2] y la parte mecánica se hará a partir de las leyes elásticas y el principio del trabajo virtual (PTV), combinado con el modelo de Norton para el caso creep, ver [38] y [5]. El PTV se seleccionó porque es el más general de los principios de equilibrio y es válido para problemas:

- Con pequeños y grandes desplazamientos con linealidad o no linealidad geométrica.
- Para relaciones tensiones-deformaciones lineales y no lineales (linealidad o no linealidad mecánica). Para profundizar sobre el tipo de no linealidades ver [5].
- En regimen estático o dinámico.

1.3 Modelos Matemáticos de Solución

En el transcurso de los años, se han desarrollado varias técnicas numéricas para solucionar sistemas de ecuaciones diferenciales que explican el comportamiento de sistemas en ingeniería como son el método de diferencias finitas (DF) y el método de elementos finitos (MEF). El MEF requiere hacer una división del dominio del problema en muchos subdominios, cada subdominio se llama elemento finito, esto se llama discretizar el problema. Diferencias finitas por su parte es similar a MEF en la solución de ecuaciones diferenciales de problemas que requieren discretización, DF también requiere generar una malla sobre el cuerpo ha ser analizado. DF es usado en la solución de problemas transientes y se ha usado exitosamente en problemas de transferencia de calor y análisis de flujo, pero presenta dificultad en la modelación de geometrías complejas. La aproximación es menos intuitiva que el MEF y su popularidad a declinado en la medida que las técnicas MEF han mejorado. Las comparaciones entre estos dos métodos y el campo de aplicación de los elementos finitos se puede consultar en [12].

El tratamiento matemático del método de los elementos finitos está basado en la formulación variacional de las ecuaciones diferenciales elípticas. Las soluciones de las más importantes ecuaciones diferenciales se pueden caracterizar por propiedades mínimas y los correspondientes problemas variacionales tienen solución en los espacios de funciones llamados espacio de Sobolev, ver [7]. El tratamiento matemático comprende la minimización en subespacios lineales de dimensión finita; y una escogencia apropiada de estos subespacios son los **Espacios de elementos finitos** que se explica más adelante. Para las ecuaciones diferenciales lineales, es suficiente trabajar con el método del espacio de Hilbert. En este contexto, inmediatamente se consideran las soluciones basadas en las formulaciones débiles.

Los cálculos con MEF buscan una expresión aproximada de la solución de la forma

$$u \approx \hat{u} = \sum N_i a_i = Na$$

donde N_i son las funciones de forma, definidas localmente para cada elemento y todos o algunos de los parámetros a_i son incógnitas; para obtenerlas se debe resolver una ecuación integral que nos permite obtener la aproximación elemento por elemento para luego proceder al ensamblaje. Se dispone de dos procedimentos distintos para obtener la aproximación, el primero es el método de los residuos ponderados y el segundo consiste en determinar funcionales variacionales.El método de residuos ponderados busca minimizar una expresión de error en la ecuación diferencial; el método variacional se basa en la búsqueda de la solución del problema a partir de la resolución de una ecuación integral que representa una propiedad general del sistema, las cuales son aplicables cuando las ecuaciones que gobiernan el sistema son ecuaciones diferenciales.

CAPÍTULO 2

Soluciones Generalizadas Basadas en el Principio de Trabajo Virtual [PTV]

En este capítulo se va a realizar el tratamiento matemático, de las características esenciales de las soluciones generalizadas basadas en el PTV, el cual va a ser empleado para modelar el comportamiento mecánico del plato cerámico, que como se estableció en el capítulo anterior, es el más general de los principios de equilibrio, válido para comportamientos lineales o no lineales. Este tema se puede profundizar en las siguientes referencias [3], [38] y [42].

El PTV es una de las muchas posibles formulaciones generalizadas que pueden ser construidas. Todas las formulaciones generalizadas son de la forma: encontrar $u \in X$ tal que $\mathscr{B}(u,v) = \mathscr{F}(v), \forall v \in Y$, donde $\mathscr{B}(u,v)$ es una forma bilineal cuyas propiedades se describen en el apendice A y $\mathscr{F}(v)$ es un funcional lineal, cuyas propiedades se describen en el apendice B. Los espacios X son llamados espacios ensayo y las funciones que están en X son llamadas funciones ensayo. Los espacios Y son llamados espacios de prueba y las funciones en Y se llaman funciones prueba. En el caso del trabajo virtual las funciones de ensayo son llamadas desplazamientos virtuales. El PTV se puede siempre establecer de tal forma que $X = Y \subset E(\Omega)$. Las formulaciones generalizadas a menudo se llaman formulaciones débiles. La particularización de la formulación generalizada depende de las condiciones de frontera impuestas.

Para apoyar la descripción matemática del PTV, se considera el plato cerámico como un cuerpo elástico axi-simétrico, por tratarse de un sólido de revolución, ubicado en un sistema de coordenadas cilíndricas, sujeto a fuerzas externas con condiciones cinemáticas de frontera establecidas y que sufre unas deformaciones ocasionadas por estas fuerzas y expresadas mediante las funciones de desplazamiento (u). En adelante se denota por Ω el dominio abierto y $\overline{\Omega}$ el dominio cerrado del cuerpo de trabajo.

2.1 Trabajo Virtual en Coordenadas Cilíndricas

Este tipo de problemas son similares a los análisis de tensión y deformación en cuerpos planos, y pueden tratarse como un problema bidimensional, usándose las mismas funciones de deformación para el caso plano. El volumen del material de cada elemento es el de un sólido de revolución.

El trabajo interno está ligado a las tres componentes de la deformación, situados en el plano de las coordenadas, no apareciendo la componente de la tensión normal a dicho plano por ser nulos los valores de dicha tensión, así mismo la deformación. En los problemas de revolución todo desplazamiento radial induce automaticamente una deformación en dirección circunferencial y como las tensiones en esa dirección son no nulas, hay que considerar esta cuarta componente de la deformación y de la tensión asociada a ella $(\epsilon_\theta, \sigma_\theta)$.

Para formular el PTV en coordenadas cilíndricas, se define el vector de desplazamiento $\{u\}$ como

$$\{u\} \stackrel{def}{=} \{u_r(r,z),\ u_z(r,z)\}^T \tag{2.1}$$

donde u_r y u_z son las componentes de desplazamiento en la dirección radial y axial respectivamente. Ubicados en un sistema coordenado cilíndrico, los subindices r, z denotan estas direcciones y el superindice T denota la transpuesta.

Se denota la deformación unitaria que se corresponde con el vector $\{u\}$ por $\{\epsilon^{(u)}\}$.

La relación entre deformación y desplazamiento está dada por:

$$\epsilon_r^{(u)} = \frac{\partial u_r}{\partial r}, \qquad \epsilon_\theta^{(u)} = \frac{u_r}{r}, \qquad \epsilon_z^{(u)} = \frac{\partial u_z}{\partial z}, \qquad \gamma_{rz}^{(u)} = \frac{\partial u_r}{\partial z} + \frac{\partial u_z}{\partial r} \tag{2.2}$$

Escribiendo lo anterior en forma compacta se obtiene

$$\{\epsilon^{(u)}\} = [L]\{u\} \tag{2.3}$$

donde $\{\epsilon^{(u)}\} = \begin{pmatrix} \epsilon_r^{(u)} & \epsilon_\theta^{(u)} & \epsilon_z^{(u)} & \gamma_{rz}^{(u)} \end{pmatrix}^T$ y $[L]$ es el operador matricial:

$$[L] \stackrel{def}{=} \begin{pmatrix} \frac{\partial}{r} & 0 \\ \frac{1}{r} & 0 \\ 0 & \frac{\partial}{\partial z} \\ \frac{\partial}{\partial z} & \frac{\partial}{\partial r} \end{pmatrix} \tag{2.4}$$

La ecuacion (2.3) es válida para cualquier relación $\epsilon - \sigma$ lineal o no lineal y es una función continua dentro del elemento.

Se denota el correspondiente esfuerzo debido a $\{\epsilon^{(u)}\}$ por $\{\sigma^{(u)}\}$ y se escribe:

$$\{\sigma^{(u)}\} \stackrel{def}{=} \{\sigma_r^{(u)} \quad \sigma_\theta^{(u)} \quad \sigma_z^{(u)} \quad \tau_{rz}^{(u)}\}^T. \tag{2.5}$$

La relación entre esfuerzo y deformación, ley elástica, es de la forma

$$\{\sigma^{(u)}\} = [D]\left(\{\epsilon^{(u)}\} - \{\epsilon^{(o)}\}\right) \tag{2.6}$$

donde en el caso de materiales elásticos isotrópicos

$$[D] = \frac{E}{(1+\nu)(1-2\nu)} \begin{pmatrix} 1-\nu & \nu & \nu & 0 \\ \nu & 1-\nu & \nu & 0 \\ \nu & \nu & 1-\nu & 0 \\ 0 & 0 & 0 & \dfrac{1-2\nu}{2} \end{pmatrix} \tag{2.7}$$

y $[D]$ es la matriz de constantes del material, la cual es simétrica y semidefinida positiva, $\{\epsilon^{(o)}\}$ es la deformación inicial, E: es el módulo de elasticidad de Young y ν : es el coeficiente de Poisson.

Las ecuaciones de equilibrio están dadas por:

$$\frac{\partial\, \sigma_r^{(u)}}{\partial\, r} + \frac{\partial\, \tau_{rz}^{(u)}}{\partial\, z} + \frac{1}{r}\left(\sigma_r^{(u)} - \sigma_\theta^{(u)}\right) + F_r = 0 \tag{2.8}$$

$$\frac{\partial\, \tau_{rz}^{(u)}}{\partial r} + \frac{\partial\, \sigma_z^{(u)}}{\partial z} + \frac{\tau_{rz}^{(u)}}{r} + F_z = 0 \tag{2.9}$$

.

donde $F_r = F_r(r,z)$, $F_z = F_z(r,z)$ son las componentes de las fuerzas del cuerpo en las direcciones radial y axial, respectivamente. Ahora se multiplica (2.8) por una función de **desplazamiento virtual** v_r y (2.9) por v_z, componentes de un vector de desplazamiento v definido similarmente al vector u en la ecuación (2.1) y cuyas propiedades se establecerán mas adelante. Si se integra sobre el volumen Ω se tiene:

$$\int_\Omega \left(\frac{\partial\, \sigma_r^{(u)}}{\partial\, r} + \frac{\partial\, \tau_{rz}^{(u)}}{\partial\, z} + \frac{\sigma_r^{(u)} - \sigma_\theta^{(u)}}{r} + F_r\right) v_r\, r\, dr\; d\theta\; dz +$$

$$\int_\Omega \left(\frac{\partial\, \tau_{rz}^{(u)}}{\partial r} + \frac{\partial\, \sigma_z^{(u)}}{\partial z} + \frac{\tau_{rz}^{(u)}}{r} + F_z\right) v_z\, r\, dr\, d\theta\, dz = 0 \tag{2.10}$$

.

Como Ω es la intersección del cuerpo axi-simétrico con cualquier plano coordenado para el cual θ es constante, se puede considerar que el integrando es independiente de θ y se obtiene:

$$\int_{\Omega} \left\{ \left(\frac{\partial}{\partial r}(r\sigma_r^{(u)}) + r\frac{\partial \tau_{rz}^{(u)}}{\partial z} - \sigma_\theta^{(u)} + rF_r \right) v_r + \left(\frac{\partial}{\partial r}(r\tau_{rz}^{(u)}) + r\frac{\partial \sigma_z^{(u)}}{\partial z} + rF_z \right) v_z \right\} dr\, dz = 0 \tag{2.11}$$

Se aplica el lema de Green a la ecuación (2.11) para convertir la integral de área en una doble y una de contorno, resultando:

$$\begin{aligned} &\oint (r\tau_{rz}v_r + r\sigma_z^{(u)}v_z)\, dr + (r\sigma_r^{(u)}v_r + r\tau_{rz}^{(u)}v_z)\, dz \\ &- \iint_{\Omega} (r\sigma_r^{(u)}\frac{\partial\, v_r}{\partial\, r} + r\tau_{rz}^{(u)}\frac{\partial\, v_z}{\partial\, r} + r\tau_{rz}^{(u)}\frac{\partial\, v_r}{\partial\, z} + r\sigma_z^{(u)}\frac{\partial v_z}{\partial z} - \sigma_\theta^{(u)}v_r)\, dr\, dz \qquad + \\ &\iint_{\Omega} (F_r v_r + F_z v_z)\, r\, dr\, dz = 0 \end{aligned} \tag{2.12}$$

Ahora se analiza la frontera

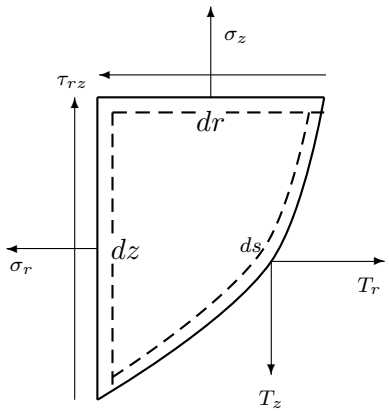


Figura 2.1: Frontera

Se tiene que:

$$\begin{cases} T_r.ds = \sigma_r.dz + \tau_{rz}.dr \\ T_z.ds = \tau_{rz}.dz + \sigma_z.dr \end{cases} \tag{2.13}$$

Si se expresa $\dfrac{\partial v_i}{\partial j} = \epsilon_k^v$ y aplicando la ley elástica (2.6)

$$\{\epsilon^{(v)}\}^T\{\sigma^{(u)}\} = ([L]\{v\})^T\,[D][L]\{u\} - ([D]\{v\})^T\,E\{\epsilon^o\} \tag{2.14}$$

se obtiene que:

$$\iint_{\Omega} \left(\sigma_r^{(u)}\epsilon_r^{(v)} + \sigma_\theta^{(u)}\epsilon_\theta^{(v)} + \sigma_z^{(u)}\epsilon_z^{(v)} + \tau_{rz}^{(u)}\gamma_{rz}^{(v)}\right) r\, dr\, dz =$$

$$\iint_{\Omega} (F_r v_r + F_z v_z)\, r\, dr\, dz + \int_{\partial\Omega} (T_r v_r + T_z v_z)\, r\, ds + \iint_{\Omega} ([L]\{v\})^T [D]\{\epsilon^o\}$$

La expresión anterior es entonces equivalente a:

$$\mathscr{B}(u, v) = \mathscr{F}(v) \tag{2.15}$$

Donde:

$$\mathscr{B}(u, v) \stackrel{def}{=} \iint_{\Omega} \left(\sigma_r^{(u)}\epsilon_r^{(v)} + \sigma_\theta^{(u)}\epsilon_\theta^{(v)} + \sigma_z^{(u)}\epsilon_z^{(v)} + \tau_{rz}^{(u)}\gamma_{rz}^{(v)}\right) r\, dr\, dz \tag{2.16}$$

equivalente a:

$$\mathscr{B}(u, v) \stackrel{def}{=} \iint_{\Omega} ([L]\{v\})^T\, [D]([L]\{u\}) r\, dr\, dz \tag{2.17}$$

Y cuya interpretación fisica es el trabajo virtual de los esfuerzos que se corresponde con los desplazamientos elásticos $\{u\}$ debido al desplazamiento virtual $\{v\}$ en un sector de un radian del cuerpo axi-simétrico.
y

$$\mathscr{F}(v) \stackrel{def}{=} \iint_{\Omega} (F_r v_r + F_z v_z)\, r\, dr\, dz + \int_{\partial\Omega} (T_r v_r + T_z v_z)\, r\, ds + \iint_{\Omega} ([L]\{v\})^T [D]\{\epsilon^o\} \tag{2.18}$$

y cuya interpretación fisica es el trabajo virtual de las fuerzas de gravedad y de las tracciones sobre un sector de un radian del cuerpo axi-simétrico.

Ahora se define la energía de deformación del cuerpo $\mathscr{U}(u)$ como:

Definición 2.1 *: Energía de deformación.*

$$\mathscr{U}(u) = \frac{1}{2}\mathscr{B}(u, u). \tag{2.19}$$

y el conjunto de todas las funciones que tienen energía de deformación finita en Ω denotado por $E(\Omega)$ se llama espacio de energía y se define como:

Definición 2.2 *: Espacio de energía.*

$$E(\Omega) = \{u / \mathscr{U}(u) < \infty\} \tag{2.20}$$

Esto indica que cualquier desplazamiento específico pertenece a $E(\Omega)$ y se escribe $\{u\} \in E(\Omega)$. Se asocia con $E(\Omega)$ la norma de energía $||u||_{E(\Omega)}$ definida como

Definición 2.3 *:Norma de energía.*

$$||u||_{E(\Omega)} \stackrel{def}{=} \sqrt{\mathscr{U}(u)} \tag{2.21}$$

$E(\Omega)$ dotado de la norma $||.||_{E(\Omega)}$ es un espacio lineal normado, ver apendice C.

Las funciones que caen en $E(\Omega)$, no tienen que ser continuas cuando Ω no es unidimensional. Ahora bien, el conjunto de las funciones de desplazamiento que está en $E(\Omega)$ y satisface las condiciones de frontera cinemáticas prescritas, se llaman funciones de desplazamiento cinemáticamente admisibles y se denota por $\widetilde{E}(\Omega)$.

Definición 2.4 *Se define el conjunto de funciones que tienen energía de deformación finita en Ω y nula en la frontera, donde las condiciones cinemáticas son prescritas, por $\overset{o}{E}(\Omega)$.*

Si no se prescriben condiciones cinemáticas de frontera, entonces $\overset{o}{E}(\Omega) = E(\Omega)$.
Se considera las componentes normal y tangencial de las funciones u y v en la frontera como:

$$\begin{cases} u_n(r,z) = v_n(r,z) = 0, & (r,z) \in \Gamma_D^n \\ u_t(r,z) = v_t(r,z) = 0, & (r,z) \in \Gamma_D^T \end{cases} \tag{2.22}$$

Cualquier función de desplazamiento $\widetilde{u}$ es admisible si:

a) La energía de deformación de $\{\widetilde{u}\}$ es finita y

b) $\widetilde{u}_n$ es igual al desplazamiento normal impuesto en Γ_D^n y $\{\widetilde{u}_t\}$ es igual al desplazamiento tangencial impuesto en Γ_D^T.

y puede ser escrita como:

$$\{\widetilde{u}\} = \{\widetilde{u}^*\} + \{u\} \tag{2.23}$$

donde $\{\widetilde{u}^*\}$ es una función de desplazamiento admisible fija arbitraria y $\{u\}$ es una función seleccionada apropiadamente de $\overset{o}{E}(\Omega)$.

Definición 2.5 *La solución exacta obtenida a través de la aplicación del PTV, y denotada por U_{Ex} es definida por:*

$$\{U_{Ex}\} = \{\widetilde{u}^*\} + \{u_o\} \tag{2.24}$$

donde $\{u_o\}$ es la función de desplazamiento en $\overset{o}{E}(\Omega)$; que satisface:

$$\mathscr{B}(u_o, v) = \mathscr{F}(v) - \mathscr{B}(\widetilde{u}^*, v) \qquad \forall \{v\} \in \overset{o}{E}(\Omega) \tag{2.25}$$

Note que los espacios prueba y ensayo son el mismo y que $\{U_{Ex}\}$ es independiente de la escogencia de $\{\widetilde{u}^*\}$ y minimiza la energía potencial con respecto a todas la funciones de desplazamiento admisibles.

2.2 Unicidad de la Solución Generalizada

Una de las características importantes del PTV para los análisis en ingeniería, es la existencia de una solución exacta y única. se ve ahora que la solución es realmente única. La prueba de existencia requiere algunas consideraciones topológicas y no se presentará acá, pero puede ser consultado en [12].

De la ecuación (2.15). Se supone que existen dos soluciones diferentes u_1 y u_2, por lo tanto

$$\mathscr{B}(u_1, v) = \mathscr{F}(v) \qquad \forall v \in \overset{o}{E}(\Omega) \tag{2.26}$$

y

$$\mathscr{B}(u_2, v) = \mathscr{F}(v) \qquad \forall v \in \overset{o}{E}(\Omega) \tag{2.27}$$

Restando las ecuaciones anteriores se tiene que

$$\mathscr{B}(u_1, v) - \mathscr{B}(u_2, v) = 0$$

y aplicando las propiedades de la función bilineal, ver apendice A, se tiene que $\mathscr{B}(u_1 - u_2, v) = 0 \qquad \forall v \in E(\Omega)$ y como $u_1 - u_2 \in E(\Omega)$ se puede escoger $v = u_1 - u_2$ y obtener que $\mathscr{B}(u_1 - u_2, u_1 - u_2) = 0$ lo que significa que $||u_1 - u_2||_{E(\Omega)} = 0$, lo cual contradice el supuesto de que u_1 y u_2 son diferentes.

CAPÍTULO 3

Acercamiento al Método de los Elementos Finitos [MEF]

En el capítulo anterior se estableció el PTV en coordenadas cilíndricas como una solución generalizada del problema elástico de un cuerpo axi-simétrico (plato) sometido a fuerzas externas. En este cápítulo se explicará como resolverlo mediante la discretización por elementos finitos. Antes de ello, se ve un breve resumen historico del MEF.

El MEF tuvo su primera contribución matemática con el artículo de Courant (1943). El método llegó a ser popular en los sesenta cuando ingenieros independientemente lo desarrollaron y nombraron MEF; el artículo de Babuska y Aziz (1972) brindó una amplia fundamentación para muchas herramientas de análisis funcional, y el primer libro en este campo fué escrito por Strang y Fix en 1973. Paralelamente al desarrollo matemático, el método de los elementos finitos se establece como técnica computacional en análisis mecánico de estructuras, cuyo desarrollo empezó alrededor de 1956, con los artículos de Turner, Clough, Martin y Topp (1956) y Argyris en 1957; el libro de Zienkiewicz en 1971 también tuvo gran impacto. Este tema puede profundizarse en [38], [34], [7], [3], [11], [25] y [32]

3.1 Solución por Elementos Finitos

Como se vió en el capítulo anterior, la solución exacta U_{Ex} que se corresponde con la formulación del trabajo virtual, está en $\widetilde{E}(\Omega)$ y satisface la ecuación (2.15). Se considera un subespacio S de $E(\Omega)$ que contenga un número finito de funciones linealmente independientes. Como $E(\Omega)$ es un espacio lineal normado, también lo es S y por definición, ver apendice C, S es de dimensión n y existen n elementos linealmente independientes en S: $\phi_i(r,z), i = 1, 2, \cdots, n$ llamadas funciones base. Cualquier función $\{u\}$ se puede escribir como la ecuación (2.1), donde:

$$\begin{aligned} u_r(r,z) &= \sum_{i=1}^{n} a_i \phi_i(r,z) \\ u_z(r,z) &= \sum_{i=1}^{n} a_{n+i} \phi_i(r,z) \end{aligned} \tag{3.1}$$

$a_i, i = 1, 2, \cdots .2n$ son llamadas las amplitudes de las funciones base ϕ_i, las ecuaciones anteriores se puden escribir en forma matricial así:

$$\{u\} = [\phi]\{a\} \tag{3.2}$$

donde

$$[\phi] = \begin{bmatrix} \phi_1 & \phi_2 & \cdots & \phi_n & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & \phi_1 & \phi_2 & \cdots & \phi_n \end{bmatrix}$$

y

$$\{a\} = \{a_1\, a_2\, \cdots a_n\, a_{n+1} \cdots\, a_{2n}\}^T$$

la columna i de la matriz $[\phi]$ es $\{\phi_i\} = \left\{ \begin{array}{c} \phi_i \\ 0 \end{array} \right\}\ i = 1, \cdots, n$

y $\{\phi_i\} = \begin{pmatrix} 0 \\ \phi_{i-n} \end{pmatrix}\ i = n+1, n+2, \cdots, 2n$

por lo tanto $\{u\}$ puede ser escrita como

$$\{u\} = \sum_{i=1}^{2n} a_i \{\phi_i\} \tag{3.3}$$

Se denota el conjunto de todas las funciones cinematicamente admisibles en S, por $\widetilde{S}$; y sea $\overset{o}{S}$ el conjunto de las funciones en S que satisfacen las condiciones de frontera pre-establecidas. Entonces $\widetilde{S} = S \bigcap \widetilde{E}(\Omega)$ y $\overset{o}{S} = S \bigcap \overset{o}{E}(\Omega)$; $\overset{o}{S}$ es un espacio lineal normado y su dimensión N es llamado el número de grados de libertad.

Se toma una solución aproximada a U_{Ex} denotada por $U_{FE} \in \widetilde{S}$ la cual satisface la ecuación 2.15 y está definida como

$$\{U_{FE}\} = \{\widetilde{u}^*\} + \{u_s\} \tag{3.4}$$

donde $\{\widetilde{u}^*\}$ es una función fija arbitraria de $\widetilde{S}$ y $\{u_s\}$ una función de $\overset{o}{S}$ que satisface la ecuación

$$\mathscr{B}(u_s, v) = \mathscr{F}(v) - \mathscr{B}(\widetilde{u}^*, v) \quad \forall v \in \overset{o}{S} \tag{3.5}$$

$\{U_{FE}\}$ es independiente de la escogencia de $\{\widetilde{u}^* \in \widetilde{S}\}$ y minimiza la energía potencial con respecto a todas las funciones admisibles, es decir,

$$\mathscr{U}(U_{Ex} - U_{FE}) = \min_{u \in \widetilde{S}} \mathscr{U}(U_{Ex} - u) \tag{3.6}$$

Facilmente se deduce que $\mathscr{B}(u_0 - u_s, v) = 0 = \mathscr{B}(U_{Ex} - U_{FE}, v) \quad \forall v \in \overset{o}{S}$ lo que significa que el trabajo virtual del error es cero con respecto a todas las funciones de $\overset{o}{S}$.

En el método de elementos finitos, S se construye partidendo Ω en subdominios llamados elementos finitos, y las particiones que pueden ser triángulos, cuadrados, prismas, etc se llaman mallas de elementos finitos y se denota por Δ y el número de elementos que constituye esta malla se denota por $M(\Delta)$.

Definición 3.1 *: Una partición $\Delta = \{T_1, T_2, \ldots, T_m\}$ de Ω en elementos cuadriláteros se llama admisible si:*

1. $\bar{\Omega} = \cup T_i$
2. *Si $T_i \cap T_j$ consiste de exactamente un punto entonces tiene un vértice común de T_i y T_J*
3. *Si $\forall i \neq j$ $T_i \cap T_j$ consiste en más de un punto entonces tienen un lado en común.*

Hay un gran número de elementos especiales útiles para el tratamiento de sistemas de ecuaciones diferenciales, ver capitulos 3 y 4 de [7]. El uso de elementos triangulares o rectangulares, depende inicialmente de la forma del dominio. Los triángulos son más flexibles, pero en mecánica de sólidos se prefieren los elementos rectangulares. Para problemas con comportamiento suave, generalmente se obtienen mejores resultados si se utilizan elementos bicuadráticos; sin embargo llevan a tener sistemas lineales con ancho de banda muy grande que implica más trabajo en el cálculo de la matriz de rigidez. Comunmente para ahorrar tiempo de programación y obtener resultados en forma rápida se utilizan elementos lineales, ver [7].

Las funciones base sobre Δ definidas en la ecuación (3.2) son construidas usualmente a partir de polinomios. Las funciones base de nivel elemental son creadas mapeando las funciones definidas en el elemento estándar, Ω_{st} sobre los elementos finitos definidos en Ω mediante funciones de mapeo apropiadas $Q_i (i = 1, 2, \ldots, M(\Delta))$, las funciones definidas en Ω_{st} se llaman funciones de forma y el espacio de polinomios de grado p se denota por L^p.

La calidad de la solución aproximada por el método de los elementos fintos es independiente de como se construyen las funciones de forma. No existe un conjunto óptimo de funciones de forma sin embargo se pueden tener ciertas consideraciones en su selección, para más detalles ver [38] las cuales sugieren que las funciones de forma pueden ser construidas a partir de funciones polinomiales simples que tienen propiedades de ortogonalidad, en este trabajo las funciones de forma que se utilizaron se encuentran en el apendice C .

La malla de elementos finitos, el grado polinomial de los elementos y las funciones de mapeo, constituyen la discretización. El conjunto de todas las funciones que pueden ser escritas de la

forma (3.3) constituye el espacio de elementos finitos, denotado por $S^p(\Omega, \Delta, Q)$ o simplemente S y se define como:

Definición 3.2 *: El espacio de elementos finitos $S^p(\Omega, \Delta, Q)$ está definido como el conjunto de todas las funciones $\{u\} \in E(\Omega)$y el k-ésimo elemento $u(Q_k(\xi, \eta)) \in L^{p_k}$, es decir*

$$\begin{aligned} S \stackrel{def}{=} \{\{u\}/\{u\} \in E(\Omega), u_r(Q_r^{(k)}(\xi,\eta), Q_z^{(k)}(\xi,\eta)) \in L^{Pk}, \\ u_z(Q_r^{(k)}(\xi,\eta), Q_z^{(k)}(\xi,\eta)) \in L^{Pk}, k = 1, \cdots, m(\Delta)\} \end{aligned} \tag{3.7}$$

donde $r = Q_r^{(k)}(\xi, \eta)$, $z = Q_z^{(k)}(\xi, \eta)$ son las funciones de mapeo definidas para el elemento e, las cuales mapean elementos estándar sobre el elemento finito.

Las siguientes son algunas propiedades que caracterizan un espacio de elementos finitos:

- El tipo de partición del dominio utilizado: Si todos los elementos son congruentes, se dice que es un espacio con partición regular.
- Sea P_i el conjunto de todos los polinomios de grado menor o igual a i. Si todos los polinomios de grado menor o igual a i son usados en la solución se llama espacio de elementos finitos con polinomios completo.
- Un elemento finito se dice ser C^k si esta contenido en $C^k(\Omega)$, esta es una propiedad de caracter global.

3.2 Extensiones

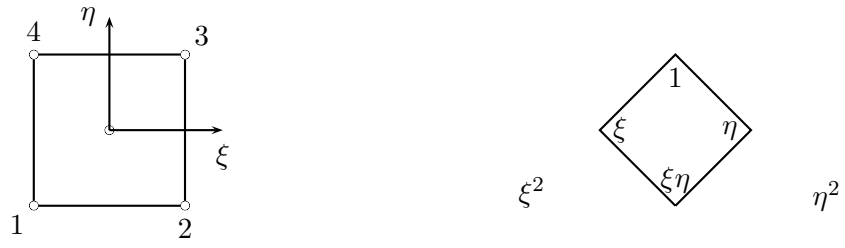
Las extensiones son los cambios sistemáticos de discretización en el cual el número de grados de libertad se incrementa en cada cambio. Más precisamente una solución de espacios de elementos finitos $S_1, S_2, \cdots$ va mejorando las soluciones. Si la extensión está basada en el refinamiento de malla se llama extensión-h, si es basada en el aumento del grado polinomial de los elementos se llama extensión-p y si es una combinación de ambas se llama extensión-hp.

3.3 Funciones de Forma Jerárquicas para Cuadriláteros

Las funciones de forma que se definen están basadas en los polinomios de Legendre y por lo tanto tienen algunas propiedades importantes como ortogonolalidad y acumulación de errores con respecto al crecimiento del grado del polinomio.

Las funciones de forma jerárquicas están definidas en un cuadrilátero estándar y se organizan en tres categorías a saber: funciones de forma en los nodos, en los lados y en los nodos internos. La figura (3.1) muestra la estructura para construir funciones de forma para elementos lineales y cuadráticos.

Elemento lineal



Elemento cuadrático

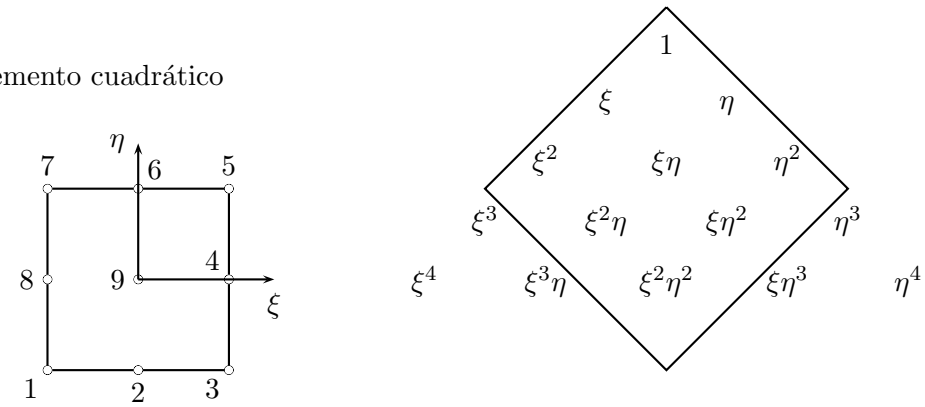


Figura 3.1: Elementos Lagrangianos. Términos polinómicos contenidos en las funciones de forma

1. Funciones de forma nodales:

$$\begin{aligned} N_1(\xi,\eta) &= \frac{1}{4}(1-\xi)(1-\eta) \\ N_2(\xi,\eta) &= \frac{1}{4}(1+\xi)(1-\eta) \\ N_3(\xi,\eta) &= \frac{1}{4}(1+\xi)(1+\eta) \\ N_4(\xi,\eta) &= \frac{1}{4}(1-\xi)(1+\eta) \end{aligned} \tag{3.8}$$

2. Funciones de forma de los lados: Hay $4(p-1)$ funciones de forma asociadas con los lados de un elemento finito ($p \geq 2$). La forma de las funciones asociadas con el lado 1 son:

$$N_i^1 = \frac{1}{2}(1-\eta)\phi_i(\xi),\ i = 2, 3, \cdots, p \tag{3.9}$$

   Con el lado 2 de la forma $N_i^2 = \frac{1}{2}(1+\xi)\phi_i(\eta)$, donde $i = 2, 3, \cdots, p$, $\phi_i(\xi)$ está definido en términos de los polinomios de Legendre P_{j-1}, ver [38].

3. Funciones de forma internas: Para el espacio $L^p(\Omega_{st}^{(q)})$ hay $(p-2)(p-3)/2$ funciones de

formas internas $p \geq 4$, las cuales se pueden escribir como:

$$N_1^0 = \phi_2(\xi)\phi_2(\eta),\ N_2^0 = \phi_3(\xi)\phi_2(\eta),\ N_3^0 = \phi_2(\xi)\phi_3(\eta),\ etc \tag{3.10}$$

Que se obtienen usando el triángulo de Pascal, ver figura (3.1).

El desarrollo de las funciones de forma utilizadas en la modelación se encuentran en el apendice C

3.4 Funciones de Mapeo

Cuando todos los lados de los elementos transformados son líneas rectas, generalmente se utiliza un mapeo lineal y es muy común en problemas unidimensionales. En dos dimensiones la frontera del dominio de la solución consiste de curvas suaves por tramos, y en estos casos se deben considerar ciertos tipos de mapeo no lineal con el fin de mantener la aproximación lo más cercano a las curvas.

En extensiones-p, muchos elementos permanecen grandes por lo tanto es importante usar una técnica de mapeo precisa. En el caso de extensiones-h, el diámetro del elemento más grande tiende a cero a medida que se aumentan los grados de libertad y por lo tanto se incrementa la precisión de la representación de la frontera.

Para el caso de cuadriláteros lineales las funciones de mapeo para el k-ésimo elemento se representan como

$$r = Q_r^{(k)}(\xi,\eta) = \sum_{i=1}^{n} R_i N_i \qquad z = Q_z^{(k)}(\xi,\eta) = \sum_{i=1}^{n} Z_i N_i \tag{3.11}$$

donde R_i y Z_i son las coordenadas en los nodos y N_i son las funciones de forma dadas en la sección anterior. Cuando $p = 2$ y se usan las funciones de forma convencionales, entonces las componentes del vector desplazamiento y el mapeo son representados por las mismas funciones de forma y por ésta razón este mapeo es llamado *isoparamétrico*, el mapeo isoparamétrico en conjunto con las extensiones-h, se usan ampliamente en los programas computacionales de elementos finitos; la precisión del mapeo isoparamétrico incrementa con el número de elementos.

CAPÍTULO 4

Modelación del Problema

Como se vió en la descripción del problema, el objeto de trabajo es un plato cerámico que se obtiene geométricamente como un sólido de revolución, por lo tanto puede modelarse bajo el enfoque axi-simétrico en coordenadas cilíndricas. En este capítulo se modela los dos procesos claves que determinan la geometría final del plato después de ser sometido a la cocción como son: la **sinterización** y la **deformación mecánica** tanto elástica como de fluencia (creep). En ambos problemas se consideran condiciones isotrópicas y densidad homogénea en cada estado. Los métodos numéricos empleados en este capítulo pueden consultarse en las referencias [5], [10], [12], [25], [32], [34], [38], y [29].

4.1 Sinterización

El procedimiento para determinar la geometría final del plato sinterizado se basa en el principio de conservación de la masa (PCM) durante el proceso de sinterización. La adaptación específica al plato se hizo a partir del documento de Sanliturk, [36] y se puede ver en el artículo publicado por los autores de este trabajo (Arango) [1].

4.1.1 Determinación de los Elementos Másicos Individuales

El principio de conservación de la masa durante el proceso de sinterización, puede escribirse en forma integral de la siguiente forma:

$$\int_{V_c} \rho_c \, dv \; = \int_{V_s} \rho_s \, dv \tag{4.1}$$

Donde: ρ_c es la densidad del cuerpo en crudo, ρ_s es la densidad del cuerpo sinterizado, V_c es el volumen del plato en crudo y V_s es el volumen del plato sinterizado.
Cuando se divide el cuerpo en elementos finitos se cumple el PCM en cada elemento y así se

puede escribir la masa total de la pieza como una suma de las contribuciones individuales, así:

$$\sum_{e=1}^{L} \int_{V_c} \rho_c \, dv \; = \sum_{e=1}^{L} \int_{V_s} \rho_s \, dv \tag{4.2}$$

Donde: L es el número total de elementos en el proceso de discretización y e es el índice del elemento.
Debido a que la densidad es uniforme tanto para el plato crudo como sinterizado, la ecuación (4.2) se transforma en:

$$\rho_c \sum_{e=1}^{L} \int_{V_c^e} dv \; = \rho_s \sum_{e=1}^{L} \int_{V_s^e} dv \tag{4.3}$$

Como el PCM también se cumple en cada elemento

$$m_e = \rho_c \int_{V_c^e} dv \; = \rho_s \int_{V_s^e} dv \tag{4.4}$$

donde $e = 1, 2, ...L$ y m_e es la masa del elemento.
Expresando el diferencial de volumen dv en coordenadas cilíndricas r, θ, z y teniendo en cuenta la simetría axial del plato se obtiene:

$$m_e = \rho_c \int_{V_c^e} r \, dr \, d\theta \, dz = 2\pi\rho_c \int_{A_c^e} r \, dr \, dz \tag{4.5}$$

Donde A_c^e es el área en la sección axi-simétrica del plato del elemento e (con $e = 1, 2, \ldots, L$). Para calcular el valor de la masa expresada en la ecuación (4.5) se procede a realizar una transformación del sistema de coordenadas cilíndricas (r, z) a un sistema de coordenadas naturales (ξ, η), ver la figura (4.1).

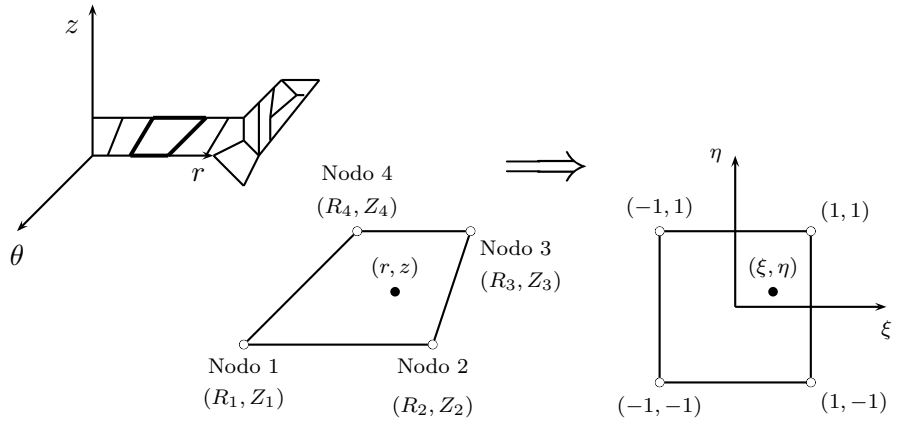


Figura 4.1: Elementos de referencia y sistemas cilíndrico y natural de coordenadas

La integración en el elemento estándar se expresa como

$$m_e = 2\pi\rho_c \int_{-1}^{1}\int_{-1}^{1} r\left(Q_r^e(\xi,\eta)\right)|J|\,d\xi\,d\eta \tag{4.6}$$

donde $r = Q_r^e(\xi,\eta)$ y $z = Q_r^e(\xi,\eta)$ son las expresiones que representan la transformación de coordenadas (r,z) del sistema cilíndrico al sistema de coordenadas naturales (ξ,η) y $|J|$ es el determinante de la matriz Jacobiana de la transformación definida por:

$$J = \begin{pmatrix} \dfrac{\partial r}{\partial \xi} & \dfrac{\partial z}{\partial \xi} \\ \dfrac{\partial r}{\partial \eta} & \dfrac{\partial z}{\partial \eta} \end{pmatrix} \tag{4.7}$$

Para calcular la integral (4.6) se utilizará la cuadratura de Gauss-Legendre descrita en el apéndice D.
Para un punto fijo η se puede integrar numéricamente con respecto a ξ y se obtiene

$$m_e \approx 2\pi\rho_c \int_{-1}^{1} \left[\sum_{s=1}^{m} w_s \left(r(Q_r^e(\xi,\eta))\ |J|\right]_{\xi=\xi_s} d\eta$$

donde w_s y ξ_s son los pesos y valores de la coordenada ξ en los puntos de Gauss y m es el número de puntos de Gauss en la dirección de integración ξ.
Integrando numéricamente nuevamente respecto a η se obtiene:

$$m_e \approx 2\pi\rho_c \sum_{r=1}^{n}\sum_{s=1}^{m} w_r w_s \left[r(Q_r^e(\xi,\eta)\,|J|\right]_{\substack{\xi\,=\,\xi_s \\ \eta\,=\,\eta_r}} \tag{4.8}$$

donde n es la cantidad de puntos de Gauss en la dirección de integración η. Comúnmente $m = n$.
La transformación de r y z se hace mediante el uso de las funciones de forma N_i, ver apendice C, y las coordenadas nodales (R_i, Z_i) del elemento i, así:

$$r = \sum_{i=1}^{N_n} N_i R_i, \qquad z = \sum_{i=1}^{N_n} N_i Z_i \tag{4.9}$$

donde N_n es el número total de nodos del elemento.
Finalmente, se puede expresar la ecuación (4.8) en forma compacta como una función del vector de coordenadas nodales de la pieza cruda $\{r_c\}$.

$$m_e = 2\pi\rho_c f(\{r_c\}) \tag{4.10}$$

Análogamente a la ecuación (4.10), se puede establecer que la masa de un elemento sinterizado es una función del vector $\{r_s\}$ de coordenadas nodales del elemento sinterizado.

$$m_e \approx 2\pi\rho_s f(\{r_s\}) \tag{4.11}$$

4.1.2 Determinación de la Geometría Final del Plato

Antes del proceso de sinterización, pueden producirse cambios en la masa del cuerpo, debido a la presencia de material orgánico proveniente de las materias primas o del uso de ligantes orgánicos necesarios para el proceso de formación por prensado con pasta granulada. Estos materiales se degradan antes de los 1.000°C, aspecto que debe tenerse en cuenta en la aplicación del principio de conservación de la masa. En el caso de estudio, las pérdidas de masa por ignición son del 8 %, debido al uso de los ligantes PVA (polivinil alcohol) y AC95, y la descomposición del agua de constitución. Esta pérdida de masa no afecta el proceso de sinterización como tal, porque las reacciones de descomposición se dan antes de los 1.000°C y la sinterización ocurre entre los 1.000 y 1.200°C.
Para determinar la geometría final del sinterizado, se puede obtener una solución aproximada contrayendo el volumen de cada elemento hasta que el producto de su volumen por su densidad dé la masa del elemento inicial afectada por sus pérdidas por ignición. En la práctica la contracción del volumen es una función continua dentro del cuerpo, pero el modelo propuesto supone que es constante para cada elemento individual, sin embargo, puede variar de elemento a elemento de forma que al ir refinando la malla se tenderá a simular el continuo. Esta simplificación es necesaria para el problema y la solución. Basados en esta suposición, la contracción del elemento individual puede ser simulada multiplicando las coordenadas nodales del elemento en crudo por una constante de proporcionalidad y encontrando su nueva localización. Repitiendo este procedimiento para todos los elementos, uno a la vez, se puede obtener la geometría final del plato sinterizado.
Este proceso puede ser implementado numéricamente así: los elementos son ordenados secuencialmente tomando un punto de referencia (el centro radial del plato con coordenadas $r = 0$ y $z = 0$). Los puntos nodales de cada elemento se mueven a su nueva posición sinterizada $\{r_s\}_e$ por medio de la siguiente ecuación:

$$\{r_s\}_e = \begin{pmatrix} \beta\, r_a \\ r_b \end{pmatrix}_e = \begin{pmatrix} \beta\, r_{a_1} \\ \beta\, r_{a_2} \\ \vdots \\ \\ r_{b_1} \\ r_{b_2} \\ \vdots \end{pmatrix}_e \tag{4.12}$$

donde el vector $\{r_a\}$ contiene las coordenadas de los nodos que no se han movido en la sinterización y $\{r_b\}$ contiene las demás coordenadas que han sido procesadas; β es una constante de proporcionalidad a determinar durante la iteración. La constante de proporcionalidad se determina usando el principio de conservación de la masa para cada elemento, mediante la siguiente ecuación:

$$m_e = 2\pi\, \rho_s\, f \begin{pmatrix} \beta\{r_a\} \\ \{r_b\} \end{pmatrix}_e \tag{4.13}$$

m_e es un valor conocido, pues se calcula usando la densidad del plato crudo y la malla de elementos finitos, por tanto, sólo se necesita calcular la constante β. Luego el problema de

encontrar la forma de la pieza sinterizada se convierte en calcular la constante β para los elementos individuales. Note, sin embargo, que en la ecuación (4.11), la función f denota el volumen de un elemento, conociendo el vector de las coordenadas nodales y es una función no lineal de β, la cual requiere un algoritmo de solución no lineal (Newton-Raphson). Después que el valor de β se obtiene para el elemento e, los nodos del elemento adyacente son movidos a su posición sinterizada y el mismo procedimiento se repite para otros elementos. Por fin, cuando todos los elementos están procesados, la localización final de los nodos de la malla de elementos finitos proporciona la geometría sinterizada del plato cerámico completo.
Para explicar el algoritmo se puede observar en la figura (4.2) una malla simple con 4 elementos en dos dimensiones y 9 nodos. Se observa que la posición sinterizada del elemento (1) es calculada y los nodos 1, 4, 5 y 2 movidos a su nueva localización formando el vector $\{r_a\}$. Para el segundo elemento las coordenadas de los nodos 5 y 2, que ya se han movido, forman el vector $\{r_b\}$ y no cambian, sin embargo, las coordenadas de los nodos 3 y 6 se multiplican por la constante de proporcionalidad, calculando su nueva posición sinterizada, el mismo procedimiento se sigue para los elementos 3 y 4 hasta obtener la forma final del plato.

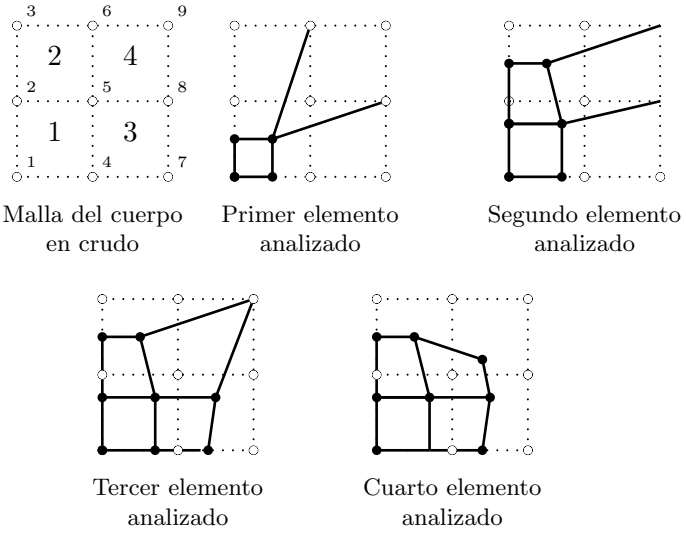


Figura 4.2: Ilustración del algoritmo de sinterización

4.1.3 Proceso de Solución

El diagrama de flujo que se muestra en la figura (4.3) describe el procedimiento general utilizado para la predicción de la geometría del sinterizado. Con los datos de entrada se calcula la masa de los elementos mediante integración numérica, usando la densidad en crudo y la geométria del plato, luego se forman los vectores $\{r_a\}$ y $\{r_b\}$ para calcular β a traves de una solución iterada, mediante el método de Newton-Raphson, que lo que busca es encontrar las raíces de la ecuación:

$$g(\beta_e) = m_e - \rho_s f \begin{pmatrix} \beta\{r_a\} \\ \{r_b\} \end{pmatrix} = 0 \tag{4.14}$$

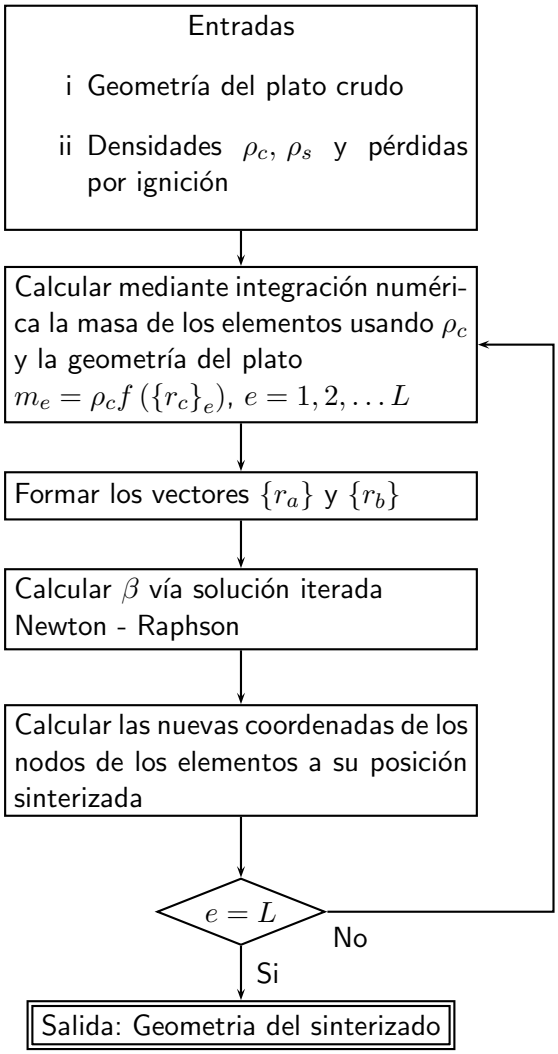


Figura 4.3: Predicción de la geometría del sinterizado

para cada elemento. Una vez encontrado β se calculan las coordenadas finales para obtener la geométria del sinterizado.

4.2 Comportamiento Mecánico Elástico

Se analiza un elemento del plato, ver figura (4.4), sea (r, θ, z) el sistema coordenado y u, v los desplazamientos real y virtual de los puntos.
El problema básico es la deformación elástica de un cuerpo sometido a su propio peso a una temperatura T. Los subindices r y z denotan las direcciones radial y axial respectivamente y el superindice u, denota la deformación elástica que está definida por la ecuación (2.1).

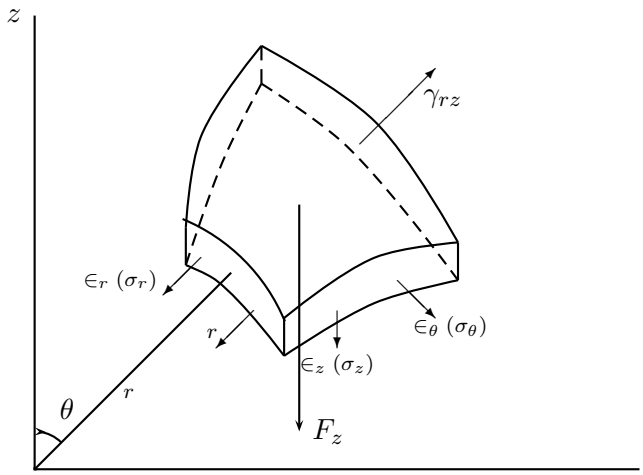


Figura 4.4: Balance de fuerzas

Las ecuaciones de equilibrio que rigen este proceso son:

$$\frac{\partial \sigma_r^{(u)}}{\partial r} + \frac{\partial \tau_{rz}^{(u)}}{\partial z} + \frac{1}{r}\left(\sigma_r^{(u)} - \sigma_\theta^{(u)}\right) = 0 \tag{4.15}$$

$$\frac{\partial \tau_{rz}^{(u)}}{\partial r} + \frac{\partial \sigma_z^{(u)}}{\partial z} + \frac{\tau_{rz}^{(u)}}{r} + F_z = 0 \tag{4.16}$$

donde $F_z = F_z(r, z)$ es la fuerza axial debida al peso del elemento.
El problema tiene como restricciones, ver figura (4.5):

$$\begin{cases} r = 0, & u_r = 0 \\ (r_p, z_p) & u_z = 0 \end{cases} \tag{4.17}$$

Donde p denota el punto de apoyo del plato en el refractario que actúa como soporte durante el proceso de cocción (también llamado peana).

Ahora, aplicando el principio del trabajo virtual (PTV)(ver la sección (2.1)) se obtiene una forma bilineal $\mathscr{B}$ y un funcional lineal $\mathscr{F}$, dados por:

$$\mathscr{B}(u,v) = \iint_{\Omega} \left[([L]\{v\})^T [D][L]\{u\} \right] r\, dr\, dz \tag{4.18}$$

$$\mathscr{F}(v) = \iint_{\Omega} r F_z v_z\, dr\, dz + \int_{\partial\Omega} (T_r v_r + T_z v_z)\, r\, ds \tag{4.19}$$

que constituyen el sistema:

$$\mathscr{B}(u,v) = \mathscr{F}(v)$$

4.2.1 Determinación de las Condiciones de Frontera

A continuación se ve esquematicamente un semiplano del plato en estudio:

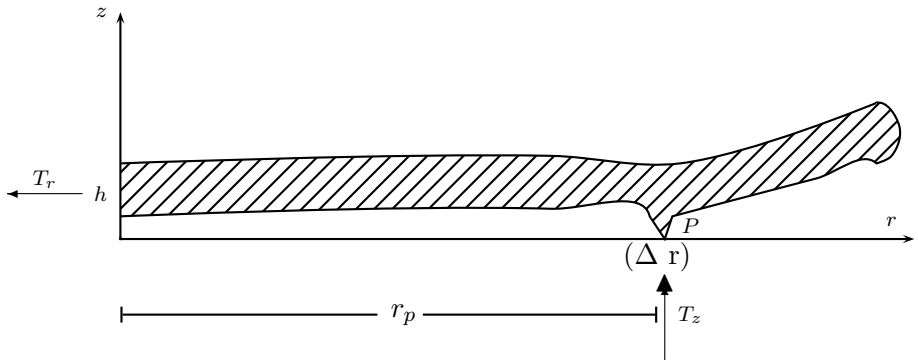


Figura 4.5: Punto de apoyo de un plato

Se considera el punto de apoyo p en la figura (4.5) como una vecindad entre $\left(r_P - \frac{\Delta r}{2}, r_P + \frac{\Delta r}{2}\right)$, donde Δr es un delta pequeño del radio.
Sea R la fuerza de reacción al peso por unidad de longitud. Entonces se tiene:

$$T_z = \begin{cases} 0 & 0 < r < r_P - \frac{\Delta r}{2} \\ \dfrac{R}{\Delta r} & r_P - \frac{\Delta r}{2} < r_p < r_P + \frac{\Delta r}{2} \\ 0 & r_P + \frac{\Delta r}{2} < r < \infty \end{cases} \tag{4.20}$$

y considere la frontera h, donde:

$$T_r = \begin{cases} 0 & r \neq 0 \\ T_r & r = 0 \end{cases} \tag{4.21}$$

Si se aplica las restricciones (4.20) y (4.21) a la integral de línea de la ecuación (4.19) se tiene que:

$$\int_{\partial\Omega} T_z\, v_r\, r\, ds = \lim_{\Delta r\to 0} \int_{r_p-\frac{\Delta r}{2}}^{r_p+\frac{\Delta r}{2}} T_z\, v_z\, r\, ds = \lim_{\Delta r\to 0} \int_{r_p-\frac{\Delta r}{2}}^{r_p+\frac{\Delta r}{2}} \frac{R}{\Delta r} v_z r\, dr$$

$$= \lim_{\Delta r\to 0} \left(\frac{R.v_z}{\Delta r}\right)(z\,\Delta r.r_p) = 2R\, v_z\, r_p = c\, v_z \tag{4.22}$$

donde $c = 2Rr_p$.
Para la componente r de la tracción se tiene:

$$\int_{\partial\Omega} rT_r, v_r\, ds = \int_0^h T_r\, v_r\, dz = 0 \qquad \forall r \tag{4.23}$$

Finalmente, las ecuaciones (4.18) y (4.19) nos quedan:

$$\iint_\Omega [([L]\{v\})^T [D][L]\{u\}] r\, dr\, dz = c.v_z + \iint_\Omega (F_z.\, v_z) r.\, dr.\, dz \tag{4.24}$$

y ésta es la ecuación que se va ha solucionar por el método de los elementos finitos, donde $F_z = -\rho_s.g$, g es la gravedad $[m/s]$ y ρ_s es la densidad del plato sinterizado.

4.2.2 Solución del Problema Elástico Mediante Elementos Finitos

Como se vió en (2.24):

$$\{U_{Ex}\} = \{\widetilde{u}^*\} + \{u_o\}$$

donde $\{u_o\} \in \overset{o}{E}(\Omega)$ y satisface la ecuación (2.25). Así

$$\mathscr{B}(u_o, v) = \mathscr{F}(v) - \iint_\Omega [([L]\{v\})^T [D][L]\{\widetilde{u}^*\}] r\, dr\, dz \tag{4.25}$$

Como no es posible trabajar infinitamente con funciones linealmente independientes, en general no es posible hallar u_o tal que satisfaga la ecuación (4.25) y por tanto se crea un conjunto de funciones, subdividiendo Ω en un número finito de elementos y definiendo un conjunto de funciones base sobre Ω, en forma tal que cada función base es diferente de cero sobre los elementos individuales o en las vecindades de estos tal como se expresó en la ecuación (3.3).
Se quiere entonces, encontrar una solución aproximada a $\{U_{Ex}\}$, y definida en la ecuación (3.4) y para ello es necesario determinar la función $u_s \in \overset{o}{S}$ que satisface la ecuación (3.5).Esta función se puede expresar con base en la ecuación (3.2) como:

$$\{u_s\} = [\phi^o]\{a^o\} \tag{4.26}$$

Similarmente se puede expresar v como:

$$\{v\} = [\phi^o]\{b^o\} \tag{4.27}$$

Sustituyendo (4.26) y (4.27) en la parte izquierda de la ecuación (4.24) se tiene:

$$\mathscr{B}(u_s, v) =: \iint_{\Omega} \left[([L]\,[\phi^o]\,\{b^o\})^T\,[D]\,[L]\,\{\phi^o\}\{a^o\} \right] r\,dr\,dz \tag{4.28}$$

Similarmente llevando (4.27) a la componente derecha de la ecuación (4.24) se tiene que:

$$\mathscr{F}(v) = \iint_{\Omega} F_z([\phi^o]\,\{b^o\})^T\,r\,dr\,dz + c([\phi^o]\,\{b^o\})^T \tag{4.29}$$

Por definición:

$$[K] \stackrel{def}{=} \iint_{\Omega} \left[([L]\,[\phi^o])^T\,[D]\,[L]\,\{\phi^o\} \right] r\,dr\,dz \tag{4.30}$$

$$\{q\} \stackrel{def}{=} \iint_{\Omega} F_z.\,[\phi^o]^T\,r\,dr\,dz + c[\phi^o]^T \tag{4.31}$$

donde $[K]$ se llama la **matriz global de rigidez** o matriz ensamblada y $\{q\}$ es el vector global de fuerza o vector ensamblado de fuerzas.
Con esta notación se puede reescribir (4.24) como

$$\{b^o\}^T\,[[K]\{a^o\} - \{q\}] = 0 \qquad \forall\{b^o\} \tag{4.32}$$

Esta condición se satisface solo si

$$[K]\{a^o\} = \{q\} \tag{4.33}$$

y la ecuación (4.33) es el sistema a resolver para determinar los coeficientes de amplitud $\{a^o\}$. Los elementos de la matriz global de regidez $[K]$, definida en la ecuación (4.30) son de la forma:

$$K_{ij} = \iint_{\Omega} \left[([L]\,[\phi_i^o])^T\,[D]\,[L]\,\{\phi_j^o\} \right] r\,dr\,dz \tag{4.34}$$

donde $\{\phi_i\}$ son las columnas de $[\phi^o]$ y los elementos del vector global de fuerzas $\{q\}$ son de la forma:

$$q_i = \iint_{\Omega} F_z\,\{\phi_i^o\}\,r\,dr\,dz + c\{\phi_i^o\}^T \tag{4.35}$$

Cómputo de la Matriz de Rigidez

La integración de la ecuación (4.34) se desarrolla elemento por elemento asi:

$$k_{ij} = \sum_e k_{ij}^{(e)} \tag{4.36}$$

donde:

$$K_{ij}^{(e)} = \iint_{\Omega^e} \left[([L]\,[\phi_i^o])^T\,[D]\,[L]\,\{\phi_j^o\} \right] r\,dr\,dz \tag{4.37}$$

y e representa el elemento.
Las funciones elementales básicas $\phi_i^o(r,z)$, y $\phi_j^o(r,z)$ están relacionadas con las funciones de forma elementales mediante la transformación explicada en la figura (4.1), luego para cualquier elemento e se tiene:

$$N_I(\xi,\eta) = \phi_i\left(Q_r^e(\xi,\eta),\, Q_z^e(\xi,\eta)\right) \tag{4.38}$$

donde el subindice I representa el número consecutivo de la función de forma elemental y de las funciones básicas; y el subindice i representa el consecutivo de las funciones básicas definidas en Ω.
En general, la función $\phi_i(r,z)$ no se calcula, sino que se usa una transformación Se define para un elemento e:

$$\begin{cases} \{N_I\} = \begin{cases} \phi_i\left(Q_r^e(\xi,\eta),\, Q_z^e(\xi,\eta)\right) & I = 1,2,\cdots,n^{(e)} \\ 0 \end{cases} \\ \{N_I\} = \begin{cases} 0 \\ \phi_i\left(Q_r^e(\xi,\eta),\, Q_z^e(\xi,\eta)\right) & I = n^{(e)}+1,\cdots,2n^{(e)} \end{cases} \end{cases} \tag{4.39}$$

donde $n^{(e)}$ es el número de funciones de forma elementales.
Con esta notación, el cómputo de la matriz de rigidez del elemento e, involucra el cálculo de los siguientes términos:

$$K_{IJ}^{(e)} = \iint\limits_{\Omega^{st}} \left[\left(\left[L^*\right]\left\{N_I\right\}\right)^T \left[D\right]\left[L^*\right]\left\{N_J\right\}\right] |J|\, r\, d\xi\, d\eta \tag{4.40}$$

$$I, J = 1,2,\cdots,2n^{(e)}$$

donde $|J|$ es el determinante del jacobiano de la tranformación de coordenadas y $[L^*]$ se obtiene como sigue:
Aplicando directamente la regla de la cadena a N_I se tiene:

$$\begin{pmatrix} \dfrac{\partial N_I}{\partial \xi} \\ \dfrac{\partial N_I}{\partial \eta} \end{pmatrix} = \begin{pmatrix} \dfrac{\partial r}{\partial \xi} & \dfrac{\partial z}{\partial \xi} \\ \dfrac{\partial r}{\partial \eta} & \dfrac{\partial z}{\partial \eta} \end{pmatrix} \begin{pmatrix} \dfrac{\partial N_I}{\partial r} \\ \dfrac{\partial N_I}{\partial z} \end{pmatrix} \tag{4.41}$$

Por definición

$$J = \begin{pmatrix} \dfrac{\partial r}{\partial \xi} & \dfrac{\partial z}{\partial \xi} \\ \dfrac{\partial r}{\partial \eta} & \dfrac{\partial z}{\partial \eta} \end{pmatrix}$$

Sea J^* la inversa de J, luego de la ecuación (4.41) se tiene

$$\begin{pmatrix} \dfrac{\partial}{\partial r} \\ \dfrac{\partial}{\partial z} \end{pmatrix} N_I = J^* \begin{pmatrix} \dfrac{\partial}{\partial \xi} \\ \dfrac{\partial}{\partial \eta} \end{pmatrix} N_I \tag{4.42}$$

y sean las componentes de J^*, J^*_{ij}. Se verá que es $[L]\{\phi^o_i\}$

$$[L]\{\phi^o_i\} = [L]\begin{Bmatrix} N_I \\ 0 \end{Bmatrix} \qquad \forall i = 1, 2, \ldots, n \tag{4.43}$$

Aplicando la definición de $[L]$ dada en la ecuación (2.4) se llega a que:

$$[L]\{\phi^o_i\} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1/r & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{Bmatrix} \dfrac{\partial}{\partial r} \\ 1 \\ \dfrac{\partial}{\partial z} \end{Bmatrix} N_I = [L] \begin{Bmatrix} \dfrac{\partial}{\partial r} \\ 1 \\ \dfrac{\partial}{\partial z} \end{Bmatrix} N_I \tag{4.44}$$

y aplicando (4.42) se obtiene:

$$[L]\{\phi^o_i\} = [L1] \begin{pmatrix} J^*_{11} & 0 & J^*_{12} \\ 0 & 1 & 0 \\ J^*_{21} & 0 & J^*_{22} \end{pmatrix} \begin{Bmatrix} \dfrac{\partial}{\partial \xi} \\ 1 \\ \dfrac{\partial}{\partial \eta} \end{Bmatrix} N_I \tag{4.45}$$

y en forma más compacta se puede decir que:

$$[L^*] = [L1][J^*_A]\{\tfrac{\partial}{\partial \xi}\ 1\ \tfrac{\partial}{\partial \eta}\}^T \tag{4.46}$$

donde:

$$J^*_A = \begin{pmatrix} J^*_{11} & 0 & J^*_{12} \\ 0 & 1 & 0 \\ J^*_{21} & 0 & J^*_{22} \end{pmatrix}$$

Si se aplica el mismo procedimiento para $i = n+1, \ldots 2n^c$ se llega a

$$[L^*] = [L2][J^*_A]\{\tfrac{\partial}{\partial \xi}\ 1\ \tfrac{\partial}{\partial \eta}\}^T \tag{4.47}$$

con:

$$[L2] = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

ahora discretizando la ecuación (4.40) como se aplicó en el capítulo (3).
Mediante la cuadratura de Gauss-Legendre, ver apendice (D), se obtiene:

$$K^e_{ij} = \sum_{r=1}^{NG}\sum_{s=1}^{NG} w_r\, w_s \left\{ \left([L^*]\{N_I\}\right)^T [D]\, [L^*]\, \{N_J\} |J|\, r^* \right\}_{\xi=\xi_r, \eta=\eta_s} \tag{4.48}$$

donde r^* está en términos de la transformación de coordenadas, NG es el número de puntos de Gauss a utilizar en la cuadratura, W_i son los pesos de la cuadratura y, r y s son los puntos de Gauss.

Para efectos de la programación, la ecuación (4.48) es muy demandante en tiempo de computador, porque calcula y acumula el valor para cada función de forma; es mejor expresar $[L]\{\phi^o\}$ en términos de todas las funciones de forma y operar con ésta matriz que se llamará: **matriz cinemática** [mtxcin].
Por definición:

$$[mtxcin] \stackrel{def}{=} [L]\{\phi^o\} = \begin{pmatrix} \dfrac{\partial}{\partial r} & 0 \\ \dfrac{1}{r} & 0 \\ 0 & \dfrac{\partial}{\partial z} \\ \dfrac{\partial}{\partial z} & \dfrac{\partial}{\partial r} \end{pmatrix} \begin{pmatrix} N_1 & N_2 & \cdots & N_n & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & N_1 & N_2 & \cdots & N_n \end{pmatrix} \tag{4.49}$$

y llevando (4.49) a (4.48), se obtiene:

$$K_{ij}^e = \sum_{r=1}^{NG} \sum_{s=1}^{NG} w_n \, w_s \left\{ \left([mtxcin]^T \, [D] \, [mtxcin] \, |J| \, r^* \right\}_{\xi=\xi_r, \eta=\eta_s} \right. \tag{4.50}$$

En el cómputo de la matriz cinemática $[mtxcin]$, aparece el coeficiente $\frac{1}{r}$ como factor; en la cuadratura de Gauss estos puntos permanecen finitos ya que no hay valores en $r = 0$. Durante el cálculo de esfuerzos aparece la forma $\frac{0}{0}$ en la expresión $\epsilon_\theta = \frac{a}{r}$ para puntos en el eje z, esto se puede obviar, calculando ϵ_θ ligeramente lejano del eje z o extrapolando los puntos de Gauss de la deformación al eje. Otra opción es utilizar la teoria que establece que $\epsilon_\theta = \epsilon_r$ en $r = 0$.

Cómputo del Vector de Fuerzas $\{q\}$

La integración del primer miembro de la ecuación (4.35) se desarrolla elemento por elemento.

$$q_i = \sum_e q_i^e = \iint\limits_{\Omega^e} F_z . \{\phi_i^o\}^T . r \, dr \, dz \tag{4.51}$$

En este caso $F_z = \{0 \quad -\rho_s . g\}$, es decir es una fuerza de gravedad actuando sólo en la dirección z.
Además se nota que el cómputo del vector $\{q\}$ involucra el cálculo de los elementos estándar:

$$q_I^{(e)} = \iint\limits_{\Omega_{st}} \{0 \quad \rho_s g\} [N_I]^T \, |J| \, r \, d\xi \, d\eta \qquad I = 1, \cdots, 2n^{(e)} \tag{4.52}$$

y si se aplica la cuadratura de Gauss-Legendre a la ecuación (4.52) se obtiene:

$$q_I^{(e)} \approx \sum_{r=1}^{NG} \sum_{s=1}^{NG} w_n \, w_s \left\{ \{0 \quad \rho_s . g\} \{N_I\}^T \, |J| \, r^* \right\}_{\xi=\xi_r, \eta=\eta_s} \tag{4.53}$$

Cuando se hace $I = 1, \cdots, n^{(e)}$ se tiene que:

$$\{0 \quad \rho_s.g\} \begin{Bmatrix} N_I \\ 0 \end{Bmatrix} = 0 \tag{4.54}$$

y éstas son las componentes de la fuerza que se corresponden con la dirección r.
Cuando $I = n+1, \cdots, 2n^{(e)}$:

$$\{0 \quad \rho_s.g\} \begin{Bmatrix} 0 \\ N_I \end{Bmatrix} = \rho_s.g.N_I \tag{4.55}$$

que son las componentes de la fuerza que se corresponden con la dirección z. Por lo tanto el vector $\{q_i\}$ será de la forma:

$$\{0 \quad 0 \quad \ldots \quad q_1 \quad q_n\}^T$$

El segundo componente del vector de fuerzas, que es $c\{\phi^o\}$ se deja como una variable desconocida y se determina en la solución general del sistema. Para ello se aplicará el metodo de penalización ver [10].

4.2.3 Cómputo de los Desplazamientos, Deformaciones y Tensiones

Una vez se tiene la solución del sistema, el cálculo de los desplazamientos es simple y directo. Las componentes de los desplazamientos se definieron como:

$$\begin{aligned} u_r &= \sum_{i=1}^{n} a_i \phi_i(r,z) \\ u_z &= \sum_{i=1}^{n} a_{n+i} \phi_i(r,z) \end{aligned} \tag{4.56}$$

Asociados con las funciones de mapeo para el elemento e como:

$$r = Q_r^{(e)}(\xi,\eta), \qquad z = Q_z^{(e)}(\xi,\eta)$$

Las funciones básicas $\phi_i(r,z)$ están relacionadas con las funciones de forma por:

$$N_I(\xi,\eta) = \phi_i \left(Q_r^{(e)}(\xi,\eta),\, Q_z^{(e)}(\xi,\eta) \right)$$

donde I es el número de la función de forma.
Entonces se puede expresar a las componentes del desplazamiento para el elemento e como:

$$\begin{aligned} u_r &= \sum_{i=1}^{n} a_i^o N_i(\xi,\eta) \\ u_z &= \sum_{i=1}^{n} a_{i+n}^o N_i(\xi,\eta) \end{aligned} \tag{4.57}$$

donde n es el número de funciones de forma, las cuales dependen del grado polinomial.
El cálculo de las deformaciones elásticas se hace a partir de la ecuación (2.3). Al discretizar esta ecuación para el elemento e, y teniendo en cuenta la ecuación (4.57), se llega a:

$$\{\epsilon^u\}^e = \sum_{I=1}^{n} a_I^o \, [L1] \, N_J + \sum_{I=1}^{n} a_{I+n}^o \, [L2] \, N_I \tag{4.58}$$

si se quiere expresar matricialmente para todo el sistema, se puede usar la matriz cinemática definida en la ecuación (4.49) y se obtiene:

$$[E^u] = [mtxcin]\{a_i^o\} \tag{4.59}$$

El cálculo de tensiones elásticas se hace con la ecuación (2.6):

$$[\sigma^u] = [D][\epsilon^{(u)}] \tag{4.60}$$

4.2.4 Cómputo de la Norma de Energía

Como se vió en el capítulo 2, la norma de energía está definida como:

$$\|\mathscr{U}\|_{E(\Omega)} \overset{def}{=} \sqrt{\mathscr{U}(u)} \equiv \sqrt{\frac{1}{2}\mathscr{B}(u,u)} \tag{4.61}$$

Para calcular $\mathscr{B}(u,u)$ se remite a lo trabajado con la ecuación (4.28), para obtener:

$$\mathscr{B}(u,u) = \iint\limits_{\Omega} \left([L]\,[\phi^o]\,\{a^o\}\right)^T [D] \left([L]\,[\phi^o]\,[a^o]\right) \, r \, dr \, dz \tag{4.62}$$

y aplicando la definición de matriz de rigidez global del sistema, la ecuación (4.30) se tiene:

$$\mathscr{B}(u,u) = \iint\limits_{\Omega} \{a^o\}^T [K] \, \{a^o\} \, r \, dr \, dz \tag{4.63}$$

En la ecuación (4.63) se requieren calcular los componentes:

$$\mathscr{B}^e(u,u) = \iint\limits_{\Omega_{st}} \{a_I^o\}^T \, [K_{ij}^e] \, \{a_I^o\} \, r^* \, d\xi \, d\eta \tag{4.64}$$

Por la cuadratura de Gauss-Legendre se tiene:

$$\mathscr{B}^e(u,u) \approx \sum_{r=1}^{NG} \sum_{s=1}^{NG} w_r w_s \left[\{a_i^o\}^T \, [K_{ij}^e] \, \{a_i^o\}\right]_{\xi=\xi_r \, \eta=\eta_s} . \, r^* \tag{4.65}$$

y finalmente:

$$\mathscr{B}(u,u) = \sum_{e} \mathscr{B}^e(u,u). \tag{4.66}$$

4.3 Comportamiento Mecánico Tipo "Creep"

Para modelar el comportamiento de fluencia "Creep" multi-axial, se construye a partir de ensayos uni-axiales, mecanismo que se detalla a continuación.

4.3.1 Comportamiento "Creep" Uni-Axial con Carga Fija

Si se tiene un espécimen uni-axial, sometido a una carga durante un tiempo t y una temperatura T, se presentará un cambio en la geométria del espécimen, con una curva típica de deformación como se vió en el capítulo 1; esta curva está dividida en tres etapas, llamadas primaria, secundaria y terciaria.
En la primaria, se da un crecimiento de la velocidad de deformación, luego se vuelve constante o aproximadamente constante en la etapa secundaria, y posteriormente incrementa rápidamente, seguido eventualmente de una fractura del material. El comportamiento Creep puede ser representado por una variedad de relaciones, entre el esfuerzo, la deformación, el tiempo, la temperatura ente otras variables.
En general, la relación típica uni-axial bajo carga constante, esta dada por la ecuación (4.67):

$$\epsilon_c = f(\sigma, T, t) \tag{4.67}$$

donde: ϵ_c es la deformación Creep, σ es la tensión nominal, t tiempo y T es la temperatura.
Esta ecuación se puede simplificar separando los efectos como sigue:

$$\epsilon_c = f_1(\sigma).f_2(T).f_3(t) \tag{4.68}$$

Se ha probado que esta ecuación simplificada es útil en la solución de problemas ingenieriles, sin embargo, su aplicación requiere evidencia experimental.
Se han propuesto muchos modelos matemáticos para relacionar la dependencia del esfuerzo y la deformación en la etapa secundaria; entre ellos se tiene el modelo de Norton.(usado por Navarro para predecir la deformación de una baldosa cerámica), ver [31]:

$$\dot{\epsilon}_c = A\sigma^n \tag{4.69}$$

Y el modelo de Prandlt:

$$\dot{\epsilon}_c = B.\sinh(\beta\sigma) \tag{4.70}$$

donde: A, n, B y β son constantes del material.
Para la dependencia con el tiempo se tiene los modelos:

$$f_2(t) = t \qquad \text{relación lineal}$$
$$f_2(t) = b\,t^m \qquad \text{relación de Bailey,} \quad \text{con } m < 1 \text{ creep primario}$$

Y para el caso de la temperatura, la relación es del tipo:

$$f_3(t) = \exp\left(-\frac{Q}{RT}\right) \tag{4.71}$$

donde Q es la energía de activación, R es la constante de Boltzman y T la temperatura absoluta. Para nuestro caso, se toma el modelo de Norton en condiciones isotérmicas, validado con una prueba experimental, esto es:

$$\epsilon_c = A.\sigma^n t \tag{4.72}$$

donde A y n se estiman inicialmente, con base en una prueba uni-axial. Luego de realizar la prueba experimental con un barrita cerámica, ver la sección (4.5), se encontró que la ecuación que representa el comportamiento creep para nuestro plato es de la forma:

$$\epsilon_c = (A\sigma^2 + B\sigma)t \tag{4.73}$$

4.3.2 Formulación Creep Multiaxial

El desarrollo de modelos creep multiaxiales es usualmente un compromiso entre intentar ajustar el comportamiento complejo en pruebas experimentales con modelos matemáticos relativamente simples. Estas formulaciones están basadas en el criterio de esfuerzo efectivo de Von Mises y la regla de flujo de Prandtl-Reuss ver [5] y [28] con base en ellos se puede expresar la velocidad de deformación creep efectiva $\overset{\bullet}{\epsilon}{}^c_{\text{eff}}$ de la siguiente forma:

$$\overset{\bullet}{\epsilon}{}^c_{\text{eff}} = \frac{\sqrt{3}}{2}\left[(\overset{\bullet}{\epsilon}{}^c_r - \overset{\bullet}{\epsilon}{}^c_z)^2 + (\overset{\bullet}{\epsilon}{}^c_r - \overset{\bullet}{\epsilon}{}^c_\theta)^2 + (\overset{\bullet}{\epsilon}{}^c_z - \overset{\bullet}{\epsilon}{}^c_\theta)^2 + 6\,\gamma^2_{rz}\right]^{1/2} \tag{4.74}$$

y la tensión efectiva σ^c_{eff} como:

$$\sigma^c_{\text{eff}} = \frac{1}{\sqrt{2}}\left[(\sigma_r - \sigma_z)^2 + (\sigma_r - \sigma_\theta)^2 + (\sigma_z - \sigma_\theta)^2 + 6\,\gamma^2_{rz}\right]^{\frac{1}{2}} \tag{4.75}$$

La evidencia experimental indica que para pequeñas deformaciones, la deformación creep es un proceso a volumen constante.
Basados en la regla de Prandlt-Reuss se tiene que:

$$\overset{\bullet}{\epsilon}{}^c_{ij} = \frac{d(\in^c_{ij})}{dt} = \lambda.\, S_{ij} \qquad S_{ij} \text{ esfuerzos Deviatoric y} \tag{4.76}$$

λ determinada de pruebas uniaxiales experimentales:

$$\lambda = \frac{3}{2}\,\frac{\overset{\bullet}{\epsilon}{}^c_{\text{eff}}}{\sigma_{\text{eff}}} \tag{4.77}$$

Al aplicar la ley de Norton modificada a la ecuación (4.73) se tiene:

$$\epsilon^c_{\text{eff}} = \left(A\sigma^2_{\text{eff}} + B\sigma_{\text{eff}}\right)t \tag{4.78}$$

$$\overset{\bullet}{\epsilon}{}^c_{\text{eff}} = A\sigma^2_{\text{eff}} + B\sigma_{\text{eff}} \tag{4.79}$$

Finalmente con base en las ecuaciones (4.76), (4.77) y (4.79) se tiene la ley multiaxial en forma tensorial:

$$\overset{\bullet}{\epsilon}{}^c_{\text{eff}} = \frac{3}{2}\,\left[A\sigma_{\text{eff}} + B\right].\, S_{ij} \tag{4.80}$$

Los esfuerzos Deviatoric S_{ij} se calculan con base en las componentes cilíndricas del esfuerzo y el esfuerzo medio, como sigue:

$$\sigma_m = \frac{1}{3}\left(\sigma_r + \sigma_\theta + \sigma_z\right) \tag{4.81}$$

$$S_{ij} = (\sigma_i - \sigma_m) \qquad \forall\, i = j \tag{4.82}$$

$$S_{ij} = \tau_{ij} \qquad \forall\, i \neq j \tag{4.83}$$

Esta formulación multiaxial es apropiada para cuando no hay cambios de signo en los componentes cilíndricos del esfuerzo y para relaciones entre esfuerzos que no presenten grandes cambios.

4.3.3 Procedimientos de Marcha o Simulación en el Tiempo

En los problemas con deformación tipo creep, la deformación es una función del tiempo de acuerdo con la ecuación constitutiva y por lo tanto la solución se realiza mediante marchas en periodos de tiempo, dividiendo el tiempo total de exposición de la pieza cerámica al fenómeno térmico, en pequeños intervalos.
La velocidad de deformación o la velocidad del desplazamiento $\dot{u}$, se asume como una función del desplazamiento actual (u) y del tiempo (t) como sigue:

$$\dot{u} = \frac{du}{dt} = f(u,t) \tag{4.84}$$

Hay muchos métodos numéricos establecidos que se pueden utilizar para dar marcha a la simulación, entre ellos se elige el método theta, o método de la diferencia hacia atras, ya que es un acercamiento mucho más general que los otros (como el método de Euler o el método implícito) y para valores de $\theta > 0.5$ es incondicionalmente estable para todos los valores de Δt utilizados, sin embargo, pasos pequeños de tiempo garantizan más precisión en los resultados ver [5]. Para validar la programación del algoritmo basado en este método, se realizó una simulación con un plato enmallado con 20 elementos y grado polinomial 2. Se realizaron entonces simulaciones variando el parametro θ desde 0.4 hasta 1.0, y graficando el desplazamiento u del punto cero, que se corresponde con el centro inferior del plato. Ver figura (4.6). En estas simulaciones se aprecia claramente la estabilidad de la solución a partir de $\theta = 0.5$ y cómo mejora al aumentar θ. Se ve inestabilidad para $\theta = 0.4$.

En este método para obtener la solución del desplazamiento al final del intervalo de tiempo $(\Delta t,\, U_{t+\Delta t})$, se usan las pendientes de un punto inicial U_t y de un punto intermedio $U_{t+\theta.\Delta t}$ como sigue:

$$U_{t+\Delta t} = U_t + \Delta t\left(f(U_{t+\Delta t}, t + \theta.\,\Delta t)\right) \tag{4.85}$$

donde:

$$U_{t+\Delta t} = \theta.\, U_{t+\Delta t} + (1-\theta)U_t \tag{4.86}$$

El parámetro θ, es efectivamente un promedio ponderado de aproximaciones de los puntos iniciales y finales del intervalo de tiempo. La figura (4.7) esquematiza el procedimiento anterior.

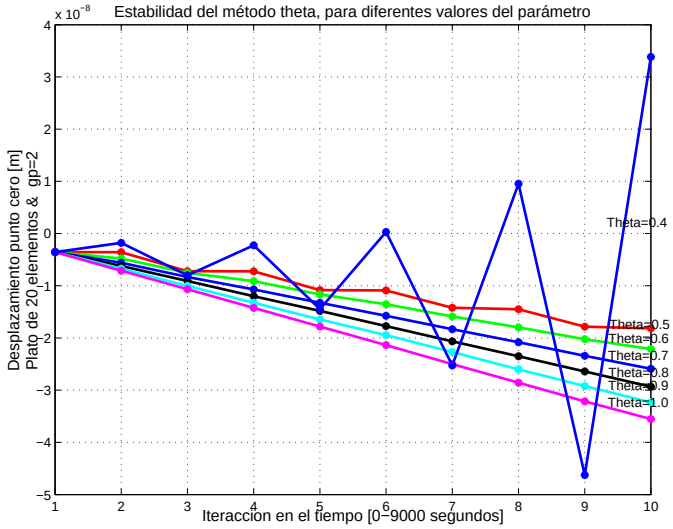


Figura 4.6: Estabilidad del método theta

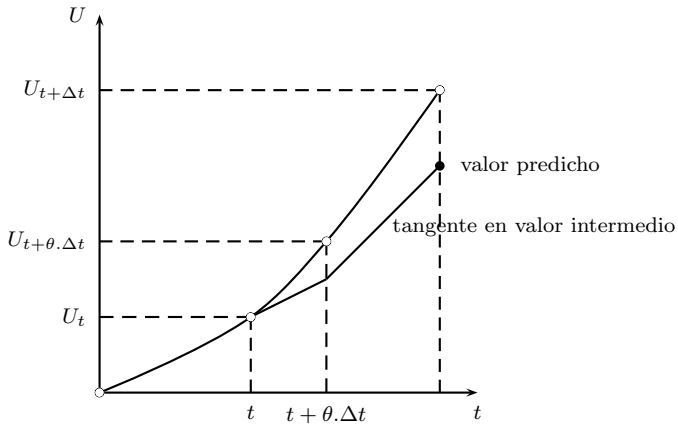


Figura 4.7: Método Theta

La desventaja de este método es que en cada nueva interación se requiere calcular la pendiente, es decir, la matriz de rigidez del sistema lo cual consume mucho tiempo y recursos de computador.

4.3.4 Cómputo del Vector de Carga Asociado al Desplazamiento Creep

En la marcha de la simulación del comportamiento mecánico de la pieza, se hace necesario calcular un vector de carga $\{q_c\}$ asociado con el desplazamiento "creep" de la pieza. Este vector de carga $\{q_c\}$ representa las fuerzas internas necesarias para resistir la deformación

"creep" que deben estar en equilibrio con las cargas generadas por el comportamiento elástico de la estructura.
Análogamente al desarrollo matemático realizado para el caso elástico, se puede encontrar que este vector es de la forma:

$$\{q_{ci}\} = \iint_{\Omega} ([L]\{\phi_i^o\})^T [D]\{\in_c\} r\, dr\, dz \tag{4.87}$$

donde i es el indice del elemento finito. Llevando esta ecuación a un elemento finito específico se tiene:

$$\{q_{ci}\} = \iint_{\Omega_E} ([L]\{\phi_i^o\})^T [D]\{\in_c\} r\, dr\, dz \tag{4.88}$$

la cual involucra el cálculo de los elementos estándar como:

$$\{q_{ci}\}_e = \iint_{\Omega_{st}} ([L]\{N_I\})^T [D]\{\in_c\} r^*\, d\xi\, d\eta \tag{4.89}$$

donde $I = 1, 2 \ldots, 2n^e$ y r^*: radio en términos de la transformación. Si se aplica la cuadratura de Gauss-Legendre a la ecuación anterior se tiene:

$$\{q_{I_c}\}^e \cong \sum_{r=1}^{NG} \sum_{s=1}^{NG} w_r w_s \left([L][N_I]\right)^T D\{\in_c\} r^* |J| \tag{4.90}$$

Usando la matriz cinemática (4.49) para efectos de cómputo, se tiene:

$$\{q_{I_c}\}^e \cong \sum_{r=1}^{NG} \sum_{s=1}^{NG} w_r w_s [mtxcin]^T [D]\{\in_c\} r^* |J| \tag{4.91}$$

4.4 Proceso de Solución

La solución de los problemas creep, normalmente adopta un acercamiento similar en la mayoría de los casos. Después de aplicar las restricciones en la frontera, se obtiene una solución elástica. Basados en estos esfuerzos inciales, se calculan los incrementos en deformación por fluencia ocurridos en un intervalo Δt, mediante la aplicación de una ley multi-axial, en cada punto de integración de Gauss. Los deplazamientos nodales resultantes de las deformaciones de fluencia estimadas, se determinan a partir de la matriz de rigidez elástica de la estructura, y se calculan los nuevos esfuerzos efectivos para iniciar un nuevo ciclo de iteracción.
En los análisis de fluencia, el incremento total de la deformación en cualquier tiempo se compone de dos factores, uno elástico $\{\epsilon^u\}$ y otro creep $\{\epsilon^c\}$. Por tanto la deformación total $\{\epsilon^{total}\}$ se calcula como:

$$\{\Delta\epsilon^{total}\} = \{\Delta\epsilon^u\} + \{\Delta\epsilon^c\}$$

El algoritmo presentado en el diagrama de flujo de la figura 4.8 adopta el siguiente procedimiento:

i) Aplicar las restricciones en la frontera y solucionar el sistema de ecuaciones para determinar los desplazamientos nodales y los esfuerzos elásticos en $t = 0$.

ii) Para un incremento de tiempo Δt se asume que los esfuerzos permanecen constantes y se determinan la velocidad de deformación creep y la deformación de fluencia, usando la ley multi-axial para el material.

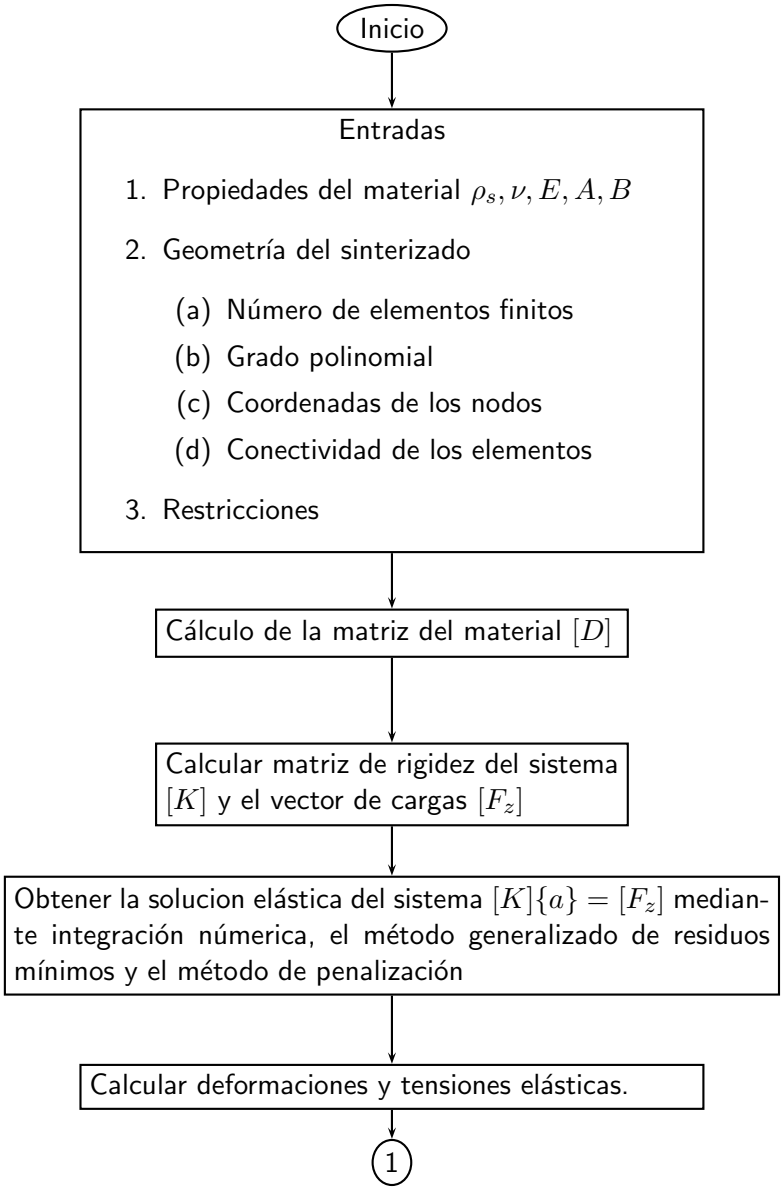
iii) A partir del incremento en la deformación de fluencia se determina un vector de cargas creep y se solucionan las ecuaciones para determinar un incremento en el desplazamiento $\{\Delta u\}$, con base en el método theta. Físicamente las cargas creep representan las fuerzas internas necesarias para resistir la deformación creep y mantenerse en equilibrio con las cargas generadas por el comportamient elástico de la estructura. En esta cálculo se debe garantizar que la variación en tensión $\Delta\sigma$ sea pequeña para asegurar precisión, pues la estabilidad la da la escogencia de theta.

iv) Actualizar coordenadas y recalcular la matriz de rigidez del sistema.

v) Calcular una nueva tensión e incrementar en un Δt el tiempo de marcha de la simulación. Repetir el proceso hasta alcanzar el tiempo final.

La precisión de la solución puede ser monitoreada mediante controles que limitan un cambio excesivo en la tensión efectiva de una iteración a otra. Adicionalmente se establecen controles con base en el residual de la solución y el número de iteraciones máximas permitidas en cada ciclo.

Los métodos numéricos empleados en la simulación fueron: Integración numérica (cuadratura de Gauss-Legendre), método de penalización, método generalizado de residuos mínimos, método theta e interpolación de Legendre.


Inicio
Entradas
1. Propiedades del material ρ_s, ν, E, A, B
2. Geometría del sinterizado
(a) Número de elementos finitos
(b) Grado polinomial
(c) Coordenadas de los nodos
(d) Conectividad de los elementos
3. Restricciones
Cálculo de la matriz del material $[D]$
Calcular matriz de rigidez del sistema $[K]$ y el vector de cargas $[F_z]$
Obtener la solucion elástica del sistema $[K]\{a\} = [F_z]$ mediante integración númerica, el método generalizado de residuos mínimos y el método de penalización
Calcular deformaciones y tensiones elásticas.
1

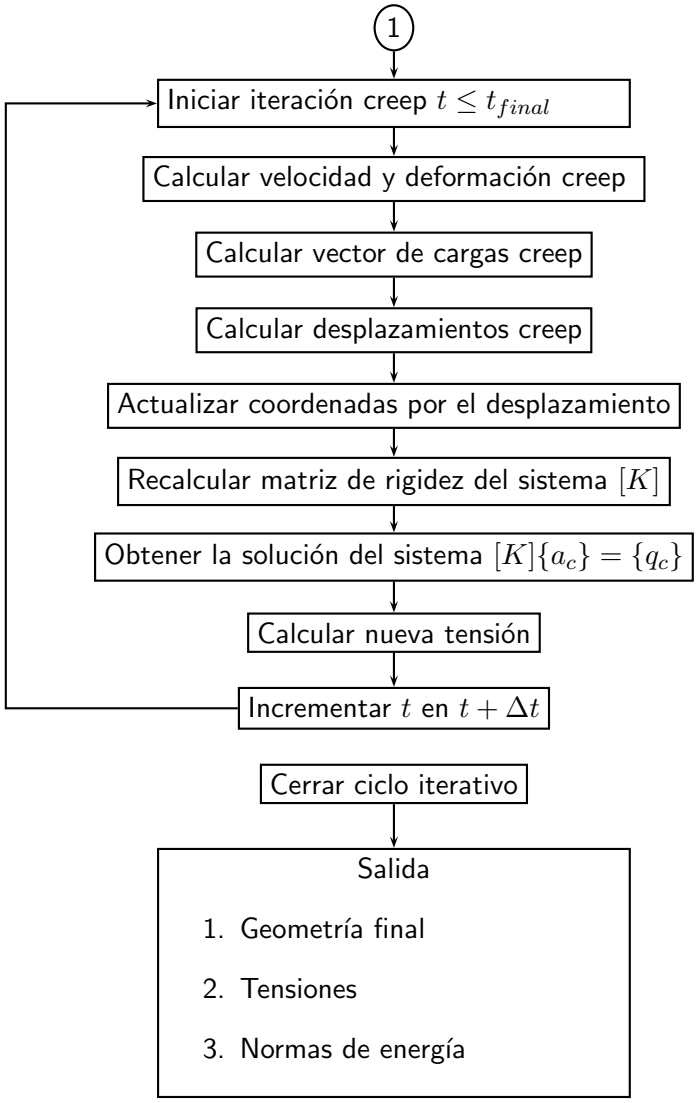


Figura 4.8: Flujograma para simular el comportamiento mecánico

4.5 Propiedades del Material y Constantes Usadas en la Modelación

Estimación de las Constantes del Modelo "Creep"

Para estimar las constantes del modelo de Norton modificado ecuación (4.73), se diseñó la siguiente prueba experimental uni-axial: Se fabricaron barritas cerámicas rectangulares, del mismo material de la pieza como se aprecia en la figura (4.9) de sección tranvesal $a \times b$ y longitud de prueba l_o.

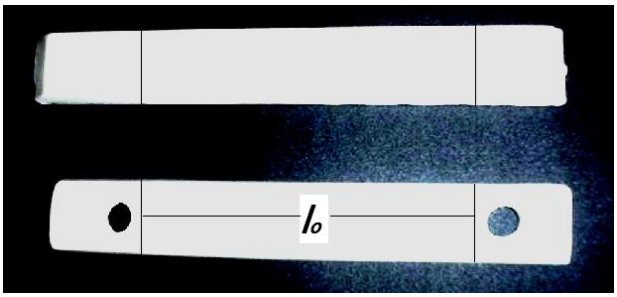


Figura 4.9: Fotografía de las barras cerámicas en la prueba uni-axial, de sección transversal axb

A cada extremo de la barrita en crudo, se les perforó un agujero, el superior sirve para montarlo en una estructura refractaria, que soporta la barrita en el proceso de cocción, y en el extremo inferior se le adapta un recipiente, que nos sirve para cargar la barrita con un peso determinado como se aprecia en la figura (4.10).

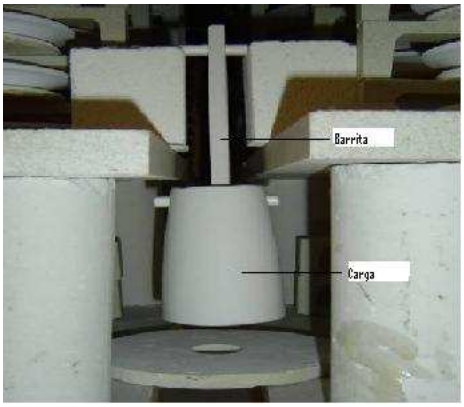


Figura 4.10: Montaje de la barrita en el horno

Para obtener la relación $\sigma - \epsilon$ (tensión deformación) se realizaron experimentos replicados con diferentes pesos, mediante la adición de Alúmina y zirconio al recipiente que soporta la barrita

en su parte inferior. Como durante la cocción se presentan los fenómenos de sinterización y deformación mecánica, cada experimento se acompaña de una barrita de igual longitud, la cual viaja a traves de horno en forma horizontal y cercana a la barrita vertical; de esta forma se sabe cuanta es la contracción del espécimen ocacionado por la sinterización y cuanta es la deformación mecánica de la barrita. Las mediciones de la barrita se realizaron con un calibrador de precisión de 0.05 mm.
Para el cálculo de las constante de la ley uni-axial (4.73) se consideraron las siguientes variables: L_o longitud inicial en crudo de las barritas, L_{f_1} Longitud final de la barrita quemada horizontalmente y L_{f_2} Longitud final de la barrita quemada verticalmente. Entonces por definición:

$$\begin{aligned} \in &\stackrel{def}{=} \frac{L_{f_2} - L_{f_1}}{L_{f_1}} \text{ : deformación unitaria mecánica} \\ \sigma &\stackrel{def}{=} \frac{\text{Peso}}{\text{area transversal}} \\ &= \frac{W}{a \times b} \quad \text{tensión uniaxial} \end{aligned}$$

El tiempo de permanencia de las barritas a la máxima temperatura es de 2.5 horas, los datos obtenidos se aprecian en la siguiente tabla:

Cuadro 4.1: Datos para el cálculo de las constantes Creep

No	L_{f_1}	L_{f_2}	**w**	**b**	**Contracción**	**Deformación unitaria** ϵ	**Tensión**
	$[mm]$	$[mm]$	$[g]$	$[mm]$	$[\%]$	$[mm]$	$[N/m^2]$
1	87,20	87,60	134	23,43	12,4	0,004587	631,4609
2	87,50	88,30	200	23,39	11,7	0,009143	946,8874
3	87,80	88,30	200	23,39	11,7	0,005695	946,8874
4	87,80	88,75	300	23,48	11,3	0,010820	1415,0370
5	87,74	88,70	300	23,48	11,3	0,010941	1415,0370
6	87,00	88,10	400	23,33	11,9	0,012644	1898,8492
7	87,00	88.90	400	23,33	11,1	0,021839	1898,8492
8	87,20	88,10	500	23,28	11,9	0,010321	2378,6604
9	87,75	89,60	500	23,28	10,4	0,021083	2378,6604
10	87,10	89,60	600	23,58	10,4	0,028703	2818,0694
11	87,00	88,90	600	23,58	11,1	0,021839	2818,0694
12	87,50	89,70	700	23,27	10,3	0,025143	3331,9140
13	87,25	90,45	700	23,27	9,6	0,036676	3331,9140
14	87,50	90,90	800	23,20	9,1	0,038857	3818,1601
15	87,45	91,80	800	23,20	8,2	0,049743	3818,1601
16	87,35	92,10	900	22,34	7,9	0,054379	4459,7887
17	87,15	90,95	900	22,34	9,1	0,043603	4459,7887
18	87,60	91,95	1000	23,28	8,1	0,049658	4757,3209
19	87,20	92,55	1000	23,28	7,5	0,061353	4757,3209
Fin de la tabla							

Nota: Longitud inicial $l_o = 100\ [mm]$, Espesor $a = 9,0313\ [mm]$ y Tiempo $t = 9000\ [s]$
Realizando un ajuste polinomial grado 2, el cual es adecuado según los criterios de bondad de ajuste (La prueba de bondad de ajuste el valor $p = 0,9900$ que debe ser mayor a $0,10$ hace suponer que es adecuado), ver figura (4.11), se obtienen las constantes del modelo (4.73):

- $At = 1.62528 \times 10^{-11}$
- $Bt = -4.00782 \times 10^{-7}$
- $R^2 = 93.2883\,\%$
- $R^2(\text{ajustado}) = 92.2883\,\%$

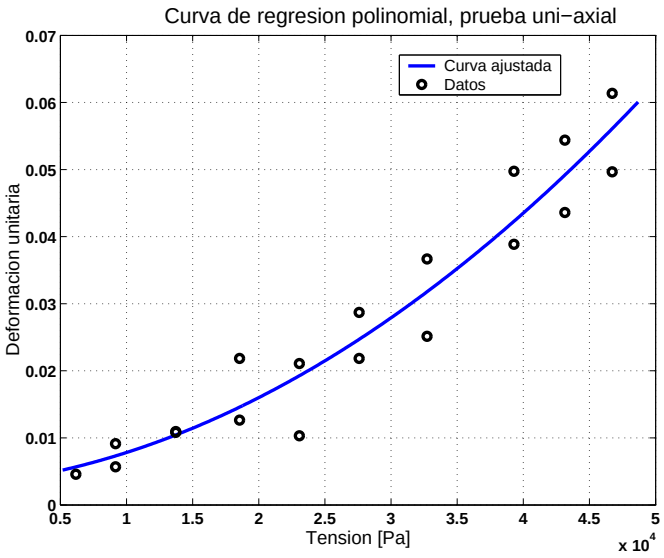


Figura 4.11: Correlación entre la tensión y la deformación unitaria

Un dato interesante en la industria cerámica es la contracción total de la pieza que se incluye en la tabla (4.1) y que se calcula como $\%C = \frac{l_f - l_o}{l_o}$. También se obtuvo la curva de contracción-tensión (4.12), útil para análisis cerámicos de carga de platos en pilas, que permiten predecir la variación dimensional con la carga aplicada. Adicionalmente permite validar el modelo de sinterización empleado en la simulación.

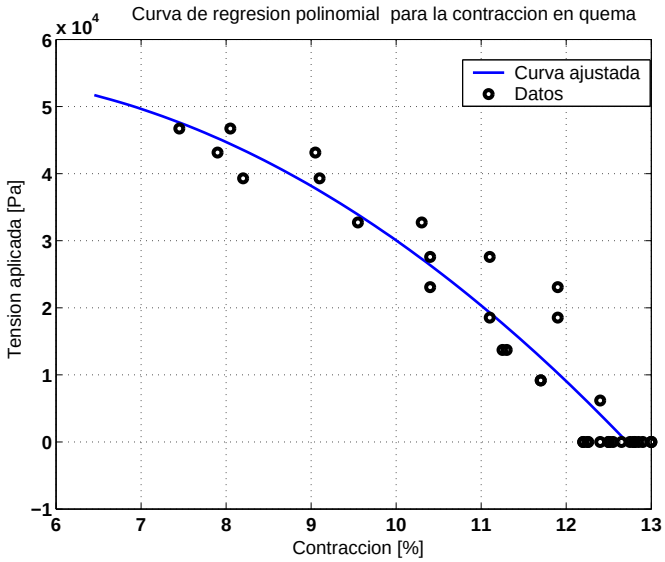


Figura 4.12: Contraccion de la pieza con la carga aplicada

Cuadro 4.2: Propiedades del material y modelo

Propiedades o constantes	Valor	Fuente
Densidad en crudo	$2100[kg/m^3]$	Mediciones realizadas en laboratorio cerámico de la empresa
Densidad del sinterizado	$2400[kg/m^3]$	Mediciones realizadas en laboratorio cerámico de la empresa
Pérdidas por ignición	9.2 %	Mediciones realizadas en laboratorio cerámico de la empresa
Módulo de Young	$8 * 10^9[Pa]$	Extractado de bibliografía, ver [31]
Coeficiente de Poisson	0.4949	Extractado de bibliografía, ver [15]. En los procesos creep el volumen permanece constante
Constantes del modelo creep	$A = 1.80587 * 10^{-15}$ $B = 4.45313 * 10^{-11}$	Esimado experimentalmente en proceso industrial
Tiempo de sinterización	9000[s]	Curva de cocción del horno

CAPÍTULO 5

Programación de la Solución

Para llevar a cabo la simulación numérica, se programaron los algoritmos de solución en códigos de Matlab (versión 6.1.5). La captura de la geometría de la pieza cruda se realizó mediante el escaneo de un corte transvesal de la pieza, cargado directamente al programa Autocad, donde se procesó la imagen para obtener un perfil digitalizado, que se pudiera alimentar a un generador de mallas. El enmallado, se realizó en el programa Cosmos/M (versión 2.5), donde se obtuvieron las diferentes mallas necesarias para realizar las simulaciones numéricas. Este programa genera mallas admisibles según la definición (3.1) y controla en número de elementos mediante el tamaño medio del elemento. La generación de mallas es en sí misma un campo muy amplio de la investigación matémática, algunos aspectos de este tópico se pueden profundizar en las referencias [27], [25] y [33].

En cuanto al Hardware utilizado, la programación se implementó en un computador con procesador Pentium III, velocidad de procesamiento de 1.8 MHz y un giga de memoria RAM. Una vez iniciadas las simulaciones, se encontró que la combinación Software- Hardware seleccionado no soportó mallas superiores a 310 elementos y grados polinomiales mayores a seis, aspecto que limitó el alcance de los resultados relacionados con las extensiones-hp; a pesar de haber optimizado la programación con uso de variables globales-locales, manipulación de matrices sparse y objetos simbólicos entre otros. Cabe anotar que el uso del método theta en la busqueda de la solución creep requiere calcular en cada nueva iteración la matriz de rigidez del sistema con tolerancias muy pequeñas para asegurar la precisión de los resultados lo cual consume mucho tiempo y memoria de máquina. Los tiempos de simulación variaron entre 1200 y 345000 segundos (20 minutos y 4 días).

En las siguientes secciones se explica la estructura de la programación, la descripción de los programas y los flujogramas de los principales algoritmos desarrollados para ejecutar la simulación.

5.1 Estructura y Descripción de los Programas

La programción de la solución se articuló alrededor de un programa principal llamado **Modela-Plato.m**. Este programa llama las diferentes subrutinas y funciones necesarias para completar la simulación; su estructura básica se esquematiza en la figura (5.1).

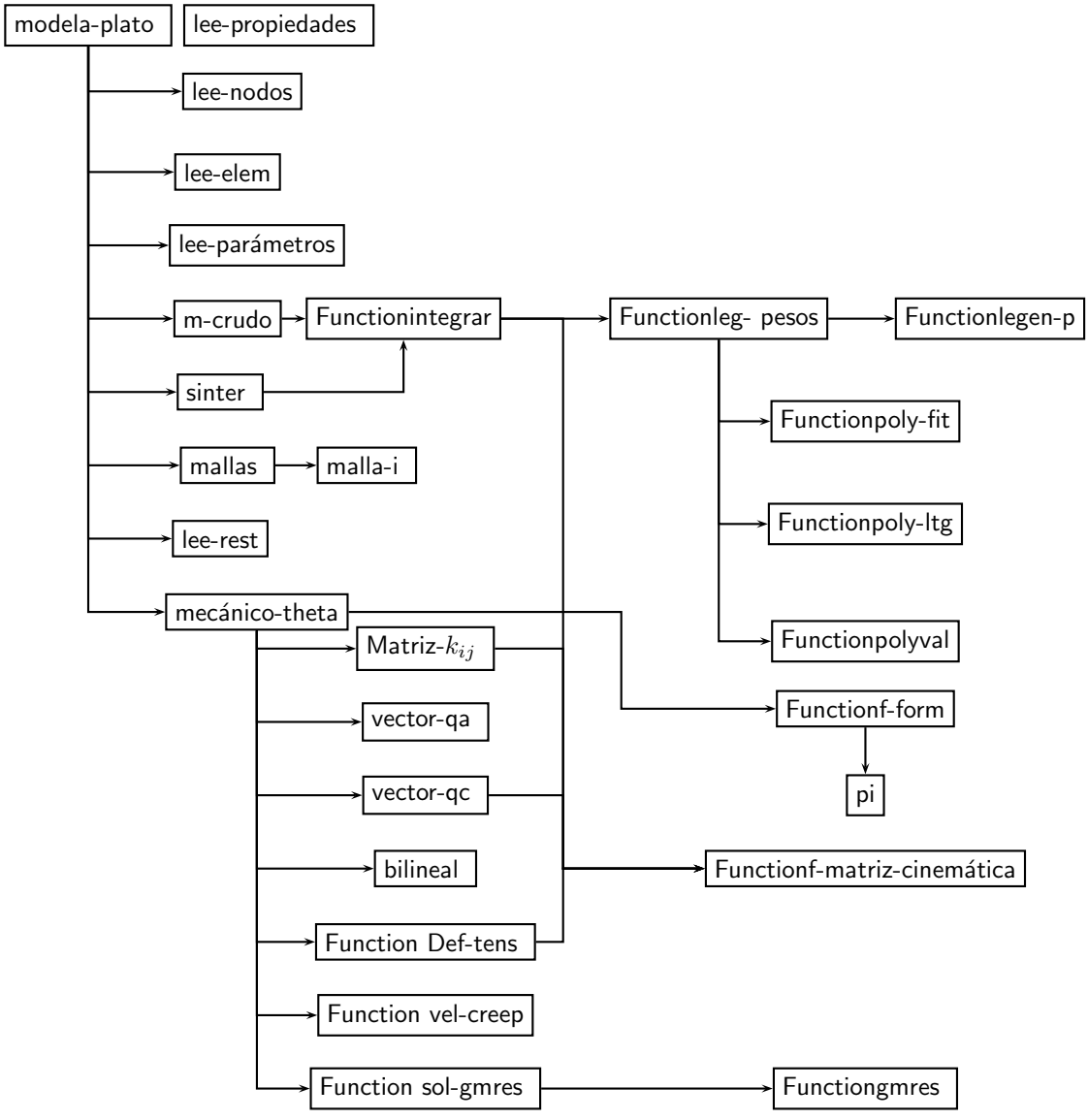


Figura 5.1: Estructura jerárquica de programas

El programa inicia con la lectura de los datos de entrada necesarios en toda la simulación como

son:

- Propiedades del material cerámico empleado en la fabricación del plato.
- Las coordenadas de los nodos de los elementos, generados como un archivo de salida del programa Cosmos/M.
- La conectividad de los elementos igualmente entregados por Cosmos/M.
- Finalmente, los parámetros de los métodos numéricos empleados tales como: tolerancias, número de iteraciones, grado polinomial y puntos de Gauss.

Estos cuatro tipos de datos entran en formato de archivo de texto. En el apendice (F) se incluye una copia de cada uno de estos archivos.

Posteriormente el programa calcula los elementos másicos y simula el proceso de sinterización, cuyos resultados son necesrios para iniciar la simulación del comportamiento mecánico, y que a su vez son una salida del programa principal.

Una vez se tienen las coordenadas de los nodos sinterizados, se procede a leer las restricciones de la frontera, a partir de un archivo de texto donde se especifican los nodos restringidos. A partir de esta etapa, inicia la simulación del comportamiento mecánico elástico y de fluencia del plato cerámico para obtener finalmente la información de las coordenadas desplazadas y de más variables de interés las cuales quedan consignadas en una base de datos de Matlab. Esta base se puede utilizar unz vez terminada la simulación para realizar análisis de interés por parte del investigador.

A continuación se detallan el propósito y parámetros principales de cada subprograma y en el apendice (E) se encuentran los códigos completos en el lenguaje Matlab.

Cuadro 5.1: Estructura de los programas en Matlab

Programa o función	Propósito	Parámetros
Modela-plato	Integrar las diferentes subrutinas para modelar la sinterización y comportamiento mecánico del plato	Se accesa digitando: modela-plato directamente en la pantalla
Lee propiedades	Leer desde un archivo de texto (Note pad) las propiedades fisicas del material, módulo de Young, coeficiente de Poisson, densidades,pérdidas por ignición y constantes del modelo mecánico	-
Lee-nodos.m	Lee desde un archivo de texto el número de nodos y las coordenadas de los nodos	-
Lee-elem.m	Lee la conectividad de los elementos desde un archivo de texto. Además reordena los elementos a partir del nodo $(0, 0)$ que se corresponde con el centro inferior del plato.	-
Lee.parámetros.m	Lee los parámetros matemáticos a ser usados. Número máximo de iteraciones para la convergencia de la integración numérica (Gauss.Legendre), tolerancias.	-
Sigue . . .		

Cuadro 5.1: Estructura de los programas en Matlab(continuación)

Programa o función	Propósito	Parámetros
Mecánico-theta.m	Calcular el comportamieto mecánico, elástico y "creep" del plato. Para el modelo elástico usa la formulación del PTV y el método de los elementos finitos y para el comportamiento creep usa el método theta. Entrega las coordenadas sinterizadas, las tensiones y deformaciones, entre otras variables y las graficas solicitadas.	-
Función integrar	Calcula la integral numérica solicitada en el programa donde se llama la función. Requiere las coordenadas del elemento, la tolerancia para la convergencia y el número máximo de iteraciones para evitar una estagnación del programa.	-
Función leg-pesos	Calcular los puntos y pesos necesarios para realizar la integración numérica, se hace con base en los polinomios de Legendre.	La tolerancia para la convergencia de la integración númerica, el primer beta supuesto para iniciar la iteración de Newton-Rapson en la sinterización; el delta-beta para calcular de la derivada en el método de Newton-Rapson; la tolerancia para estimar la densidad; el n+umero de grados de libertad; el número de grado polinomial a ser usado en la solución mecánica (extensión-p),el número de puntos de Gauss a usar en la integral, el tiempo final de sinterización e incremento de tiempo en la iteración creep y el número de nodos restringidos.
m-crudo.m	Calcula los elementos másicos mediante integración numérica de Gauss-Legendre.	
Sinter.m	Calcula las coordenadas sinterizadas.	Beta supuesto para iteración N-R
Función f-form	Direcciona el cálculo de las funciones de forma según el grado polinomial a usar el modelamiento mecánico, llama a los subprogramas pi que calcula las funciones de forma y sus derivadas parciales.	
lee-rest.m	Lee los grados de libertad restringidos	Grado polinomial
Funciones: legen-p, poly-fit, poly-itg, poly-val	Construir los polinomios de Legendre. Las tres últimas funciones las trae Matlab por defecto.	Número de puntos de Gauss.
mallas.m	Direcciona el cálculo de los nodos laterales e internos según el grado polinomial a usar en el modelamiento mecánico. Llama a los subprogramas malla.i que calcula las coordenadas de estos nodos en forma lineal y establece las conectividades de todos los nodos.	• Número de elementos • Matriz de grados de libertad • Grados de libertad total del sistema • Coordenadas nodales • Nodos por elemento.

Sigue . . .

Cuadro 5.1: Estructura de los programas en Matlab(continuación)

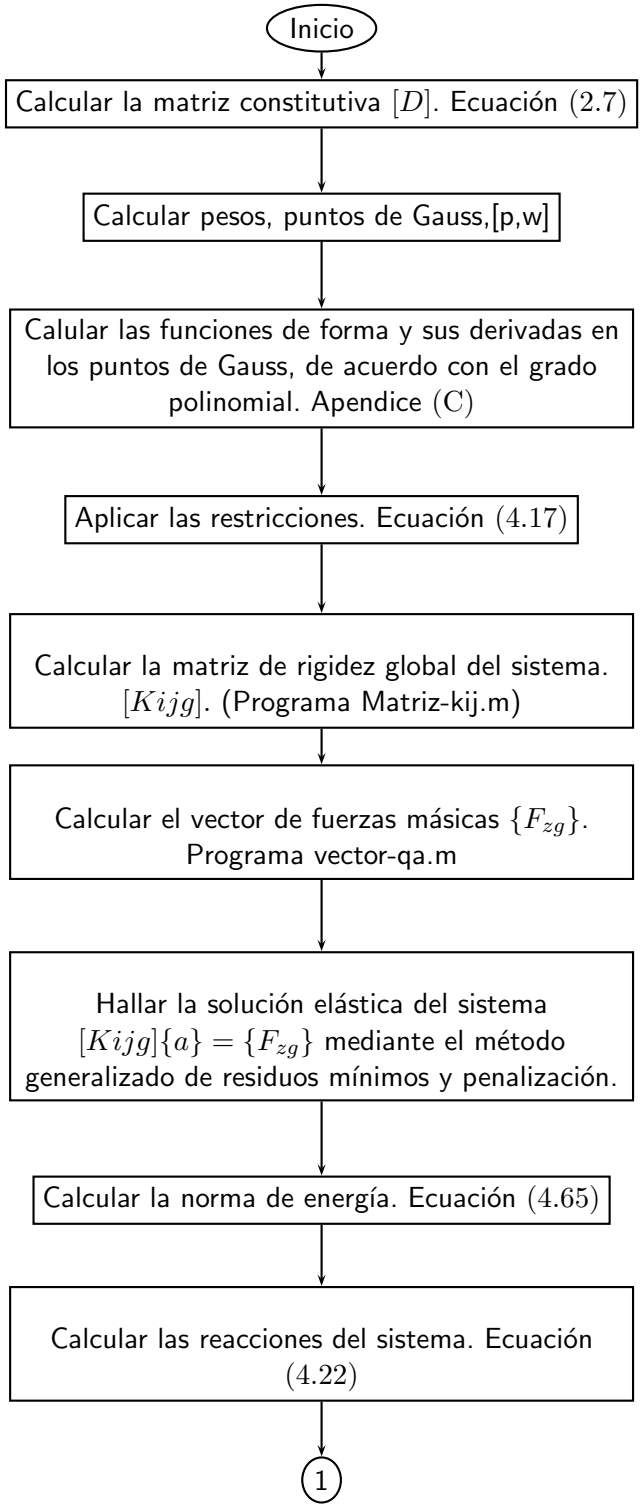
Programa o función	Propósito	Parámetros
Matriz-kij.m	Calcula las matrices de rigidez de cada elemento y ensambla la matriz de rigidez global del sistema	• Coordenadas de los nodos • Número de elementos • Número de nodos • Puntos de Gauss
f-matriz-cinemática	Ensambla las funciones de forma y sus derivadas evaluadas en los puntos de Gauss, en una sola matriz para optimizar los cálculos.	• Número de funciones de forma • Distancia radial en coordenadas naturales • Puntos de Gauss
Vector-qa.m	Calcula el vector de fuerzas másicas de cada elemento y ensambla el vector global.	• Densidad sinterizado • Número de elementos • Coordenadas de los nodos.
Vector qc.m	Calcular el vector de cargas asociado al comportamieto creep y ensamblar un vector global	• Número de elementos • Coordenadas de los nodos • Puntos de Gauss • Matriz constitutiva del material • Distancia radial en coordenadas naturales.
Bilineal.m	Calcular la función $\mathscr{B}(u,u)$ necesaria para hallar la norma de energía.	• Número de elementos • Conectividad de elementos • desplazamientos • Número de puntos de Gauss.
Sigue ...		

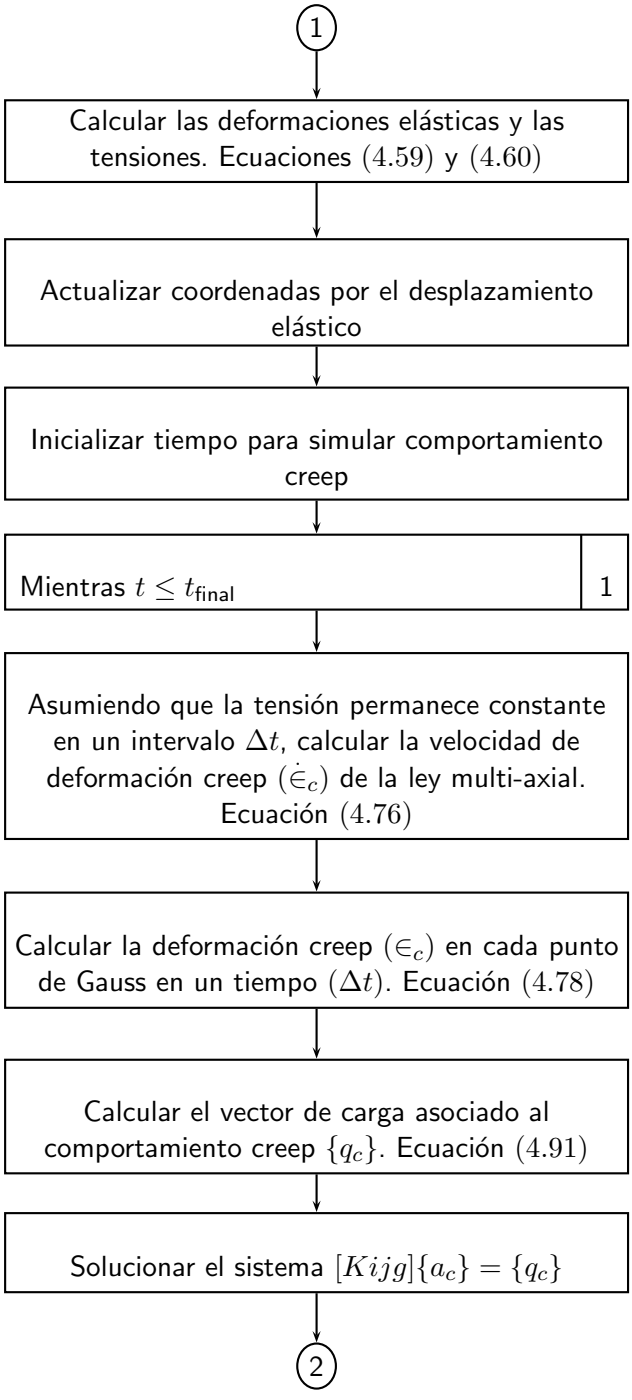
Cuadro 5.1: Estructura de los programas en Matlab(continuación)

Programa o función	Propósito	Parámetros
Función Def-tens	Calcula las deformaciones y tensiones elásticas	• Grado polinomial • Número de elementos • Puntos de Gauss • Grado polinomial
Función vel-creep	Calcula las velocidades de deformación creep en los puntos de Gauss a partir de la ley multi-axial	• Número de elementos • Constantes del modelo de Norton-Bayley • Puntos de Gauss
Función sol-gmres	Aplica las restricciones a la matriz de rigidez del sistema, y aplica el método de penalización para la solución iterada del sistema $[K]\{a\} = \{F_z\}$	• Coeficiente de penalización $= 1 * 10^{-4}$
Función gmres	Resolver mediante el método generalizado de los residuos mínimos el sistema $[K]\{a\} = \{F_z\}$. Esta función la trae Matlab en su librería y se adaptó al problema concreto de estudio	• Re-inicio de la iteración = glt • Tolerancia de convergencia $= 1 * 10^{-6}$ • Número máximo de iteraciones = glt
Fin de la tabla		

5.2 Principales Diagramas de Flujo de los Algoritmos

Flujograma para el Algoritmo de la Modelación Mecánica (mecanico-theta.m)




1
Calcular las deformaciones elásticas y las tensiones. Ecuaciones (4.59) y (4.60)
Actualizar coordenadas por el desplazamiento elástico
Inicializar tiempo para simular comportamiento creep
Mientras $t \leq t_{\text{final}}$
1
Asumiendo que la tensión permanece constante en un intervalo Δt, calcular la velocidad de deformación creep $(\dot{\in}_c)$ de la ley multi-axial. Ecuación (4.76)
Calcular la deformación creep $(\in_c)$ en cada punto de Gauss en un tiempo (Δt). Ecuación (4.78)
Calcular el vector de carga asociado al comportamiento creep $\{q_c\}$. Ecuación (4.91)
Solucionar el sistema $[Kijg]\{a_c\} = \{q_c\}$
2

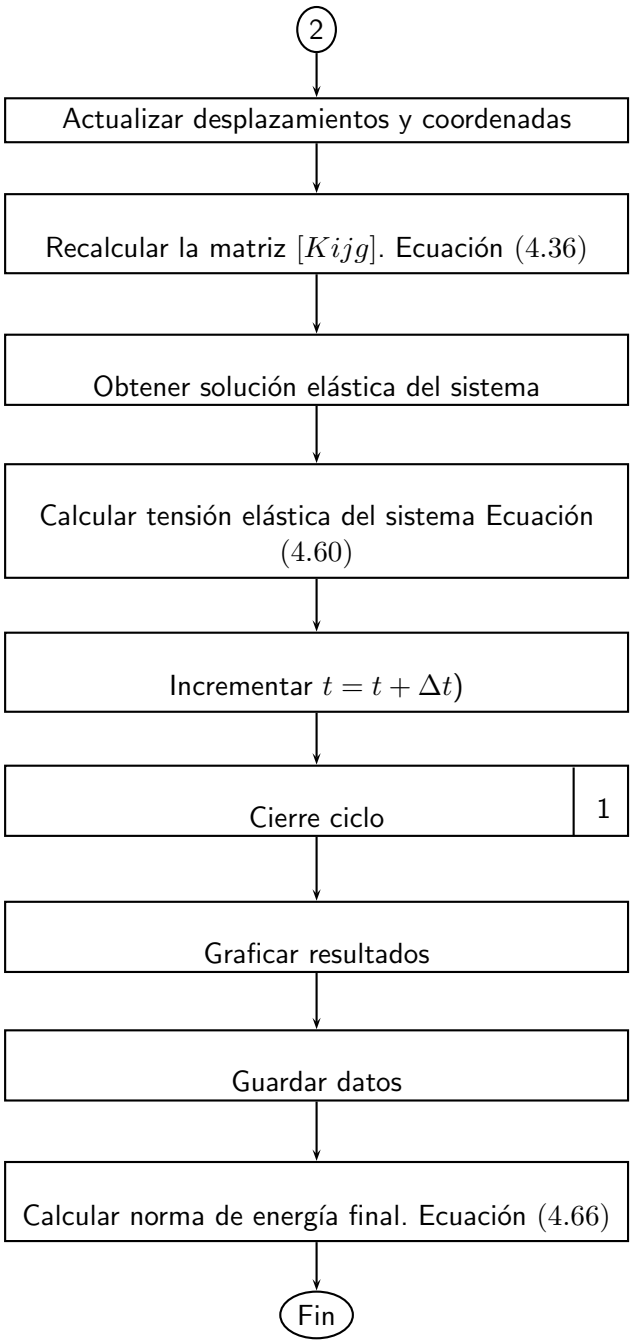
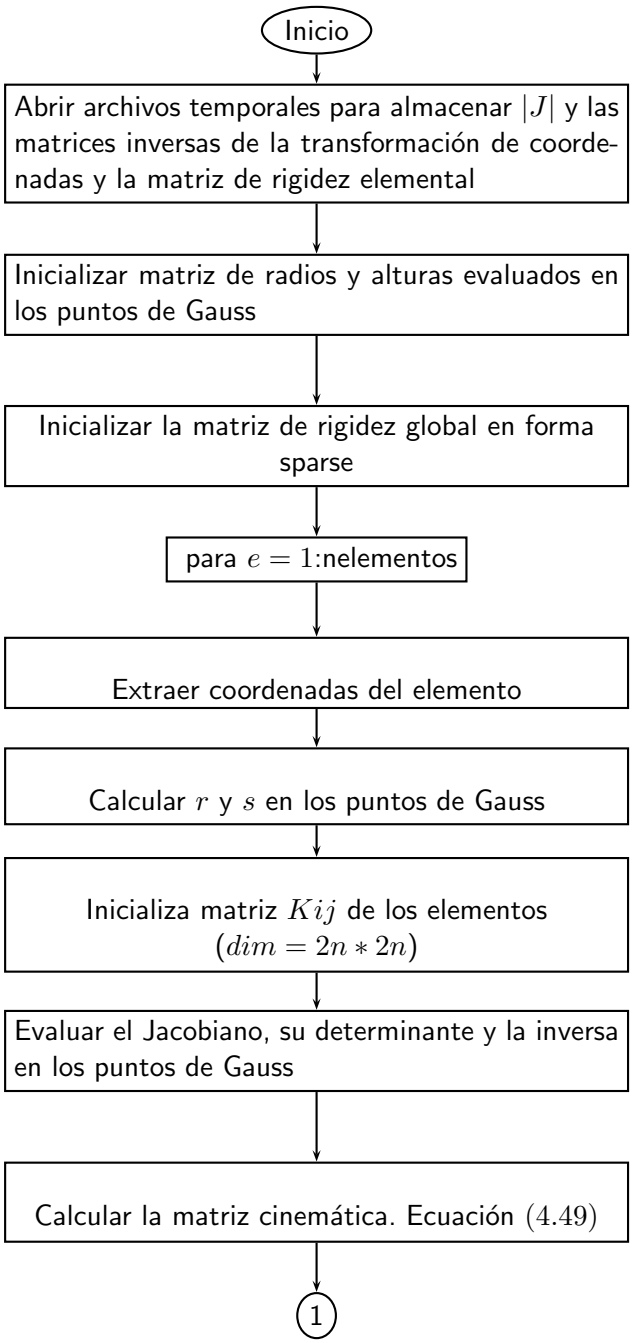


Figura 5.2: Flujograma para el algoritmo mecanico-theta.m

Flujograma para el Cálculo de la Matriz de Rigidez (matriz-kij.m)



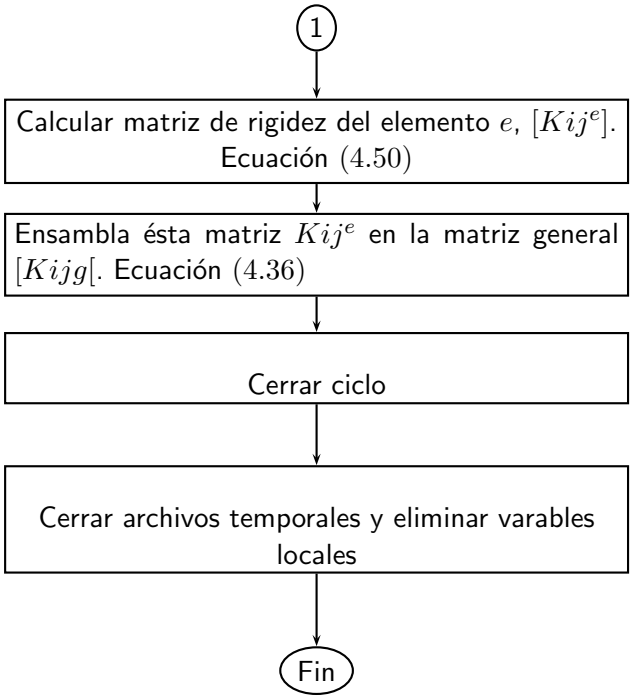


Figura 5.3: Flujograma para el algoritmo Matriz-kij.m

Flujograma para el Cálculo del Vector de Fuerzas (vector-qa.m)

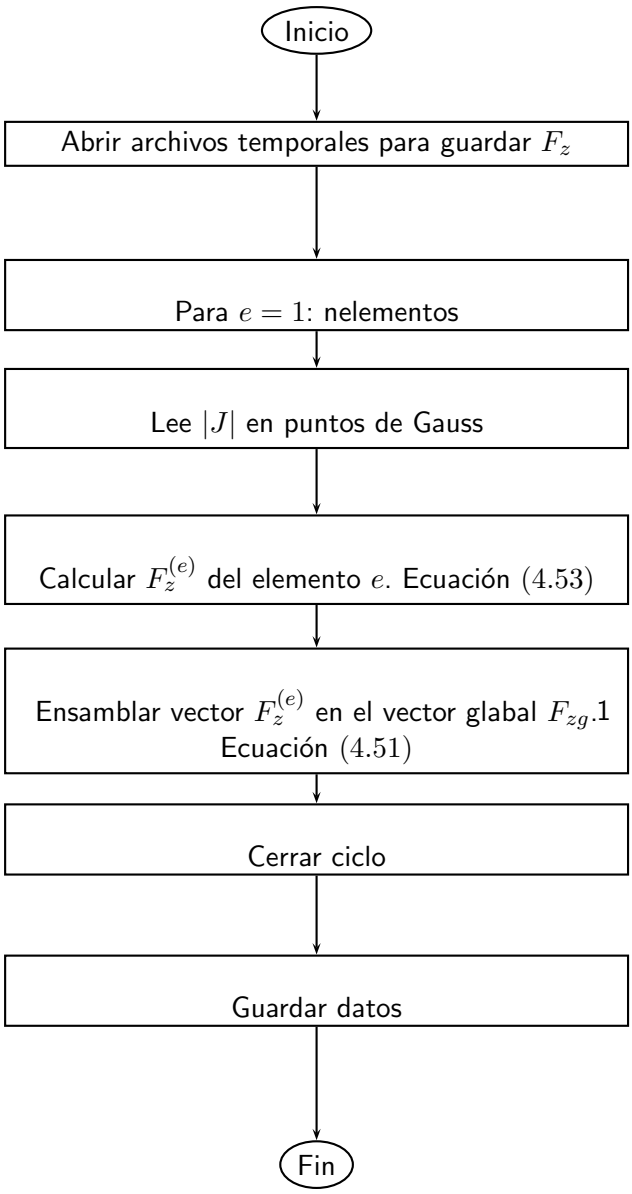


Figura 5.4: Flujograma para el algoritmo vector-qa

CAPÍTULO 6

Resultados y Análisis

Para validar las simulaciones realizadas se tomó un plato crudo de porcelana de alúmina con un diámetro $d = 31.00\ cm$ y una altura $h = 2.62\ cm$ medida desde la base del plato hasta el punto más alto en el ala como se aprecia en la figura (6.1)

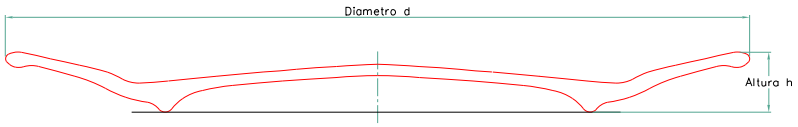


Figura 6.1: Acotamiento plato crudo

Este plato se sometió al proceso de cocción en iguales condiciones de temperatura y ciclo establecidas para la simulación. Una vez sale del proceso se miden nuevamente estas variables

arrojando un diámetro de $d = 27.00\ cm$ y una altura $h = 2.09\ cm$ producto de los procesos de sinterización expansión y fluencia (creep). Ambos platos se cortan y sus perfiles se digitalizan y transfieren al programa Autocad para digitalizar su perfil y compararlo con los resultados obtenidos de la simulación. En la figura (6.2) se esquematizan los dos platos.

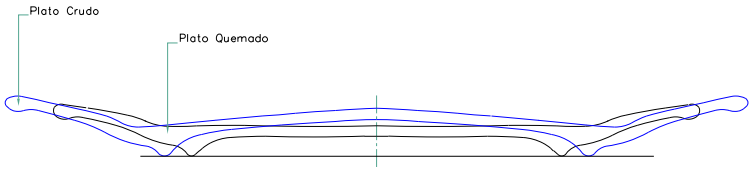


Figura 6.2: Esquema comparativo entre el plato crudo y quemado

6.1 Sinterización

De cada simulación realizada con cierta malla y grado polinomial es posible obtener el diámetro y altura final de cada plato. Para comparar con los datos reales se tomaron los resutados provenientes de la simulación con un plato de 310 elementos y grado polinomial 2, esta comparación se establece en la tabla (6.1).

El resultado obtenido en términos de diámetro presentan diferencias versus el real de -0.52 % equivalente a1.4 mm y en altura de 0.48 % equivalente a 0.1 mm; diferencias que están dentro de la tolerancia permitida para este producto en el mercado, que acepta variaciones dimensionales de $\pm$ 2 %. En la gráfica (6.3) se aprecia el semi-plano del plato simulado donde se compara con el plato real en crudo y sinterizado.

Cuadro 6.1: Resultados de la sinterización

Variable	Valor simulado	Valor experimental	Error relativo
Diámetro final $d[m]$	0.271392	0.2700	-0.52 %
Altura final $h[m]$	0.020800	0.02090	0.48 %
Factor β ecuación (4.14)	0.278740	-	-
Contracción %	12.45 %	12.90 %	3.49 %

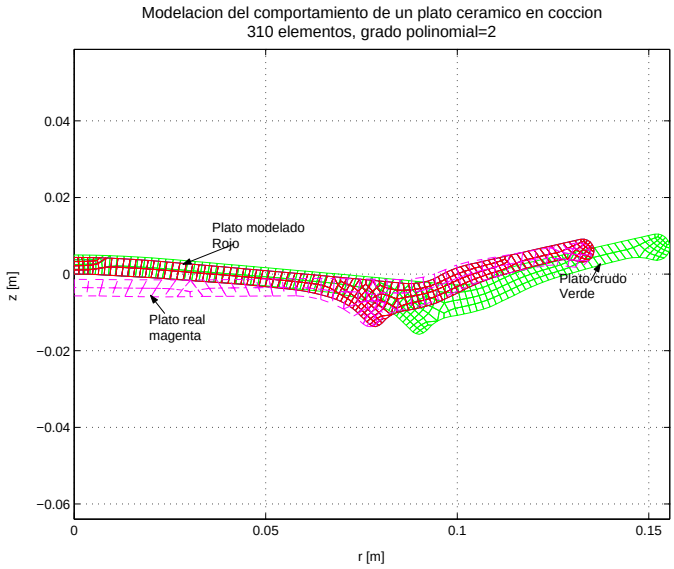


Figura 6.3: Plato real y modelado

6.2 Comportamiento Mecánico

En las figuras (6.4) y (6.5) se aprecian las distribuciones de la tensión y la velocidad de deformación "creep" con las diferentes coordenadas en un tiempo $t = 9000\ s$. El programa desarrollado permite obtener estas varibles y otras de interés mecánico para realizar análisis de esfuerzos, desplazamientos y velocidades, que son de utilidad al momento de diseñar piezas cerámicas e inferir posibles resultados en la producción industrial.

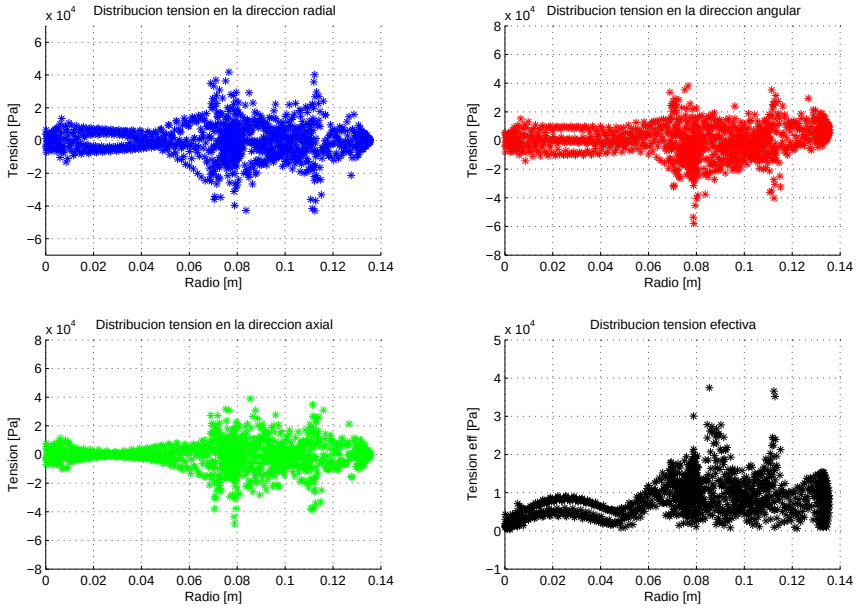


Figura 6.4: Distribución de la tensión efectiva en las diferentes direcciones

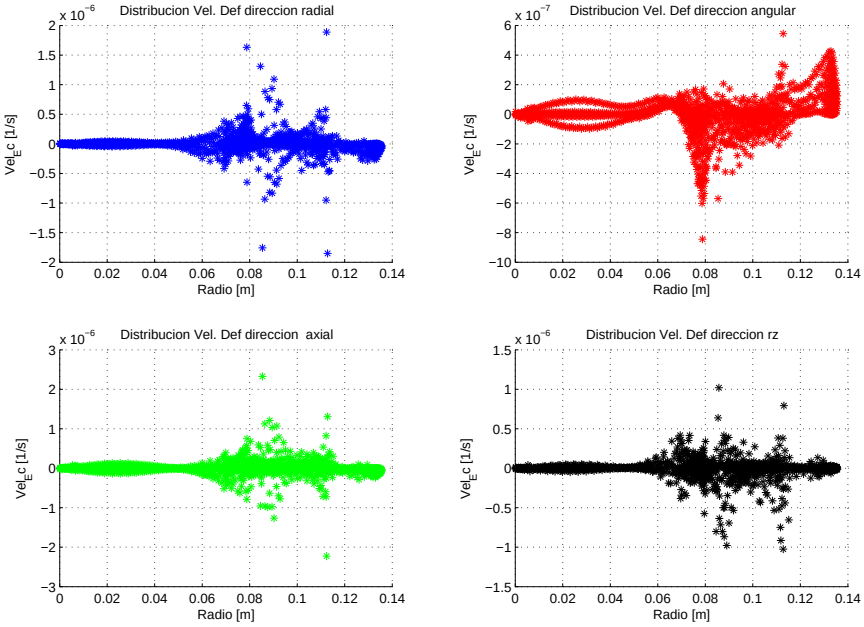


Figura 6.5: Distribución de la velocidad creep en las diferentes direcciones

En las figuras (6.6) y (6.7) se aprecian los comportamientos en el tiempo de las variables de desplazamiento $\{u\}$ para un punto determinado, la tensión efectiva σ_{eff} para el elemento 1 y la velocidad de deformación creep $\{\dot{\epsilon}_c\}$ del mismo elemento; para dos simulaciones, una con una malla de 91 elementos y grado polinomial 6 y otra con una malla de 310 elementos y grado polinomial 2. En ambos casos la velocidad de deformación es constante, que corresponde con la etapa secundaria del fenómeno creep (ver capítulo 1); también se puede apreciar que la tensión efectiva prácticamente no varía, resultado coherente desde el punto de vista físico, porque el movimiento de fluencia no debe generar esfuerzos adicionales a la estructura, esto es garantía de que el modelo implementado numéricamente, controla en forma adecuada la variación de tensión durante la marcha en el tiempo, el cual es un parámetro que es utilizado para mantener la precisión de los resultados.

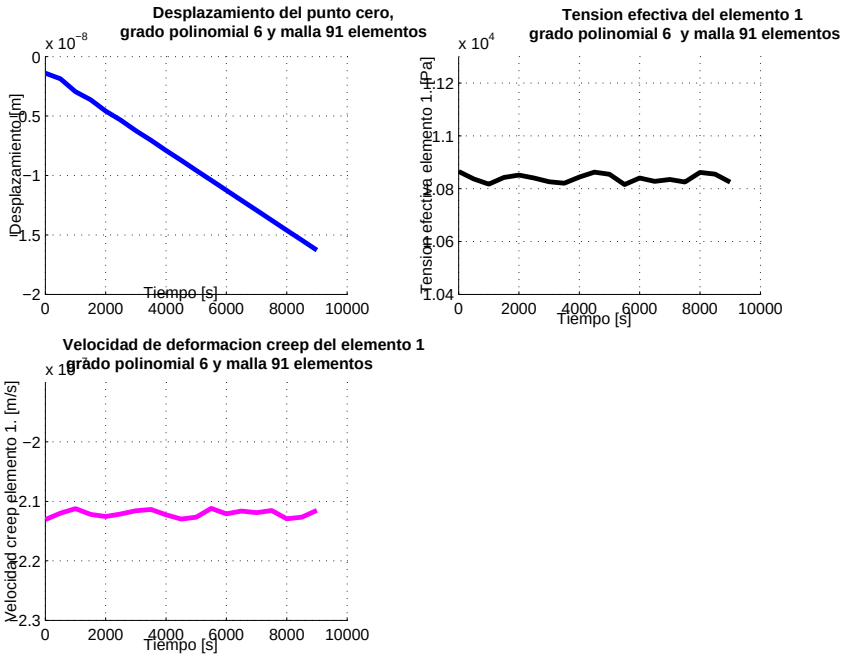


Figura 6.6: Comportamiento de variables grado polinomial seis

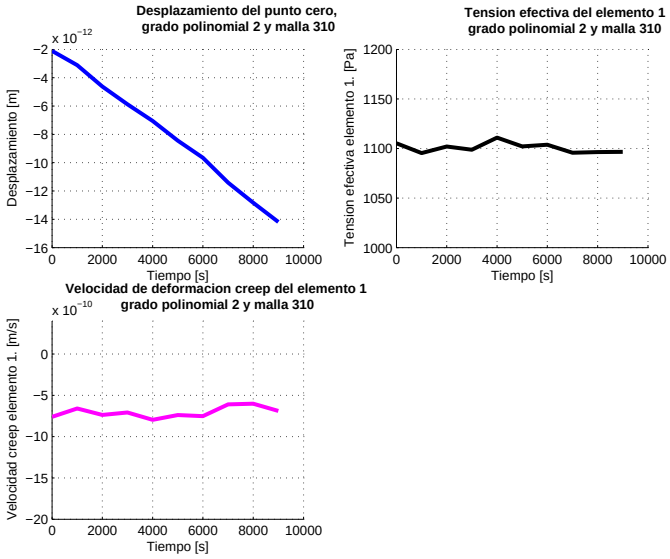


Figura 6.7: Comportamiento de variables con malla 310

6.2.1 Extensión-p

Se analizó la velocidad de convergencia con la extensión-p, medida como el número de iteraciones internas necesarias en cada Δt para alcanzar el residual de la norma de energía impuesto al algoritmo de 0.1 %, se observa en la figura (6.8) que a mayor grado polinomial requiere menor número de iteracciones, es decir, se mejora la precisión, pero se aumenta el tiempo de máquina. El tiempo de máquina osciló entre 20 mn y 8 h.

La implementación numérica para simular el comportamiento creep con la extensión-p, muestra una tendencia de disminución del error relativo (este se calculó con base en la solución de mayor grado polinomial, según el artículo de Restrepo [35]) al aumentar los grados de libertad (grado polinomial). Sin embargo, un mejor análisis de la convergencia requiere una simulación con grados mayores, lo cual a su vez exige un software y/o una máquina más potente, en la figura (6.9) se aprecia estos valores para los grados polinomiales 2, 3, 4 y 5.

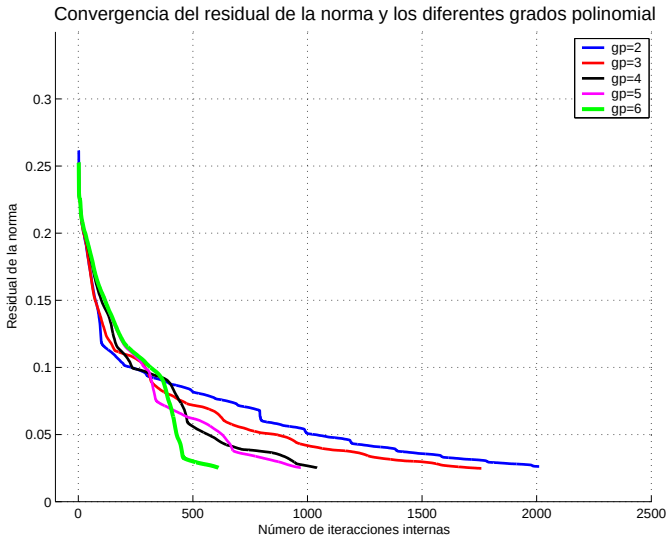


Figura 6.8: Convergencia del residual de la norma y de los diferentes grados polinomiales

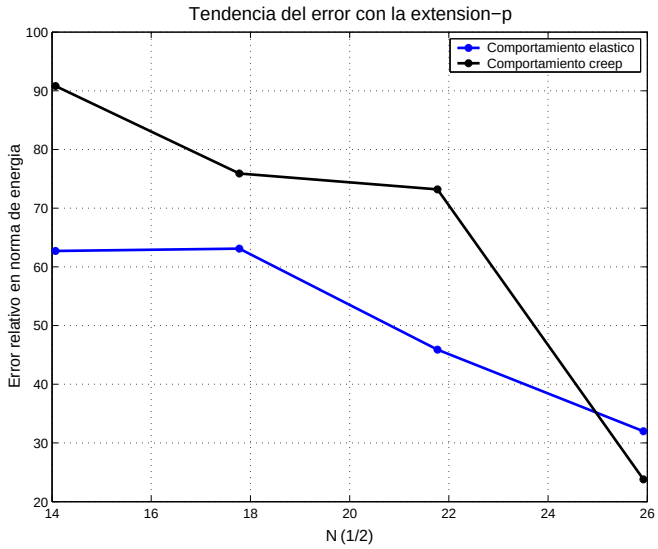


Figura 6.9: Convergencia en error relativo en norma de energía extensión-p

6.2.2 Extensión-h

Se analizó la velocidad de convergencia con la extensión-h, medida como el número de iteraciones internas necesarias en cada Δt para alcanzar el residual de la norma de energía impuesto al algoritmo de 0.1 %, se observa en la figura (6.10) que a mayor número de elementos se requiere menor número de iteraciones, es decir, se mejora la precisión, pero igualmente al caso de extensión-p se aumenta el tiempo de máquina, este osciló entre 20 mn y 4 días.

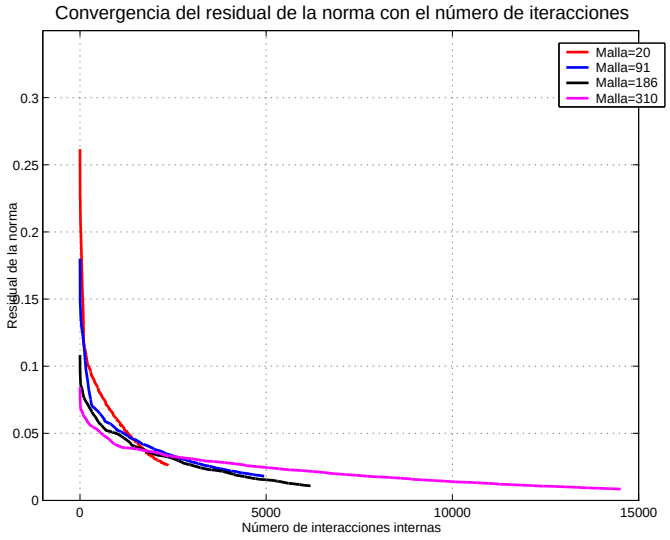


Figura 6.10: Convergencia del residual

La implementación numérica para simular el comportamiento creep con la extensión-h, al igual que la extensión-p, muestra una tendencia de disminución del error relativo (este se calculó con base en la solución más fina, según el artículo de Restrepo [35]), al aumentar los grados de libertad (número de malla). Sin embargo, un mejor análisis de la convergencia requiere una simulación con otras mallas más finas, el cual a su vez exige un software y/o una máquina más potente, en la figura (6.11) se aprecian estos valores para las mallas 20, 91 y 186.

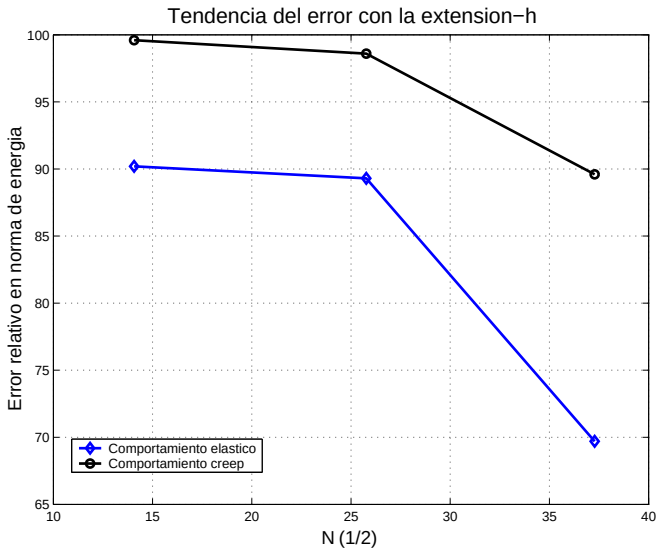


Figura 6.11: Convergencia en error relativo en norma de energía extensión-h

CAPÍTULO 7

Conclusiones y Trabajo Futuro

La modelación y simulación presentadas en este trabajo son un primer acercamiento a la predicción de la geometría final de un plato cerámico, sometido al proceso de cocción en el cual intervienen los subprocesos de sinterización y deformación mecánica elástica y de fluencia. En las principales publicaciones científicas en el ámbito cerámico, no se reportan aplicaciones de este tipo a la cerámica tradicional y por lo tanto se abre un campo de estudio en el diseño de elementos cerámicos tradicionales tales como vajillas, porcelana santitaria, revestimientos y piso; mediante el uso del método de elementos finitos para resolver los modelos implicados en este proceso.

Con base en la implementación realizada, se considera factible utilizar estas técnicas matemáticas en el diseño cerámico. Se cuenta con diversos modelos matemáticos que resueltos mediante el MEF y los métodos numéricos apropiados permiten obtener resultados coherentes y útiles. Para lograr utilizar esta técnica en la industria cerámica se requiere realizar una experimentación extensa y precisa para determinar las propiedades del material y contar con equipos de cómputo de alta capacidad de memoria y velocidad de procesamiento que permita hacer las diferentes simulaciones en un tiempo adecuado para el trabajo industrial. La programación del modelo que se realiza en Matlab es útil para iniciar este estudio, por ser un lenguaje amigable, pero, presenta deficiencias en el manejo óptimo de problemas que impliquen manipulación de matrices grandes como en nuestro caso. En nuestro estudio se alcanzaron a realizar simulaciones con grado polinomial seis equivalente a 910 grados de libertad y mallas con máximo 310 elementos equivalente a 2254 grados de libertad; estas tomaron hasta una semana para finalizar exitosamente.

Con relación a los métodos numéricos empleados complementarios al MEF tales como: iteración de Newton-Raphson, cuadratura de Gauss-Legendre, el método de penalización para solución de sistemas lineles, el método de los residuos mínimos ponderados y el método theta en la simulación dinámica, presentaron estabilidad y convergencia del error con las tolerancias

impuestas al modelo.

La modelación del proceso de sinterización mediante el principio de conservación de la masa explica satisfactoriamente el comportamiento real de la pieza cerámica durante el tratamiento térmico y permite predecir, partiendo de un diseño de la pieza en crudo, su geometría final con una variación de $\pm$ 1 % en sus dimenensiones versus el resultado experimental, ver [1].

La modelación y simulación del comportamiento mecánico elástico y de fluencia, se implementó con base en la teoría elástica y el modelo de Norton. Las propiedades físicas y las constantes del modelo creep, que como se reporta en la tabla (4.5) son extraidas, bien sea de la literatura para materiales genéricos cerámicos o a partir de pruebas uni-axiales, requieren ser mejoradas para obtener resultados más precisos y ajustados con los experimentales mediante el uso de instrumentos de medición de mejor precisón, y pruebas con mayor repetibilidad y reproductibilidad, cuyo alcance es propio de la Ingeniería Cerámica. Las extensiones-h y p realizadas muestran convergencia para los modelos que se pudieron simular; sus resultados comparados con los experimentales muestran diferencias ocacionadas por la incertidumbre de las propiedades de material.

Los trabajos futuros deben continuar en este campo de investigación debido a los beneficios potenciales para el diseño y la manufactura de productos de cerámica tradicional y pueden focalizarse en los siguientes aspectos:

i) Optimización de la programación para lograr tiempos de procesamiento apropiados para la industria.

ii) Estudio de las propiedades del material a partir de métodos estándarizados reproducibles y repetibles con instrumetos de alta precisión que den confiabilidad a los modelos empleados.

iii) Simular otros modelos de fluencia que incluyan variables termodinámicas y viscoelásticas.

iv) Una vez implementada la aplicación eficientemente, se puede establecer un conjunto de parámetros geométricos que restrinjan el diseño al tener en cuenta los efectos mecánicos de las geometrías desarrolladas. Otro campo de acción interesante es el análisis de esfuerzos y fallas mecánicas durante el proceso de cocción, que puede prevenir daños internos a los hornos; similarmente el análisis de esfuerzos residuales de las piezas que podrán prevenir fallas en el uso.

BIBLIOGRAFÍA

[1] Arango, N., Restrepo, J. L., López, N. and Carvajal, L. [2004], 'Sinterización Cerámica por Elementos Finitos', *Revista EIA* **2**, 67 – 74. ISSN: 1794-1237.

[2] Aydin, I., Briscoe, B. J. and Sanliturk, K. Y. [1997], 'Dimensional Variation of Die-Pressed Ceramic Green Compacts, Comparison of a fem with Experiment', *J. Eur. Ceramic* **17**(10), 1173–84.

[3] Bathe, K. [1982], *Finite Element Procedures in Engineering Analysis*, Prentice Hall, NJ-07632USA. 0-13-317305-4.

[4] Bazara [1982], *Programación Lineal*, Prentice Hall.

[5] Becker, A. A. [2001], *Understanding no-linear Finite Element Analysis though Illustrative Benchmarks*, NAFEMS, Antony Rowe Ltd, UK. ISBN:1-874376-35-2.

[6] Bickford, W. B. [1990], *Finite Element Method*, Richard D. Irwin Inc, USA. ISBN:0-256-14472-9.

[7] Braess, D. [English translation 1997], *Finite Element. Theory Fast Solvers and Applications in Solid Mechanics*, primera edn, Cambridge University Press, UK. ISBN:0-521-58187-7.

[8] Callister, W. D. [1985], *Materials Science and Engineering. An Introduction*, 2 edn, Ed Wiley, USA. ISBN=0-471-50488-2.

[9] Champion, E. R. [1992], *Finite Element Analysis in Manufacturing Engineering*, McGraw Hill Inc, USA. ISBN=0-07-010510-3.

[10] Chandrupatla, T. R. and Ashok D. Belegundu [1999], *Introducción al Estudio del Elemento Finito en Ingeniería*, segunda edn, Prentice Hall, Mexico+. ISBN:970-17-0260-3.

[11] Cook, R. D. [1981], *Concepts and Applications of Finite Element Analysis*, segunda edn, John Wiley and Sons.

[12] Dautray, R. and Lions, J.-L. [1984], *Mathematical and Analysis and Numerical Method for Science and Technology*, Vol. 4, Springer, Paris. ISBN:3-540-66100-x.

[13] Edwards, P. [1995], *Calculus Using Matlab*, cuarta edn, Ed Prentice Hall.

[14] Fagan, M. J. [1992], *Finite Element Analysis. Theory and Practics*, Longman, UK. ISBN:0-582-02247-9.

[15] Haddad, Y. M. [1995], *Viscoelasticity of Engineering Materials*, Ed Chapman and Hall, UK. ISBN=0-412-59030-1.

[16] Henshall, D.Ñ. [1996], 'Fea Calculations of Thermal Stresses in Whitewares Bodies During Cool Down', *University for the degree of MAster of science in Ceramic eng* p. 63.

[17] *How to Choose a Finite Element Analysis System* [1994]. NAFEMS.

[18] *How to Choose a Finite Element pre-post Processor* [1994]. NAFEMS.

[19] *How to Plan a FEA* [1994]. NAFEMS.

[20] *How to Start with Finite Element* [1994]. NAFEMS.

[21] *How to Undertake Creep Analysis with Finite Element* [1994]. NAFEMS.

[22] *Introducción al Método de los Elementos Finitos. Ingeniería Mecánica* [n.d.].

[23] Kingery, W. D., Bowenand, H. K. and Uhlmann, D. R. [1960], *Introduction to Ceramics*, second edn, John wiley and sons Inc, USA. ISBN:0-471-47860-1.

[24] Kraft, T. O., Coube and Riedel, H. [n.d.], 'Numerical simulation of pressing and sintering in the ceramic and hard metal industry. fraundhofer institute for materials mechanics', *Wotiler Strasse II* .

[25] Kwon, Y. W. and Bang, H. [1997], *The Finite Element Methods Using Matlab*, CRC press LLC, USA. 0-8493-9653-0.

[26] Levi, E. [1989], *Elementos de Mecánica del Medio Continuo*, Ed Limusa, México. ISBN:968-18-0609-3.

[27] Mitchell, V. [n.d.], *Mesh Generators in Two and Three Dimensions Algoritms*, Ed Limusa, Cornell University.

[28] Mohr, G. A. [1992], 'Finite Element for Solids, Fluids and Optimization', *University Press* . ISBN:0-19-856368-x.

[29] *Métodos Matemáticos Segundo Cuatrimestre Curso 2002-2003* [n.d.]. http:/matematicas.uclm.es/ind-cr/metmat/index.html.

[30] Nakamura, S. [1997], *Análisis Numérico y Visualización Gráfica con Matlab*, Prentice Hall, México.

[31] Navarro, J. E., Cantavella, V., Negre, F. and Sánchez, E. [1999], 'Model Predicts Tile Deformation in Firing', *Bulletin of Am.Ceramics Society* .

[32] Oñate, E. and Francisco Z. [2001], *Introducción al Método de los Elementos Finitos. Apuntes del Curso a Distancia, Métodos Numéricos para el Cálculo y Diseño en Ingeniería*, Universidad de Cataluña, España.

[33] Perssons, P. and Glibert Strang [2004], *A Simple Mesh Generator in Matlab*, Department of Mathematics, MIT, USA. http:/math.mit.edu/perssons/mesh.

[34] Restrepo, J. L. [1999], *Introducción al Método de los Elementos Finitos*, EAFIT, Medellín.

[35] Restrepo, J. L. [2002], 'Error en la Solución de Problemas de Elasticidad con el Método de los Elementos Finitos', *EAFIT* pp. 1–36.

[36] Sanliturk, K. Y., I.l Aydin and Briscoe, B. J. [1999], 'A Finite Element Approach for the Shape Compacts During Sinterin', *J. Am. Ceramic Soc* **82**, 1748–56.

[37] Schulle, W., Shultz, K. and Freiberg [1999], 'Fem Modeling of Firing Temperature and Stress Zones', *Cfi/Ber* (3).

[38] Szabo, B. and Ivo Babuska [1991], *Finite Element Analysis*, John Wiley and Sons, USA. ISBN:0-471-50273-1.

[39] Taheri, F. and Zhu, G. P. [1997], 'Simulation of Processing and Residual Stresses in a Ceramic Composite. Finite Element Simulation and Validation Part I y Part II', *Interceram* **46**(5,6), 317,436.

[40] Tsvelikh, A. W., Thompson Easton, A. and Freshwater, I. [1995], 'A Geometrical Finite Element Model of the Sintering Process of Advanced Ceramics', *Comput. Mater. Sci* **3**, 457–64.

[41] *Why do Finite Element Analysis* [1994]. NAFEMS.

[42] Zienkiewicz, O. C. [1997], *The Finite Element Method*, tercera edn, Ed McGraw-Hill, London. ISBN:0-07-084072-5.

APÉNDICE A

Funcional Lineal

Sea Y espacio lineal normado y

$$\mathscr{F} : Y \to \mathbb{R}$$

Decimos que $\mathscr{F}$ es un funcional lineal si tiene las siguientes propiedades:

1. $\mathscr{F}(v_1 + v_2) = \mathscr{F}(v_1) + \mathscr{F}(v_2) \qquad \forall\, v_2 \in Y$
2. $\mathscr{F}(\alpha v) = \alpha \mathscr{F}(v), \alpha \in \mathbb{R}, v \in Y$
3. $|\mathscr{F}(v)| \leq C||v||_Y$, donde C es independiente de v

APÉNDICE B

Forma Bilineal

Sean X y Y espacios lineales normados. Una expresión matemática $\mathscr{B}(u,v)$ es bilineal si es lineal respecto a cada variable

$$\mathscr{B}(u,v) : X \times Y \to \mathbb{R}$$

$\mathscr{B}(u,v)$ satisface las siguientes propiedades:

1. $\mathscr{B}(u_1 + u_2, v) = \mathscr{B}(u_1, v) + \mathscr{B}(u_2, v)$
2. $\mathscr{B}(u, v_1 + v_2) = \mathscr{B}(u, v_1) + \mathscr{B}(u, v_2)$
3. $\mathscr{B}(\alpha u, v) = \alpha \mathscr{B}(u, v)$
4. $\mathscr{B}(u, \alpha v) = \alpha \mathscr{B}(u, v)$
5. $|\mathscr{B}(u,v)| \leq C||u||_X \, ||v||_Y$; donde C es independiente de u y de v

APÉNDICE C

Funciones de Forma

Las funciones de forma utilizadas en la programación se relacionan en este apendice

Funciones de forma nodales

Función de forma	$\frac{\partial}{\partial \xi}$	$\frac{\partial}{\partial \eta}$
$N_1(\xi,\eta) = 1/4(1-\eta)(1-\xi)$	$-1/4(1-\eta)$	$1/4(1-\xi)$
$N_2(\xi,\eta) = 1/4(1-\eta)(1+\xi)$	$1/4(1-\eta)$	$-1/4(1+\xi)$
$N_3(\xi,\eta) = 1/4(1+\eta)(1+\xi)$	$1/4(1+\eta)$	$1/4(1+\xi)$
$N_4(\xi,\eta) = 1/4(1+\eta)(1-\xi)$	$-1/4(1+\eta)$	$1/4(1-\xi)$

Funciones de Forma de los Lados

Función de forma lado 1
$N_2^1(\xi,\eta) = 1/4 * (3/2)^{1/2}(1-\eta)(\xi^2-1)$
$N_3^1(\xi,\eta) = 1/4 * (5/2)^{1/2}(1-\eta)(\xi^3-\xi)$
$N_4^1(\xi,\eta) = 1/16 * (7/2)^{1/2}(1-\eta)(5\xi^4-6\xi^2+1)$
$N_5^1(\xi,\eta) = 1/16 * (9/2)^{1/2}(1-\eta)(7\xi^5-11\xi^3+3\xi)$
$N_6^1(\xi,\eta) = 1/32 * (11/2)^{1/2}(1-\eta)(21\xi^6-35\xi^4+15\xi^2-1)$
$N_7^1(\xi,\eta) = 1/32 * (13/2)^{1/2}(1-\eta)(33\xi^7-63\xi^5+35\xi^3-5\xi)$
$N_8^1(\xi,\eta) = 1/256 * (15/2)^{1/2}(1-\eta)(429\xi^8-924\xi^6+630\xi^4-140\xi^2+5)$

Derivada parcial con respecto a ξ del lado 1
$N_2^1(\xi,\eta)' = 1/4 * (3/2)^{1/2}(1-\eta)2\xi$
$N_3^1(\xi,\eta)' = 1/4 * (5/2)^{1/2}(1-\eta)(3\xi^2-1)$
$N_4^1(\xi,\eta)' = 1/16 * (7/2)^{1/2}(1-\eta)(20\xi^3-12\xi)$
$N_5^1(\xi,\eta)' = 1/16 * (9/2)^{1/2}(1-\eta)(35\xi^4-30\xi^2+3)$
$N_6^1(\xi,\eta)' = 1/32 * (11/2)^{1/2}(1-\eta)(126\xi^5-140\xi^3+30\xi)$
$N_7^1(\xi,\eta)' = 1/32 * (13/2)^{1/2}(1-\eta)(231\xi^6-315\xi^4+105\xi^2-5)$
$N_8^1(\xi,\eta)' = 1/256 * (15/2)^{1/2}(1-\eta)(3432\xi^7-5544\xi^5+2520\xi^3-280\xi)$

Derivada parcial con respecto a η del lado 1
$N_2^1(\xi,\eta)' = -1/4 * (3/2)^{1/2}(\xi^2 - 1)$
$N_3^1(\xi,\eta)' = -1/4 * (5/2)^{1/2}(\xi^3 - \xi)$
$N_4^1(\xi,\eta)' = -1/16 * (7/2)^{1/2}(5\xi^4 - 6\xi^2 + 1)$
$N_5^1(\xi,\eta)' = -1/16 * (9/2)^{1/2}(7\xi^5 - 10\xi^3 + 3\xi)$
$N_6^1(\xi,\eta)' = -1/32 * (11/2)^{1/2}(21\xi^6 - 35\xi^4 + 15\xi^2 - 1)$
$N_7^1(\xi,\eta)' = -1/32 * (13/2)^{1/2}(33\xi^7 - 63\xi^5 + 35\xi^3 - 5\xi)$
$N_8^1(\xi,\eta)' = -1/256 * (15/2)^{1/2}(429\xi^8 - 924\xi^6 + 630\xi^4 - 140\xi^2 + 5)$

Función de forma lado 2
$N_2^2(\xi,\eta) = 1/4 * (3/2)^{1/2}(1-\xi)(\eta^2 - 1)$
$N_3^2(\xi,\eta) = 1/4 * (5/2)^{1/2}(1-\xi)(\eta^3 - \eta)$
$N_4^2(\xi,\eta) = 1/16 * (7/2)^{1/2}(1-\xi)(5\eta^4 - 6\eta^2 + 1)$
$N_5^2(\xi,\eta) = 1/16 * (9/2)^{1/2}(1-\xi)(7\eta^5 - 11\eta^3 + 3\eta)$
$N_6^2(\xi,\eta) = 1/32 * (11/2)^{1/2}(1-\xi)(21\eta^6 - 35\eta^4 + 15\eta^2 - 1)$
$N_7^2(\xi,\eta) = 1/32 * (13/2)^{1/2}(1-\xi)(33\eta^7 - 63\eta^5 + 35\eta^3 - 5)$
$N_8^2(\xi,\eta) = 1/256 * (15/2)^{1/2}(1-\xi)(429\eta^8 - 924\eta^6 + 630\eta^4 - 140\eta^2 + 5)$

Derivada parcial con respecto a η del lado 2
$N_2^2(\xi,\eta)' = 1/4 * (3/2)^{1/2}(1-\xi)2\eta$
$N_3^2(\xi,\eta)' = 1/4 * (5/2)^{1/2}(1-\xi)(3\eta^2 - 1)$
$N_4^2(\xi,\eta)' = 1/16 * (7/2)^{1/2}(1-\xi)(20\eta^3 - 12\eta)$
$N_5^2(\xi,\eta)' = 1/16 * (9/2)^{1/2}(1-\xi)(35\eta^4 - 30\eta^2 + 3)$
$N_6^2(\xi,\eta)' = 1/32 * (11/2)^{1/2}(1-\xi)(126\eta^5 - 140\eta^3 + 30\eta)$
$N_7^2(\xi,\eta)' = 1/32 * (13/2)^{1/2}(1-\xi)(231\eta^6 - 315\eta^4 + 105\eta^2 - 5)$
$N_8^2(\xi,\eta)' = 1/256 * (15/2)^{1/2}(1-\xi)(3432\eta^7 - 5544\eta^5 + 2520\eta^3 - 280\eta)$

Derivada parcial con respecto a ξ del lado 2
$N_2^2(\xi,\eta)' = -1/4 * (3/2)^{1/2}(\eta^2 - 1)$
$N_3^2(\xi,\eta)' = -1/4 * (5/2)^{1/2}(\eta^3 - \eta)$
$N_4^2(\xi,\eta)' = -1/16 * (7/2)^{1/2}(5\eta^4 - 6\eta^2 + 1)$
$N_5^2(\xi,\eta)' = -1/16 * (9/2)^{1/2}(7\eta^5 - 10\eta^3 + 3\eta)$
$N_6^2(\xi,\eta)' = -1/32 * (11/2)^{1/2}(21\eta^6 - 35\eta^4 + 15\eta^2 - 1)$
$N_7^2(\xi,\eta)' = -1/32 * (13/2)^{1/2}(33\eta^7 - 63\eta^5 + 35\eta^3 - 5\eta)$
$N_8^2(\xi,\eta)' = -1/256 * (15/2)^{1/2}(429\eta^8 - 924\eta^6 + 630\eta^4 - 140\eta^2 + 5)$

Función de forma lado 3
$N_2^3(\xi,\eta) = 1/4 * (3/2)^{1/2}(1+\eta)(\xi^2-1)$
$N_3^3(\xi,\eta) = 1/4 * (5/2)^{1/2}(1+\eta)(\xi^3-\xi)$
$N_4^3(\xi,\eta) = 1/16 * (7/2)^{1/2}(1+\eta)(5\xi^4-6\xi^2+1)$
$N_5^3(\xi,\eta) = 1/16 * (9/2)^{1/2}(1+\eta)(7\xi^5-11\xi^3+3\xi)$
$N_6^3(\xi,\eta) = 1/32 * (11/2)^{1/2}(1+\eta)(21\xi^6-35\xi^4+15\xi^2-1)$
$N_7^3(\xi,\eta) = 1/32 * (13/2)^{1/2}(1+\eta)(33\xi^7-63\xi^5+35\xi^3-5\xi)$
$N_8^3(\xi,\eta) = 1/256 * (15/2)^{1/2}(1+\eta)(429\xi^8-924\xi^6+630\xi^4-140\xi^2+5)$

Derivada parcial con respecto a ξ del lado 3
$N_2^3(\xi,\eta)' = 1/4 * (3/2)^{1/2}(1+\eta)2\xi$
$N_3^3(\xi,\eta)' = 1/4 * (5/2)^{1/2}(1+\eta)(3\xi^2-1)$
$N_4^3(\xi,\eta)' = 1/16 * (7/2)^{1/2}(1+\eta)(20\xi^3-12\xi)$
$N_5^3(\xi,\eta)' = 1/16 * (9/2)^{1/2}(1+\eta)(35\xi^4-30\xi^2+3)$
$N_6^3(\xi,\eta)' = 1/32 * (11/2)^{1/2}(1+\eta)(126\xi^5-140\xi^3+30\xi)$
$N_7^3(\xi,\eta)' = 1/32 * (13/2)^{1/2}(1+\eta)(231\xi^6-315\xi^4+105\xi^2-5)$
$N_8^3(\xi,\eta)' = 1/256 * (15/2)^{1/2}(1+\eta)(3432\xi^7-5544\xi^5+2520\xi^3-280\xi)$

Derivada parcial con respecto a η del lado 3
$N_2^3(\xi,\eta)' = 1/4 * (3/2)^{1/2}(\xi^2-1)$
$N_3^3(\xi,\eta)' = 1/4 * (5/2)^{1/2}(\xi^3-\xi)$
$N_4^3(\xi,\eta)' = 1/16 * (7/2)^{1/2}(5\xi^4-6\xi^2+1)$
$N_5^3(\xi,\eta)' = 1/16 * (9/2)^{1/2}(7\xi^5-10\xi^3+3\xi)$
$N_6^3(\xi,\eta)' = 1/32 * (11/2)^{1/2}(21\xi^6-35\xi^4+15\xi^2-1)$
$N_7^3(\xi,\eta)' = 1/32 * (13/2)^{1/2}(33\xi^7-63\xi^5+35\xi^3-5\xi)$
$N_8^3(\xi,\eta)' = 1/256 * (15/2)^{1/2}(429\xi^8-924\xi^6+630\xi^4-140\xi^2+5)$

Función de forma lado 4
$N_2^4(\xi,\eta) = 1/4 * (3/2)^{1/2}(1+\xi)(\eta^2-1)$
$N_3^4(\xi,\eta) = 1/4 * (5/2)^{1/2}(1+\xi)(\eta^3-\eta)$
$N_4^4(\xi,\eta) = 1/16 * (7/2)^{1/2}(1+\xi)(5\eta^4-6\eta^2+1)$
$N_5^4(\xi,\eta) = 1/16 * (9/2)^{1/2}(1+\xi)(7\eta^5-11\eta^3+3\eta)$
$N_6^4(\xi,\eta) = 1/32 * (11/2)^{1/2}(1+\xi)(21\eta^6-35\eta^4+15\eta^2-1)$
$N_7^4(\xi,\eta) = 1/32 * (13/2)^{1/2}(1+\xi)(33\eta^7-63\eta^5+35\eta^3-5\eta)$
$N_8^4(\xi,\eta) = 1/256 * (15/2)^{1/2}(1+\xi)(429\eta^8-924\eta^6+630\eta^4-140\eta^2+5)$

Derivada parcial con respecto a η del lado 4
$N_2^4(\xi,\eta)' = 1/4 * (3/2)^{1/2}(1+\xi)2\eta$
$N_3^4(\xi,\eta)' = 1/4 * (5/2)^{1/2}(1+\xi)(3\eta^2-1)$
$N_4^4(\xi,\eta)' = 1/16 * (7/2)^{1/2}(1+\xi)(20\eta^3-12\eta)$
$N_5^4(\xi,\eta)' = 1/16 * (9/2)^{1/2}(1+\xi)(35\eta^4-30\eta^2+3)$
$N_6^4(\xi,\eta)' = 1/32 * (11/2)^{1/2}(1+\xi)(126\eta^5-140\eta^3+30\eta)$
$N_7^4(\xi,\eta)' = 1/32 * (13/2)^{1/2}(1+\xi)(231\eta^6-315\eta^4+105\eta^2-5)$
$N_8^4(\xi,\eta)' = 1/256 * (15/2)^{1/2}(1+\xi)(3432\eta^7-5544\eta^5+2520\eta^3-280\eta)$

Derivada parcial con respecto a ξ del lado 4
$N_2^4(\xi,\eta)' = 1/4 * (3/2)^{1/2}(\eta^2-1)$
$N_3^4(\xi,\eta)' = 1/4 * (5/2)^{1/2}(\eta^3-\eta)$
$N_4^4(\xi,\eta)' = 1/16 * (7/2)^{1/2}(5\eta^4-6\eta^2+1)$
$N_5^4(\xi,\eta)' = 1/16 * (9/2)^{1/2}(7\eta^5-10\eta^3+3\eta)$
$N_6^4(\xi,\eta)' = 1/32 * (11/2)^{1/2}(21\eta^6-35\eta^4+15\eta^2-1)$
$N_7^4(\xi,\eta)' = 1/32 * (13/2)^{1/2}(33\eta^7-63\eta^5+35\eta^3-5\eta)$
$N_8^4(\xi,\eta)' = 1/256 * (15/2)^{1/2}(429\eta^8-924\eta^6+630\eta^4-140\eta^2+5)$

Funciones de forma internas

Función de forma
$N_1(\xi,\eta) = 3/8(\xi^2-1)(\eta^2-1)$
$N_2(\xi,\eta) = \sqrt{15}/8(\xi^3/3-\xi-2/3)(\eta^2-1)$
$N_3(\xi,\eta) = \sqrt{15}/8(\eta^3/3-\eta-2/3)(\xi^2-1)$
$N_4(\xi,\eta) = \sqrt{21}/32(5\xi^4-6\xi^2+1)(\eta^2-1)$
$N_5(\xi,\eta) = 5/8 * (\xi^3/3-\xi-2/3)(\eta^3/3-\eta-2/3)$
$N_6(\xi,\eta) = \sqrt{21}/32(\xi^2-1)(5\eta^4-6\eta^2-1)$
$N_7(\xi,\eta) = 3\sqrt{3}/32(7\xi^5-10\xi^3+3\xi)(\eta^2-1)$
$N_8(\xi,\eta) = \sqrt{35}/32(5\xi^4-6\xi^2+1)(\eta^3/3-\eta-2/3)$
$N_9(\xi,\eta) = \sqrt{35}/32(5\eta^4-6\eta^2+1)(\xi^2/3-\xi-2/3)$
$N_10(\xi,\eta) = 3\sqrt{3}/32(7\eta^5-10\eta^3+3\eta)(\xi^2-1)$
$N_11(\xi,\eta) = 1/64\sqrt{33/2}(21\xi^6-35\xi^4+15\xi^2-1)(\eta^2-1)$
$N_12(\xi,\eta) = 3\sqrt{5}/32(7\xi^5-10\xi^3+3\xi)(\eta^2/3-\eta-2/3)$
$N_13(\xi,\eta) = 7/128(5\eta^4-6\eta^2+1)(5\xi^4-6\xi^2+1)$
$N_14(\xi,\eta) = 3\sqrt{5}/32(\xi^2/3-\xi-2/3)(7\eta^5-10\eta^3+3\eta)$
$N_15(\xi,\eta) = \sqrt{33}/64(\xi^2-1)(21\eta^6-35\eta^4+15\eta^2-1)$

Derivada parcial con respecto a ξ
$N_1(\xi,\eta)' = 3/4(\xi)(\eta^2-1)$
$N_2(\xi,\eta)' = \sqrt{15}/8(\xi^2-1)(\eta^2-1)$
$N_3(\xi,\eta)' = \sqrt{15}/4(\eta^3/3-\eta-2/3)(\xi)$
$N_4(\xi,\eta)' = \sqrt{21}/32(20\xi^3-12\xi)(\eta^2-1)$
$N_5(\xi,\eta)' = 5/8(\xi^2-1)(\eta^3/3-\eta-2/3)$
$N_6(\xi,\eta)' = \sqrt{21}/16(\xi)(5\eta^4-6\eta^2-1)$
$N_7(\xi,\eta)' = 3\sqrt{3}/32(35\xi^4-30\xi^2+3)(\eta^2-1)$
$N_8(\xi,\eta)' = \sqrt{35}/32(20\xi^3-12\xi)(\eta^3/3-\eta-2/3)$
$N_9(\xi,\eta)' = \sqrt{35}/32(5\eta^4-6\eta^2+1)(2\xi/3-1$
$N_10(\xi,\eta)' = 3\sqrt{3}/16(7\eta^5-10\eta^3+3\eta)(\xi)$
$N_11(\xi,\eta)' = 1/64\sqrt{33/2}(63\xi^5-70\xi^3+15\xi)(\eta^2-1)$
$N_12(\xi,\eta)' = 3\sqrt{5}/32(35\xi^4-30\xi^2+3)(\eta^2/3-\eta-2/3)$
$N_13(\xi,\eta)' = 7/128(5\eta^4-6\eta^2+1)(20\xi^3-12\xi)$
$N_14(\xi,\eta)' = 3\sqrt{5}/32(2\xi/3-1)(7\eta^5-10\eta^3+3\eta)$
$N_15(\xi,\eta)' = \sqrt{36}/32\xi(21\eta^6-35\eta^4+15\eta^2-1)$

Derivada parcial con respecto a η
$N_1(\xi,\eta)' = 3/4(\xi^2-1)(\eta)$
$N_2(\xi,\eta)' = \sqrt{15}/4(\xi^3/3-\xi-2/3)(\eta)$
$N_3(\xi,\eta)' = \sqrt{15}/8(\eta^2-1)(\xi^2-1)$
$N_4(\xi,\eta)' = \sqrt{21}/16(5\xi^4-6\xi^2+1)(\eta)$
$N_5(\xi,\eta)' = 5/8(\xi^3/3-\xi-2/3)(\eta^2-1)$
$N_6(\xi,\eta)' = \sqrt{21}/32(\xi^2-1)(20\eta^3-12\eta)$
$N_7(\xi,\eta)' = 3\sqrt{3}/16(7\xi^5-10\xi^3+3\xi)(\eta)$
$N_8(\xi,\eta)' = \sqrt{35}/32(5\xi^4-6\xi^2+1)(\eta^2-1)$
$N_9(\xi,\eta)' = \sqrt{35}/32(20\eta^3-12\eta)(\xi^3/3-\xi-2/3)$
$N_10(\xi,\eta)' = 3\sqrt{3}/32(35\eta^4-30\eta^2+3)(\xi^2-1)$
$N_11(\xi,\eta)' = 1/64\sqrt{33/2}(21\xi^6-35\xi^4+15\xi^2-1)(\eta)$
$N_12(\xi,\eta)' = 3\sqrt{5}/32(7\xi^5-10\xi^3+3\xi)(2\eta/3-1)$
$N_13(\xi,\eta)' = 7/128(20\eta^3-12\eta)(5\xi^4-6\xi^2+1)$
$N_14(\xi,\eta)' = 3\sqrt{5}/32(\xi^2/3-\xi-2/3)(35\eta^4-30\eta^2+3)$
$N_15(\xi,\eta)' = \sqrt{33}/64(\xi^2-1)(63\eta^5-70\eta^3+15\eta)$

APÉNDICE D

Integración Numérica

Para la solución de algunos problemas usando elementos finitos se requieren calcular integrales que por los métodos tradicionales no es posible calcular, es por esto que se deben resolver por métodos numéricos, tales como Newton - Cote, Gauss - Lobato, Gauss - legendre entre otros. En este trabajo se utiliza el método de Gauss - Legendre por ser uno de los que alcanza mayor precisión; a continuación se explicará este método.
Se supone que se quiere calcular la integral

$$I = \int_{-1}^{1} f(\xi)\, d\xi$$

Dividiendo el entervalo $[-1, 1]$ en n partes y haciendo pasar un polinomio por los valores de la función en esos puntos, así

$$I \approx \sum_{i=1}^{n} w_i f(x_i)$$

donde w_i son los valores de los polinomios de Legendre evaluados en cada punto, w_i se conoce como factores de peso. x_i están localizados simetricamente en $[-1, 1]$ y son las raíces del polinomio de Legendre, ver[42]
Para n puntos se tienen $2n$ incógnitas (f_i, x_i) y por tanto se puede formar un polinomio de grado $2n - 1$ y obtener su integral exacta.
Si se quiere calcular

$$I = \int_{-1}^{1} \int_{-1}^{1} f(\xi, \eta)\, d\xi\, d\eta \tag{D.1}$$

se puede usar el procedimiento anterior manteniendo η constante, es decir,

$$\int_{-1}^{1} f(\xi, \eta)\, d\xi = \sum_{j=1}^{n} w_j f(x_j, \eta) = \psi(\eta) \tag{D.2}$$

Luego

$$I = \int_{-1}^{1} \psi(\eta)\, d\eta = \sum_{i=1}^{n} \psi(x_i) = \sum_{1}^{n} w_i \sum_{j=1}^{n} w_i f(x_j, x_i) = \sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j f(x_j, x_i) \tag{D.3}$$

Las abscisas x_i se conocen como puntos de Gauss.
En el computo de la matriz de rigidez 4.34 es necesario evular integrales de la forma

$$I = \iint_{\Omega_k} F(x,y) t_z(x,y)\, dx\, dy \tag{D.4}$$

donde Ω_k representa el dominio del k - ésimo elemento, $t_z(x,y) = t_z$ es el espesor. El mapeo $x = Q_x^{(k)}(\xi,\eta), y = Q_y^{(k)}(\xi,\eta)$ transforma esta expresión y la integral es desarrollada en el elemento estándar Ω_{st}:

$$I = \iint_{\Omega_{st}} F(Q_x^{(k)}(\xi,\eta), Q_y^{(k)}(\xi,\eta))\; t_z\,(Q_x^{(k)}(\xi,\eta), Q_y^{(k)}(\xi,\eta))|J|\, d\xi\, d\eta \tag{D.5}$$

donde $|J|$ es el determiniante de la matriz Jacobiana

$$[J] = \begin{pmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{pmatrix}$$

Se Verá que [D.4] y [D.5] son equivalentes. Se denota el vector posición por $\vec{r} = \vec{x_i} + \vec{y_j} + \vec{z_k}$. El diferencial de volumen está dado por:

$$dV = \left(\frac{\partial \vec{r}}{\partial x}\, dx\right) \times \left(\frac{\partial \vec{r}}{\partial y}\, dy\right) \bullet \left(\frac{\partial \vec{r}}{\partial z}\, dz\right) = dx\, dy\, dz \tag{D.6}$$

donde $\times$ denota el producto cruz. Cuando se usa el mapeo

$$x = Q_x^{(k)}(\xi,\eta),\ y = Q_y^{(k)}(\xi,\eta),\ z = \zeta$$

entonces $\vec{r}$está en términos de ξ, η, ζ y se tiene:

$$\begin{aligned} dV &= \left(\frac{\partial \vec{r}}{\partial \xi}\, d\xi\right) \times \left(\frac{\partial \vec{r}}{\partial \eta}\, d\eta\right) \bullet \left(\frac{\partial \vec{r}}{\partial \zeta}\, d\zeta\right) \\ &= \left(\frac{\partial x}{\partial \xi}\vec{i} + \frac{\partial y}{\partial \xi}\vec{j}\right) d\xi \times \left(\frac{\partial x}{\partial \eta}\vec{i} + \frac{\partial y}{\partial \eta}\vec{j}\right) d\eta\, \vec{k}\, d\zeta \\ &= \left(\frac{\partial x}{\partial \xi}\frac{\partial y}{\partial \eta} - \frac{\partial x}{\partial \eta}\frac{\partial y}{\partial \xi}\right) d\xi\, d\eta\, d\zeta \end{aligned}$$

Como en problemas en dos dimensiones la integral es independiente de z (equivalentemente ζ) se puede escribir

$$dV = |J| t_z \, d\xi \, d\eta$$

Usualmente las integrales son evaluadas numericamente por medio de la cuadratura Gaussiana. Se asume que Ω_{st} es el cuadrilatero estándar y así

$$I = \int_{-1}^{1} \int_{-1}^{1} F\left(Q_x^{(k)}(\xi,\eta),\, Q_y^{(k)}(\xi,\eta)\right) |J| \; t_z\left(Q_x^{(k)}(\xi,\eta),\, Q_y^{(k)}(\xi,\eta)\right) d\xi \, d\eta$$

para cualquier η fijo se puede integrar numericamente con respecto a ξ y luego respecto a η

$$I \cong \sum_{r=1}^{n} \sum_{s=1}^{m} w_r w_s \left[F\left(Q_x^{(k)}(\xi,\eta),\, Q_y^{(k)}(\xi,\eta)\right) \; t_z \; \left(Q_x^{(k)}(\xi,\eta),\, Q_y^{(k)}(\xi,\eta)\right) |J|\right]$$

$\xi = \xi_s$, $\eta = \eta_r$, donde w_s, w_r, ξ y ξ_r son los pesos y las abscisas de los puntos de Gauss en las direcciones ξ y η; m y n son el número de puntos de Gauss en las direcciones ξ y η respectivamente.

APÉNDICE E

Código en Matlab

Archivo: Bilineal.m

```
%bilineal.m   Version Definitiva Oct 10-2004
%-------------------------------------------------------------------
%  Propósito:
% Calcula la funcion bilineal en B(u,u) para la norma de energia.
%
%  Sintaxis:
% Es llamado directamente dentro del programa mecanico.
%
%  Descripción de variables:
%       a0e: Vector de desplazamientos elasticos.
%       BUU: Valor de la funcion bilineal.
% conec: Matriz de conectividades. Dim (nelem,nne).
% conecn: vector temporal para almacenar los vectores movidos.
% e, k, intr, intz, CNG: Contadores.
%       kij: Matriz de rigidez elemental. Dim (2*N,2*N)
% nelem: Número de elementos del sistema
%       ngle: Grado de libertad local
%       nggl: Grado de libertad global
% nne: Número de nodos/elemento.
%       rt: radio transformado
%       w: pesos de la cuadratura de Gauss.

%-------------------------------------------------------------------
```

```
%Inicializa la funcion Bilineal
BUU=0;

%Abre archivos de lectura de datos.
fid1=fopen('kij','r');

for e=1:nelem
   kij=fread(fid1,[2*N,2*N],'real*8'); %Lectura matrices de rigidez
   for k=1:nne
      c=conec(e,k);
      ngle(k)=nggl(c,1);
      ngle(k+nne)=nggl(c,2);
      a0e(k)=ae(ngle(k));
      a0e(k+nne)=ae(ngle(k+nne));
  end
  CNG=0;
  for intr=1:NG
      s=p(intr);
      for intz=1:NG
          n=p(intz);
          CNG=CNG+1;
          BUU=BUU+w(intr)*w(intz)*a0e*kij*a0e'*rt(CNG,e);
      end
  end
end

%Cierra archivo
fclose(fid1);

%Borrar variables locales
clear fid e n k intz intr CNG
```

Archivo: Def-tens.m

```
%Def_tens.m   Version definitiva Dic de 2004

%-------------------------------------------------------------------
%  Propósito:
% calcular las deformaciones y tensiones elásticas.
%
%  Sintaxis:
% function [Eu,tension]=Def_tens(nelem,D,conec,a,gp,nne,N,NG).
%
```

```
%  Descripción de variables:
%       a: Vector de desplazamientos.
% C, e, ina, intr, intz: Contadores.
% conec: Matriz de conectividades de nodos
% Eu: Deformación elástica. Dim(4,nelem).
% gp: Grado polinomial
% conec: Matriz de conectividades. Dim (nelem,nne).
% conecn: vector temporal para almacenar los vectores movidos.
% Jinv: Matriz Jacobiana inversa
% nelem: Número de elementos del sistema
% nelem: Número de elementos del sistema
%       ngle: Grado de libertad local
%       nggl: Grado de libertad global
% tension: matriz de tensiones . Dim(4,nelem).
% rt: vector r transformado.
% rzs: Matriz de coordenadas sinterizadas
%-------------------------------------------------------------------
function [Eu,tension]=Def_tens(nelem,D,conec,a,gp,nne,N,NG)
global rzs nggl NI DNs DNn rt

%Inicialización de matriz.
C=0;
Eu=zeros(4,nelem*NG*NG); %Matriz de deformaciones elásticas

%Apertura del archivo Jinv
fid1=fopen('Jinv','r');

%Calculo de las deformaciones y tensiones
for e=1:nelem
    CNG=0;
   %Extracción de los coeficientes {ai} para el elemento
   for ina=1:nne
      ai(ina,1)=a(nggl(conec(e,ina),1));
      ai(ina+nne,1)=a(nggl(conec(e,ina),2));
  end
  for intr=1:NG
      for intz=1:NG
          CNG=CNG+1;
          C=C+1;
          Jinv=fread(fid1,[2,2],'real*8');
          [mtxcin]=f_matriz_cinematica(N,rt(CNG,e),Jinv,CNG);
          Eu(:,C)=Eu(:,C)+(ai'*mtxcin')';
      end
```

```
  end
end

%Calculo de las tensiones elásticas
tension=D*Eu;

fclose(fid1);

%Elimina variables locales
clear CNG C e intr intz Jinv
```

Archivo: f-form

```
%f_form.m

%-------------------------------------------------------------------
%  Propósito:
% Subpragrama para decidir que funciones de forma usar en el calculo
% de la matriz de rigidez. Trabaja hasta con grado polinomial gp=10.
%
%  Sintaxis:
% function [NI,DNs,DNn]=f_form(gp,N,n,s,w). Se requieren conocer
% los parametros gp,N,n,s,w.
%
%  Descripción de variables:
% gp: Grado polinomial.
% N: Cantidad de funciones de forma.
% n,s: Valores de los puntos de Gauss.
% w: Pesos para la integración.
% NI: Valor de la función de forma en esos puntos de Gauss.
% DNs, DNn: Valores de las derivadas con respecto a las coordenadas
% naturales
%-------------------------------------------------------------------

%Decision de que subprograma llamar dependiendo del grado polinomial
%a usar

function [NI,DNs,DNn]=f_form(gp,N,p,NG)
if gp==1
   p1
else
    if gp==2
        p2
```

```
    else
        if gp==3
            p3
        else
            if gp==4
                p4
            else
                if gp==5
                    p5
                else
                    if gp==6
                        p6
                    else
                        if gp==7
                            p7
                        else
                            if gp==8
                                p8
                            else
                                break
                            end
                        end
                    end
                end
            end
        end
    end
end
```

Archivo: f-matriz-cinematica

```
function [mtxcin]=f_matriz_cinematica(N,rt,Jinv,MCNG)

%-------------------------------------------------------------
%  Propósito:
%     Determinar las ecuaciones
%
%  Sintaxis:
%     [mtxcin]=f_matriz-cinetmatica(N,MNI,MDNs,MDNn,rt,Jinv,MCNG)
%
%  Variable Description:
%     MNI - Valores de las funciones de forma
%     MDNs y MDNn - derivadas de las funciones de forma con respecto a
```

```
%     variables locales
%     rt - Distancia radial transformada
%-------------------------------------------------------------------
global NI DNs DNn
%Inicialización matriz cinemática.
mtxcin=zeros(4,2*N);

%Calculo de las derivadas con respecto a las variables globales

for i=1:N
   DNr(i)=Jinv(1,1)*DNs(i,MCNG)+Jinv(1,2)*DNn(i,MCNG);
   DNz(i)=Jinv(2,1)*DNs(i,MCNG)+Jinv(2,2)*DNn(i,MCNG);
end

%Calculo de la matriz cinemática
for i=1:N
   mtxcin(1,i)=DNr(i);
   mtxcin(2,i)=NI(i,MCNG)/rt;
   mtxcin(3,i+N)=DNz(i);
   mtxcin(4,i)=DNz(i);
   mtxcin(4,i+N)=DNr(i);
end
```

Archivo: gmres

```
function [x,flag,relres,iter,resvec] =
%gmres(b,restart,tol,maxit,M1,M2,x0,varargin)
%GMRES  Generalized Minimum Residual Method.
%   X = GMRES(kijg,b) attempts to solve the system of linear equations
    %kijg*X = b for
%   X.  The N-by-N coefficient matrix kijg must be square and the right
    %hand side
%   column vector b must have length N.  A may be a function returning
    %kijg*X. This
%   uses the unrestarted method with MIN(N,10) total iterations.
%
%   GMRES(RESTART) restarts the method every RESTART iterations.
    %If RESTART
%   is N or [] then GMRES uses the unrestarted method as above.
%
%   GMRES(kijg,b,RESTART,TOL) specifies the tolerance of the method.
    %If TOL is []
```

```
%   then GMRES uses the default, 1e-6.
%
%   GMRES(kijg,b,RESTART,TOL,MAXIT) specifies the maximum number of
    %outer
%   iterations. Note: the total number of iterations is RESTART*MAXIT.
    %If MAXIT
%   is [] then GMRES uses the default, MIN(N/RESTART,10). If RESTART
    %is N or []
%   then the total number of iterations is MAXIT.
%
%   GMRES(kijg,b,RESTART,TOL,MAXIT,M) and GMRES(kijg,B,RESTART,TOL,
    %MAXIT,M1,M2)
%   use preconditioner M or M=M1*M2 and effectively solve the
%   system inv(M)*kijg*X = inv(M)*B for X.  If M is [] then a
    %preconditioner is
%   not applied.  M may be a function returning M\X.
%
%   GMRES(kijg,b,RESTART,TOL,MAXIT,M1,M2,X0) specifies the first initial
%   guess.  If X0 is [] then GMRES uses the default, an all zero vector.
%
%   GMRES(AFUN,b,RESTART,TOL,MAXIT,M1FUN,M2FUN,X0,P1,P2,...) passes
    %parameters
%   to functions: AFUN(X,P1,P2,...), M1FUN(X,P1,P2,...),
% M2FUN(X,P1,P2,...).
%
%   [X,FLAG] = GMRES(kijg,b,RESTART,TOL,MAXIT,M1,M2,X0) also returns a
    %convergence
%   FLAG:
%    0 GMRES converged to the desired tolerance TOL within MAXIT
    %iterations.
%    1 GMRES iterated MAXIT times but did not converge.
%    2 preconditioner M was ill-conditioned.
%    3 GMRES stagnated (two consecutive iterates were the same).
%
%   [X,FLAG,RELRES] = GMRES(kijg,B,RESTART,TOL,MAXIT,M1,M2,X0) also
%returns
%   the relative residual NORM(B-kijg*X)/NORM(B).  If FLAG is 0,
    %RELRES <= TOL.
%
%   [X,FLAG,RELRES,ITER] = GMRES(kijg,B,RESTART,TOL,MAXIT,M1,M2,X0) also
%   returns both the outer and inner iteration numbers at which X was
%   computed: 0 <= ITER(1) <= MAXIT and 0 <= ITER(2) <= RESTART.
%
```

```
%   [X,FLAG,RELRES,ITER,RESVEC] = GMRES(kijg,B,RESTART,TOL,
%MAXIT,M1,M2,X0)
    %also
%   returns a vector of the residual norms at each inner iteration,
    %including
%   NORM(B-kijg*X0).
%
%   Example:
%      kijg = gallery('wilk',21);  b = sum(kijg,2);
%      tol = 1e-12;  maxit = 15; M1 = diag([10:-1:1 1 1:10]);
%      x = gmres(kijg,b,10,tol,maxit,M1,[],[]);
%   Alternatively, use this matrix-vector product function
%      function y = afun(x,n)
%      y = [0; x(1:n-1)] + [((n-1)/2:-1:0)'; (1:(n-1)/2)'] .*x +

    %[x(2:n); 0];
%   and this preconditioner backsolve function
%      function y = mfun(r,n)
%      y = r ./ [((n-1)/2:-1:1)'; 1; (1:(n-1)/2)'];
%   as inputs to GMRES
%      x1 = gmres(@afun,b,10,tol,maxit,@mfun,[],[],21);
%
%   See also BICG, BICGSTAB, CGS, LSQR, MINRES, PCG, QMR, SYMMLQ,
% LUINC, @.

%   Copyright 1984-2002 The MathWorks, Inc.
%   $Revision: 1.21 $ $Date: 2002/04/09 00:26:12 $
    %global kijg

if (nargin < 0)
   error('Not enough input arguments.');
end

% Determine whether kijg is a matrix, a string expression,
% the name of a function or an inline object.
[atype,afun,afcnstr] = iterchk(kijg);
if isequal(atype,'matrix')
   % Check matrix and right hand side vector inputs have appropriate
   % sizes
   [m,n] = size(kijg);
   if (m ~= n)
      error('Matrix must be square.');
   end
```

```
   if ~isequal(size(b),[m,1])
      es = sprintf(['Right hand side must be a column vector of' ...
            ' length %d to match the coefficient matrix.'],m);
      error(es);
   end
else
   m = size(b,1);
   n = m;
   if (size(b,2) ~= 1)
      error('Right hand side must be a column vector.');
   end
end

% Assign default values to unspecified parameters
%Adaptacion al programa
if nargin==3
    restarted=1;
else
    if (nargin < 3) | isempty(restart) | (restart == n)
        restarted = 0;
    else
        restarted = 1;
    end
    if (nargin < 4) | isempty(tol)
        tol = 1e-6;
    end
end
%sigue normal el programa
if restarted
    outer = maxit;
    inner = restart;
else
    outer = 1;
    inner = maxit;
end

% Check for all zero right hand side vector => all zero solution
n2b = norm(b);                 % Norm of rhs vector, b
if (n2b == 0)                  % if    rhs vector is all zeros
   x = zeros(n,1);             % then  solution is all zeros
   flag = 0;                   % a valid solution has been obtained
   relres = 0;                 % the relative residual is actually 0/0
   iter = [0 0];               % no iterations need be performed
```

```
   resvec = 0;                 % resvec(1) = norm(b-kijg*x) = norm(0)
   if (nargout < 2)
      itermsg('gmres',tol,maxit,0,flag,iter,NaN);
   end
   return
end

if ((nargin >= 6) & ~isempty(M1))
   existM1 = 1;
   [m1type,m1fun,m1fcnstr] = iterchk(M1);
   if isequal(m1type,'matrix')
      if ~isequal(size(M1),[m,m])
         es = sprintf(['Preconditioner must be a square matrix' ...
               ' of size %d to match the problem size.'],m);
         error(es);
      end
   end
else
   existM1 = 0;
   m1type = 'matrix';
end

if ((nargin >= 7) & ~isempty(M2))
   existM2 = 1;
   [m2type,m2fun,m2fcnstr] = iterchk(M2);
   if isequal(m2type,'matrix')
      if ~isequal(size(M2),[m,m])
         es = sprintf(['Preconditioner must be a square matrix' ...
               ' of size %d to match the problem size.'],m);
         error(es);
      end
   end
else
   existM2 = 0;
   m2type = 'matrix';
end

if ((nargin >= 8) & ~isempty(x0))
   if ~isequal(size(x0),[n,1])
      es = sprintf(['Initial guess must be a column vector of' ...
            ' length %d to match the problem size.'],n);
      error(es);
   end
```

```
else
   x0 = zeros(n,1);
end
x = x0;

if ((nargin > 8) & isequal(atype,'matrix') & ...
      isequal(m1type,'matrix') & isequal(m2type,'matrix'))
   error('Too many input arguments.');
end

% Set up for the method
flag = 1;
xmin = x;                          % Iterate which has minimal residual
                                      %  so far
imin = 0;                          % "Outer" iteration at which xmin
                                       % was computed
jmin = 0;                          % "Inner" iteration at which xmin
                                        %was computed
tolb = tol * n2b;                  % Relative tolerance
if isequal(atype,'matrix')
   r = b - kijg * x;                  % Zero-th residual
else
   r = b - iterapp(afun,atype,afcnstr,x,varargin{:});
end
normr = norm(r);                    % Norm of residual

if (normr <= tolb)                  % Initial guess is a good enough
                                        %solution
   flag = 0;
   relres = normr / n2b;
   iter = [0 0];
   resvec = normr;
   if (nargout < 2)
      itermsg('gmres',tol,maxit,[0 0],flag,iter,relres);
   end
   return
end


resvec = sparse (inner*outer+1,1,0); % Preallocate vector for norm of
                                        %residuals
resvec(1) = normr;                  % resvec(1) = norm(b-kijg*x0)
normrmin = normr;                   % Norm of residual from xmin
```

```
rho = 1;
stag = 0;                          % stagnation of the method

% loop over "outer" iterations (unless convergence or failure)


for i = 1 : outer
   V = sparse(n,inner+1,0);          % Arnoldi vectors
   h = sparse(inner+1,1,0);          % upper Hessenberg st kijg*V =
     % V*H ...
   QT = sparse(inner+1,inner+1,0);   % orthogonal factor st QT*H = R
   R = sparse(inner,inner,0);        % upper triangular factor st
    %H = Q*R
   f = sparse(inner,1,0);            % y = R\f => x = x0 + V*y
   W = sparse(n,inner,0);            % W = V*inv(R)
   j = 0;                          % "inner" iteration counter

   if existM1
      if isequal(m1type,'matrix')
         vh1 = M1 \ r;
      else
         vh1 = iterapp(m1fun,m1type,m1fcnstr,r,varargin{:});
      end
      if isinf(norm(vh1,inf))
         flag = 2;
         break
      end
   else
      vh1 = r;
   end
   if existM2
      if isequal(m2type,'matrix')
         vh = M2 \ vh1;
      else
         vh = iterapp(m2fun,m2type,m2fcnstr,vh1,varargin{:});
      end
      if isinf(norm(vh,inf))
         flag = 2;
         break
      end
   else
      vh = vh1;
   end
```

```
h(1) = norm(vh);
V(:,1) = vh / h(1);
QT(1,1) = 1;
phibar = h(1);

for j = 1 : inner
   if isequal(atype,'matrix')
      u2 = kijg * V(:,j);
   else
      u2 = iterapp(afun,atype,afcnstr,V(:,j),varargin{:});
   end
   if existM1
      if isequal(m1type,'matrix')
         u1 = M1 \ u2;
      else
         u1 = iterapp(m1fun,m1type,m1fcnstr,u2,varargin{:});
      end
      if isinf(norm(u1,inf))
         flag = 2;
         break
      end
   else
      u1 = u2;
   end
   if existM2
      if isequal(m2type,'matrix')
         u = M2 \ u1;
      else
         u = iterapp(m2fun,m2type,m2fcnstr,u1,varargin{:});
      end
      if isinf(norm(u,inf))
         flag = 2;
         break
      end
   else
      u = u1;
   end
   for k = 1 : j
      h(k) = V(:,k)' * u;
      u = u - h(k) * V(:,k);
   end
   h(j+1) = norm(u);
   V(:,j+1) = u / h(j+1);
```

```
R(1:j,j) = QT(1:j,1:j) * h(1:j);
rt = R(j,j);

% find cos(theta) and sin(theta) of Givens rotation
if h(j+1) == 0
   c = 1.0;                      % theta = 0
   s = 0.0;
elseif abs(h(j+1)) > abs(rt)
   temp = rt / h(j+1);
   s = 1.0 / sqrt(1.0 + abs(temp)^2); % pi/4 < theta < 3pi/4
   c = - temp * s;
else
   temp = h(j+1) / rt;
   c = 1.0 / sqrt(1.0 + abs(temp)^2); % -pi/4 <= theta < 0 <
                                        %theta <= pi/4
   s = - temp * c;
end
R(j,j) = c' * rt - s' * h(j+1);
%   zero = s * rt + c * h(j+1);

q = QT(j,1:j);
QT(j,1:j) = c' * q;
QT(j+1,1:j) = s * q;
QT(j,j+1) = -s';
QT(j+1,j+1) = c;
f(j) = c' * phibar;
phibar = s * phibar;

if j < inner
   if f(j) == 0                  % stagnation of the method
      stag = 1;
   end
   W(:,j) = (V(:,j) - W(:,1:j-1) * R(1:j-1,j))/ R(j,j);
   % Check for stagnation of the method
   if stag == 0
      stagtest = zeros(n,1);
      ind = (x ~= 0);
      if ~(i==1 & j==1)
         stagtest(ind) = W(ind,j) ./ x(ind);
         stagtest(~ind & W(:,j) ~= 0) = Inf;
         if abs(f(j))*norm(stagtest,inf) < eps
            stag = 1;
         end
```

```
      end
   end
   x = x + f(j) * W(:,j);       % form the new inner iterate
else % j == inner
   vrf = V(:,1:j)*(R(1:j,1:j)\f(1:j));
   % Check for stagnation of the method
   if stag == 0
      stagtest = zeros(n,1);
      ind = (x0 ~= 0);
      stagtest(ind) = vrf(ind) ./ x0(ind);
      stagtest(~ind & vrf ~= 0) = Inf;
      if norm(stagtest,inf) < eps
         stag = 1;
      end
   end
   x = x0 + vrf;                % form the new outer iterate
end

if isequal(atype,'matrix')
   normr = norm(b - kijg * x);
else
   normr = norm(b - iterapp(afun,atype,afcnstr,x,varargin{:}));
end
resvec((i-1)*inner+j+1) = normr;

if normr <= tolb               % check for convergence
   if j < inner
      y = R(1:j,1:j) \ f(1:j);
      x = x0 + V(:,1:j) * y;     % more accurate computation of
       % xj
      if isequal(atype,'matrix')
         r = b - kijg * x;
      else
         r = b - iterapp(afun,atype,afcnstr,x,varargin{:});
      end
      normr = norm(r);
      resvec((i-1)*inner+j+1) = normr;
   end
   if normr <= tolb             % check using more accurate xj
      flag = 0;
      iter = [i j];
      break
   end
```

```
      end

      if stag == 1
         flag = 3;
         break
      end

      if normr < normrmin            % update minimal norm quantities
         normrmin = normr;
         xmin = x;
         imin = i;
         jmin = j;
      end
   end                               % for j = 1 : inner

   if flag == 1
      x0 = x;                         % save for the next outer iteration
      if isequal(atype,'matrix')
         r = b - kijg * x0;
      else
         r = b - iterapp(afun,atype,afcnstr,x0,varargin{:});
      end
   else
      break
   end

end                                  % for i = 1 : maxit

%returned solution is that with minimum residual
if flag == 0
   relres = normr / n2b;
else
   x = xmin;
   iter = [imin jmin];
   relres = normrmin / n2b;
end

% truncate the zeros from resvec
if flag <= 1 | flag == 3
   resvec = resvec(1:(i-1)*inner+j+1);
else
   if j == 0
      resvec = resvec(1:(i-1)*inner+1);
```

```
   else
      resvec = resvec(1:(i-1)*inner+j);
   end
end

% only display a message if the output flag is not used
if nargout < 2
    if restarted
        itermsg(sprintf(['gmres(%d)'],restart),tol,maxit,[i j],flag,
                                        iter,relres);
    else
        itermsg(sprintf(['gmres']),tol,maxit,j,flag,iter(2),relres);
    end
end
```

Archivo: integrar

```
%Function integrar

%-------------------------------------------------------------------
%  Propósito:
% Calcular la integral numérica de volumen en coordenadas cilindricas
% transformadas a coordenadas naturales.
% Acá r=x y z=y, al colocarnos en el plano z=0.
%
%  Sintaxis:
% I=integrar(r,z,tol,it)
%
%  Descripción de variables:
% r,z vectores de las coordenadas del elemento.
% tol: tolerancia para la integracion.
% it: número máximo de iteracciones para convergencia o rompimiento del
% programa.
% error: parámetro para controlar el error en la integración
% k, m: contadores
% p y w: puntos y pesos para la cuadratura de Gauss_Legendre.
% IK: valor de la integral numérica
% n y s: puntos de Gauss.
% JDNs y JDNn: derivadas de las funciones de forma con respecto a las
%     variables naturales.
% J: Jacobiano de la transformación.
% rt y zt: coordenadas trasnformadas.
```

```
%-------------------------------------------------------------------------

%Calculo de la integral numerica integrar.m
function I=integrar(r,z,tol,it)

%Inciar error y contadores
error=tol+1; k=1; m=1; IK(1)=0;

%Inicio ciclo
while abs(error)>tol
   k=k+1;
   [p,w]=leg_pesos(k); %Llama la funciòn de Legendre para pesos y raices
   m=m+1;IK(m)=0;
   if m>=it+1 break; end
   for i=1:k
      s=p(i);
      for j=1:k
         n=p(j);
         rt=0.25*[(1-s)*(1-n) (1+s)*(1-n) (1+s)*(1+n) (1-s)*(1-n)]*r';
         JDNs=[-0.25*(1-n) 0.25*(1-n) 0.25*(1+n) -0.25*(1+n)];
         JDNn=[-0.25*(1-s) -0.25*(1+s) 0.25*(1+s) 0.25*(1-s)];
         J=[JDNs*r' JDNs*z'; JDNn*r' JDNn*z'];
         IK(m)=w(j)*w(i)*det(J)*rt+IK(m);
      end
   end
   error=IK(m)-IK(m-1);
end
I=2*pi*abs(IK(m));

clear k m i n JDNs JDNn J rt IK
```

Archivo: lee-elem

```
%lee_elem.m
%%-----------------------------------------------------------------------
%  Propósito:
% Lee la conectividad de los elementos y reordena para la
% sinterización.
%
%  Sintaxis:
% Es llamado directamente dentro del programa principal.
%
%  Descripción de variables:
```

```
% nelem: Número de elementos del sistema.
% nne: Número de nodos/elemento.
% i: Contador del número de elementos.
% n: Numeración de los nodos.
% j: Contador de nodos/elemento.
% conec: Matriz de conectividades. Dim (nelem,nne).
% conecn: vector temporal para almacenar los vectores movidos.
% cer: Contador de elementos reordenados.
% punto0: Indicador de cual es el nodo (0,0).
% idelem: Indicador del elemento que posee el punto0.
% id: contador de elementos que buscan la conectividad.
% k: contador de nodos en el elemento a reordenar.
% ce: contador de elementos buscados.
%-------------------------------------------------------------------

%Apertura archivo elementos.txt
fname=strcat(subdir,'\elementos.txt');
fid=fopen(fname,'rt');

%Leer titulo y número de elementos
nelem=fscanf(fid,'%i',1);
nne=fscanf(fid,'%i',1);
punto0=1;
tit=fscanf(fid,'%s',6);

%lee nodos de cada elemento.
for i=1:nelem
   n=fscanf(fid,'%i',1);
   fscanf(fid,'%s',4);
   for j=1:nne
      conec(n,j)=fscanf(fid,'%i',1);
   end
end

%Cierra archivo
fclose(fid);

clear tit fid i n j

%-------------------------------------------------------------------

%Reordenamiento de los nodos alrededor del elemento que posee el punto
%con coordenadas (0,0).
```

```
conecn=zeros(1,nne);

%Buscar el primer nodo con coordenadas (0,0)
cer=0;
while cer==0
   for i=1:nelem
      for j=1:nne
         if conec(i,j)==punto0
            cer=1;
            idelem=i;
         end
      end
   end
end
conecn(1,:)=conec(idelem,:);
conec(idelem,:)=conec(1,:);
conec(1,:)=conecn(1,:);

%Reordenamiento de los elementos alrededor de ese punto.
id=1; k=1; j=1; ce=2;

while cer<(nelem-1)
   while k<nne
      while ce<=nelem
         while j<=nne
            if (conec(id,k)==conec(ce,j))
               cer=cer+1;
               conecn(1,:)=conec(ce,:);
               conec(ce,:)=conec(cer,:);
               conec(cer,:)=conecn(1,:);
               j=nne+1;
               ce=cer;
               if cer==(nelem-1)
                  ce=nelem+1;
                  k=nne+1;
               end
            else
               j=j+1;
            end
         end
         ce=ce+1;
         j=1;
```

```
      end
      k=k+1;
      ce=cer+1;
   end
   k=1;
   id=id+1;
end

clear conecn cer i j idelem id k ce
```

Archivo: lee-nodos

```
%lee_nodos.m

%-------------------------------------------------------------------------
%  Propósito:
% Lee las coordenadas nodales del sistema.
%
%  Sintaxis:
% Es llamado directamente del programa principal
%
%  Descripción de variables:
% nnodos: # de nodos totales del sistema.
% nodos_iniciales: son los nodos de los vertices.
% i: contador del número de nodos.
% n: el número del nodo dentro del sistema.
% rz: Matriz de coordenadas r,z del cuerpo en crudo. Dim(n,2).
%
%-------------------------------------------------------------------------
%Apertura del archivo nodos.txt
fname=strcat(subdir,'/nodos.txt');
fid=fopen(fname,'rt');

%Lee titulo y número de nodos
nnodos=fscanf(fid,'%i',1);
tit=fscanf(fid,'%s',4);
nodos_iniciales=nnodos;

%lectura de los nodos
for i=1:nnodos
   n=fscanf(fid,'%i',1);
   rz(n,1)=fscanf(fid,'%g',1);
   rz(n,2)=fscanf(fid,'%g',1);
```

```
   tit=fscanf(fid,'%g',1);
end

rz=rz/1000; %Las coordenadas de cosmos están en [mm], hay que pasarlas
% a [m]

%Cierre de archivo
fclose(fid);

%Borrar variables locales
clear tit fname fid cont i n
```

Archivos:lee-parametros

```
%lee_parametros.m

%-------------------------------------------------------------------
%  Propósito:
% Lee los parámetros matemáticos del programa.
%
%  Sintaxis:
% Es llamado directamente en el programa principal.
%
%  Descripción de variables:
% it: Número máximo de iteraciones para convergencia de la integral
        %numérica.
% tol: tolerancia para la integración numérica.
% Bsup: Beta inicial para estimar el parámetro de contracción en la
        %sinterización.
% delta_B: Incremento en Beta para el calculo de la derivada.
% tol_B: tolerancia para la estimación de la derivada en el método
        %de N-R.
% gp: Grado polinomial a utilizar en la solución. Se puede de 1 a 8.
% NG: Número de puntos de Gauss a utilizar en la solución númerica.
% t_final: tiempo total de permanencia a la Temperatura de cocción.
        %[s]
% delta_t: ??
%       ngrest: numero de grados restringidos.
%-------------------------------------------------------------------

%Apertura del archivo densidad.txt
fname=strcat(subdir,'\parametros.txt');
fid=fopen(fname,'rt');
```

```
%Lectura del titulo
tit = fscanf(fid,'%s',1);

%Lectura de parámetros.
tol=fscanf(fid,'%g',1);
it=fscanf(fid,'%g',1);
Bsup=fscanf(fid,'%g',1);
delta_B=fscanf(fid,'%g',1);
tol_B=fscanf(fid,'%g',1);
%gp=fscanf(fid,'%g',1);
%NG=fscanf(fid,'%g',1);
t_final=fscanf(fid,'%g',1);
delta_t=fscanf(fid,'%g',1);
ngrest=fscanf(fid,'%g',1);

%Cierra archivo
fclose(fid);

clear fid tit
```

Archivo: lee-rest

```
%lee_rest.m

%-------------------------------------------------------------------------
%  Propósito:
% Lee los grados de libertad restringidos.
%
%  Sintaxis:
% Es llamado directamente en el programa principal.
%
%  Descripción de variables:
% it: Número máximo de iteraciones para convergencia de la integral
%numérica.
% tol: tolerancia para la integración numérica.
% Bsup: Beta inicial para estimar el parámetro de contracción en la
% sinterización.
% delta_B: Incremento en Beta para el calculo de la derivada.
% tol_B: tolerancia para la estimación de la derivada en el método
% de N-R.
% gp: Grado polinomial a utilizar en la solución. Se puede de 1 a 8.
% NG: Número de puntos de Gauss a utilizar en la solución númerica.
```

```
% t_final: tiempo total de permanencia a la Temperatura de cocción.
%    [s]
% delta_t: ??
%       grest: nodo restringido en z.
%-------------------------------------------------------------------

%Apertura del archivo densidad.txt
fname=strcat(subdir,'\restricciones.txt');
fid=fopen(fname,'rt');

%Lectura del titulo
tit = fscanf(fid,'%s',1);

for i=1:ngrest
   grest(i,1)=fscanf(fid,'%g',1);
   grest(i,2)=fscanf(fid,'%g',1);
end

%Cierra archivo
fclose(fid);

clear fid tit i
```

Archivo: lee-resultados

```
%lee_datos.m

%-------------------------------------------------------------------
%Este programa es independiente para leer las matrices de rigidez
%calculadas por el método de Nelson y el de Kwon
%-------------------------------------------------------------------

%Apertura archivo elementos.txt
fname=strcat('datos_matrices.txt');
fid=fopen(fname,'rt');

%Leer titulo y el elemento 1
tit=fscanf(fid,'%s',1)
n=fscanf(fid,'%i',1)

kij1=zeros(8,8);
kij2=zeros(8,8);
kij3=zeros(8,8);
```

```
kij4=zeros(8,8);
kij=zeros(20,20);
kk1=zeros(8,8);
kk2=zeros(8,8);
kk3=zeros(8,8);
kk4=zeros(8,8);
kk=zeros(20,20);


%lee nodos de cada elemento.
for i=1:8
   for j=1:8
      kij1(i,j)=fscanf(fid,'%g',1);
   end
end

n=fscanf(fid,'%i',1);

%lee nodos de cada elemento.
for i=1:8
   for j=1:8
      kij2(i,j)=fscanf(fid,'%g',1);
   end
end

n=fscanf(fid,'%i',1);
%lee nodos de cada elemento.
for i=1:8
   for j=1:8
      kij3(i,j)=fscanf(fid,'%g',1);
   end
end

n=fscanf(fid,'%i',1);
%lee nodos de cada elemento.
for i=1:8
   for j=1:8
      kij4(i,j)=fscanf(fid,'%g',1);
   end
end

%Leer titulo y el elemento 1
tit=fscanf(fid,'%s',1);
```

```
tit=fscanf(fid,'%s',1);

%lee nodos de cada elemento.
for i=1:20
   for j=1:7
      kij(i,j)=fscanf(fid,'%g',1);
   end
end

tit=fscanf(fid,'%s',1);
for i=1:20
   for j=8:14
      kij(i,j)=fscanf(fid,'%g',1);
   end
end

tit=fscanf(fid,'%s',1);
for i=1:20
   for j=15:20
      kij(i,j)=fscanf(fid,'%g',1);
   end
end

%TErmina de leer los datos de Nelson

%Leer titulo y el elemento 1
tit=fscanf(fid,'%s',1);
n=fscanf(fid,'%i',1);

%lee nodos de cada elemento.
for i=1:8
   for j=1:8
      kk1(i,j)=fscanf(fid,'%g',1);
   end
end

n=fscanf(fid,'%i',1);

%lee nodos de cada elemento.
for i=1:8
   for j=1:8
      kk2(i,j)=fscanf(fid,'%g',1);
   end
```

```
end

n=fscanf(fid,'%i',1);
%lee nodos de cada elemento.
for i=1:8
   for j=1:8
      kk3(i,j)=fscanf(fid,'%g',1);
   end
end

n=fscanf(fid,'%i',1);
%lee nodos de cada elemento.
for i=1:8
   for j=1:8
      kk4(i,j)=fscanf(fid,'%g',1);
   end
end

%Leer titulo y el elemento 1
tit=fscanf(fid,'%s',1);
tit=fscanf(fid,'%s',1);

%lee nodos de cada elemento.
for i=1:20
   for j=1:7
      kk(i,j)=fscanf(fid,'%g',1);
   end
end

tit=fscanf(fid,'%s',1);
for i=1:20
   for j=8:14
      kk(i,j)=fscanf(fid,'%g',1);
   end
end

tit=fscanf(fid,'%s',1);
for i=1:20
   for j=15:20
      kk(i,j)=fscanf(fid,'%g',1);
   end
end
```

```
%Cierra archivo
fclose(fid);
```

Archivo: leg-pesos

```
%Leg_pesos.m

%-------------------------------------------------------------------
%  Propósito:
% Programa para calcular raíces y pesos, Mediante los polinomios
% de Legendre.
%
%  Sintaxis:
% Sintaxis [p,w]=leg_pesos(n) para integrar en coordenadas
% naturales...
% n:# nodos. se debe especificar el argumento [p,w], que es el vector
% respuesta.
%
%  Descripción de variables:
% p, w: Puntos y pesos de Gauss Legendre
% n: Número de puntos a calcular
% L: Raices del polinomio de Legendre
% j: contador
%-------------------------------------------------------------------
function [p,w]=leg_pesos(n)
L=Legen_p(n);
p=sort(roots(L))';
for j=1:n
   y=zeros(1,n); y(j)=1;
   po=polyfit(p,y,n-1);
   Po=poly_itg(po);
   w(j)=polyval(Po,1)-polyval(Po,-1);
end

clear j
```

Archivo: Legen-p

```
%Legend_p.m

%-------------------------------------------------------------------
%  Propósito:
```

```
% Encontrar los coeficientes del polinomio de Legendre.
%
%  Sintaxis:
% po=Legen_p(n).
%
%  Descripción de variables:
% n: Número de puntos de Gauss
%--------------------------------------------------------------------

function po = Legen_p(n)

%y si n<0????

pbb=[1]; if n==0,, po=pbb; break; end
pb=[1 0]; if n==1, po=pb; break; end
for i=2:n
   po=((2*i-1)*[pb,0]-(i-1)*[0, 0, pbb])/i;
   pbb=pb; pb=po;
end
```

Archivo: m-crudo

```
m_crudo%m_crudo.m

%--------------------------------------------------------------------
%  Propósito:
% Calcula los elementos másicos, los cuales son anillos generados por
% de área de un cuadrilatero, y los pesos de cada elemento para el
% vector de fuerza Fz.
%
%  Sintaxis:
% Es llamado directamente en el programa principal.
%
%  Descripción de variables:
% e: Contador del número de elementos del sistema.
% l: Contador del número de nodos/elemento.
% rz: Matriz de coordenadas de la pieza en crudo. Dim(nelem,2).
% r y z: vectore de las coordendas globales de cada nodo. Dim(nne).
% I: Valor de la integral numérica.
% mc: Vector de elementos másicos [kg]. Dim(nelem).
% peso: Vector de fuerzas de peso [N]. Dim(nelem).
%--------------------------------------------------------------------
```

```
%Inicio del ciclo
for e=1:nelem
   for l=1:nne
      r(l)=rz(conec(e,l),1);
      z(l)=rz(conec(e,l),2);
   end
   I=integrar(r,z,tol,it);
   mc(e)=ig*dc*I;   %mc: masa del elemento en crudo.
end

%Calculo del peso de los elementos
peso=-9.8*mc; %Gravedad en m/s^2

%Abrir archivo temporal "peso" para almacenar el peso.
fid=fopen('peso','w');
fwrite(fid,peso,'real*8');

%cerrar archivo temporal
fclose(fid);

clear fid e l r z I
```

Archivo: malla1

```
%malla1.m

%-------------------------------------------------------------------
%  Propósito:
% Calcula los nodos faltantes para un grado polinomial p=1.
%
%  Sintaxis:
% Es llamado directamente dentro del programa mecanico.m.
%
%  Descripción de variables:
%       nelem: Cantidad de elementos
%       Ind: Matriz indicadora para insertar los nodos.
% Dim(Parejas_de_nodosx3)
%       ci, i, k, v: contadores
% nggl: Matriz numeración global de los grados de libertad. Dim(n,2).
%       glt: grados de libertad totales del sistema.
%-------------------------------------------------------------------

%Numeración global de los grados de libertad por variable nggl.
```

```
%La numeración se hace primero las r y despues las z.

for i=1:nnodos
    nggl(i,1)=i;
    nggl(i,2)=i+nnodos;
end

%Cálculo de los grados de libertad totales glt
glt=2*nnodos;

clear i
```

Archivo: malla2

```
%malla2.m

%-------------------------------------------------------------------
%  Propósito:
% Calcula los nodos faltantes para un grado polinomial p=2.
%
%  Sintaxis:
% Es llamado directamente dentro del programa mecanico.m.
%
%  Descripción de variables:
%       nelem: Cantidad de elementos
%       Ind: Matriz indicadora para insertar los nodos.
% Dim(Parejas_de_nodosx3)
%       ci, i, k, v: contadores
% nggl: Matriz numeración global de los grados de libertad. Dim(n,2).
%       glt: grados de libertad totales del sistema.
%-------------------------------------------------------------------

%Creacion de un indicador para saber si se ha creado un nuevo nodo
ci=1;
for e=1:nelem
    for i=1:nne
        Ind(ci,1)=conecg(e,i);
        Ind(ci,2)=conecg(e,i+1);
        Ind(ci,3)=0;
        ci=ci+1;
    end
end
```

```
%Nodos para el elemento 1.
e=1;
for i=1:nne
    nnodos=nnodos+1;
    Ind(i,3)=Ind(i,3)+nnodos;
%    rzs(nnodos,1)=(rzs(Ind(i,1),1)+rzs(Ind(i,2),1))/2;
%    rzs(nnodos,2)=(rzs(Ind(i,1),2)+rzs(Ind(i,2),2))/2;
end

%Nodos para los demas elementos.

k=5; nnodos=nnodos+1;
while k<=nne*nelem
    for v=1:k-1
        if (and((or((Ind(k,1)==Ind(v,1)),(Ind(k,2)==Ind(v,1))))...
                ,(or((Ind(k,1)==Ind(v,2)),(Ind(k,2)==Ind(v,2))))))
            Ind(k,3)=Ind(v,3);
            v=k;
        end
    end

    if (Ind(k,3)==0)
        Ind(k,3)=Ind(k,3)+nnodos;

 %       rzs(nnodos,1)=(rzs(Ind(k,1),1)+rzs(Ind(k,2),1))/2;
 %       rzs(nnodos,2)=(rzs(Ind(k,1),2)+rzs(Ind(k,2),2))/2;
        nnodos=nnodos+1;
    end
    k=k+1;
end
nnodos=nnodos-1;

%Numeración global de los grados de libertad por variable nggl.
%La numeración se hace primero las r y despues las z.

for i=1:nnodos
    nggl(i,1)=i;
    nggl(i,2)=i+nnodos;
end

%Cálculo de los grados de libertad totales glt
glt=2*nnodos;
```

```
%Ampliacion de la matriz de conectividades
i=1;
for e=1:nelem
    conec(e,5)=Ind(i,3);
    i=i+1;
    conec(e,6)=Ind(i,3);
    i=i+1;
    conec(e,7)=Ind(i,3);
    i=i+1;
    conec(e,8)=Ind(i,3);
    i=i+1;
end

%Recalculo de los nodos por elemento
nne=nne+4;

clear k v i ci Ind e
```

Archivo: malla3

```
%malla3.m

%-------------------------------------------------------------------
%  Propósito:
% Calcula los nodos faltantes para un grado polinomial p=2.
%
%  Sintaxis:
% Es llamado directamente dentro del programa mecanico.m.
%
%  Descripción de variables:
%       nelem: Cantidad de elementos
%       Ind: Matriz indicadora para insertar los nodos.
% Dim(Parejas_de_nodosx3)
%       ci, i, k, v: contadores
% nggl: Matriz numeración global de los grados de libertad. Dim(n,2).
%       glt: grados de libertad totales del sistema.
%-------------------------------------------------------------------

%Creacion de un indicador para saber si se ha creado un nuevo nodo
ci=1;
for e=1:nelem
    for i=1:nne
        Ind(ci,1)=conecg(e,i);
```

```
        Ind(ci,2)=conecg(e,i+1);
        Ind(ci,3)=0;
        Ind(ci,4)=0;
        ci=ci+1;
    end
end

%Nodos para el elemento 1.
e=1;
for i=1:nne
    nnodos=nnodos+1;
    Ind(i,3)=Ind(i,3)+nnodos;
    nnodos=nnodos+1;
    Ind(i,4)=Ind(i,4)+nnodos;
end

%Nodos para los demas elementos.

k=5; nnodos=nnodos+1;
while k<=nne*nelem
    for v=1:k-1
        if (and((or((Ind(k,1)==Ind(v,1)),(Ind(k,2)==Ind(v,1))))...
                ,(or((Ind(k,1)==Ind(v,2)),(Ind(k,2)==Ind(v,2))))))
            Ind(k,3)=Ind(v,3);
            Ind(k,4)=Ind(v,4);
            v=k;
        end
    end

    if (Ind(k,3)==0)
        Ind(k,3)=Ind(k,3)+nnodos;
        nnodos=nnodos+1;
        Ind(k,4)=Ind(k,4)+nnodos;
        nnodos=nnodos+1;
    end
    k=k+1;
end

nnodos=nnodos-1;

%Numeración global de los grados de libertad por variable nggl.
%La numeración se hace primero las r y despues las z.
```

```
for i=1:nnodos
    nggl(i,1)=i;
    nggl(i,2)=i+nnodos;
end

%Cálculo de los grados de libertad totales glt
glt=2*nnodos;

%Ampliacion de la matriz de conectividades
i=1;
for e=1:nelem
    conec(e,5)=Ind(i,3);
    conec(e,6)=Ind(i,4);
    i=i+1;
    conec(e,7)=Ind(i,3);
    conec(e,8)=Ind(i,4);
    i=i+1;
    conec(e,9)=Ind(i,3);
    conec(e,10)=Ind(i,4);
    i=i+1;
    conec(e,11)=Ind(i,3);
    conec(e,12)=Ind(i,4);
    i=i+1;
end

%Recalculo de los nodos por elemento
nne=N;

clear k v i ci e Ind
```

Archivo: malla4

```
%malla4.m

%-------------------------------------------------------------------
%  Propósito:
% Calcula los nodos faltantes para un grado polinomial p=4.
%
%  Sintaxis:
% Es llamado directamente dentro del programa mecanico.m.
%
%  Descripción de variables:
%       nelem: Cantidad de elementos
```

```
%       Ind: Matriz indicadora para insertar los nodos.
%   Dim(Parejas_de_nodosx3)
%       ci, i, k, v: contadores
% nggl: Matriz numeración global de los grados de libertad. Dim(n,2).
%       glt: grados de libertad totales del sistema.
%--------------------------------------------------------------------

%Creacion de un indicador para saber si se ha creado un nuevo nodo y
%de los contadores para calcular las coordenadas del nodo interno.
ci=1;
for e=1:nelem
    for i=1:nne
        Ind(ci,1)=conecg(e,i);
        Ind(ci,2)=conecg(e,i+1);
        Ind(ci,3)=0;
        Ind(ci,4)=0;
        Ind(ci,5)=0;
        ci=ci+1;
    end
end

%Nodos para el elemento 1.
e=1;
for i=1:nne
    nnodos=nnodos+1;
    Ind(i,3)=Ind(i,3)+nnodos;
    nnodos=nnodos+1;
    Ind(i,4)=Ind(i,4)+nnodos;
    nnodos=nnodos+1;
    Ind(i,5)=Ind(i,5)+nnodos;
end

%Nodos para los demas elementos.

k=5; nnodos=nnodos+1;
while k<=nne*nelem
    for v=1:k-1
        if (and((or((Ind(k,1)==Ind(v,1)),(Ind(k,2)==Ind(v,1))))...
                ,(or((Ind(k,1)==Ind(v,2)),(Ind(k,2)==Ind(v,2))))))
            Ind(k,3)=Ind(v,3);
            Ind(k,4)=Ind(v,4);
            Ind(k,5)=Ind(v,5);
            v=k;
```

```
        end
    end

    if (Ind(k,3)==0)
        Ind(k,3)=Ind(k,3)+nnodos;
        nnodos=nnodos+1;
        Ind(k,4)=Ind(k,4)+nnodos;
        nnodos=nnodos+1;
        Ind(k,5)=Ind(k,5)+nnodos;
        nnodos=nnodos+1;
    end
    k=k+1;
end

%Nodos interiores
ini=nnodos; %indicador de donde inicia el conteo de los nodos internos.

%Ampliacion de la matriz de conectividades
i=1;
for e=1:nelem
    conec(e,5)=Ind(i,3);
    conec(e,6)=Ind(i,4);
    conec(e,7)=Ind(i,5);
    i=i+1;
    conec(e,8)=Ind(i,3);
    conec(e,9)=Ind(i,4);
    conec(e,10)=Ind(i,5);
    i=i+1;
    conec(e,11)=Ind(i,3);
    conec(e,12)=Ind(i,4);
    conec(e,13)=Ind(i,5);
    i=i+1;
    conec(e,14)=Ind(i,3);
    conec(e,15)=Ind(i,4);
    conec(e,16)=Ind(i,5);
    i=i+1;
    conec(e,17)=ini;
    ini=ini+1;
end

%Recalculo de los nodos por elemento
nne=N;
```

```
%Numeración global de los grados de libertad por variable nggl.
%La numeración se hace primero las r y despues las z.

nnodos=nnodos+nelem-1;

for i=1:nnodos
    nggl(i,1)=i;
    nggl(i,2)=i+nnodos;
end

%Cálculo de los grados de libertad totales glt
glt=2*nnodos;


clear k v i ci ini e Ind
```

Archivo: malla5

```
%malla5.m

%-------------------------------------------------------------------
%  Propósito:
% Calcula los nodos faltantes para un grado polinomial p=2.
%
%  Sintaxis:
% Es llamado directamente dentro del programa mecanico.m.
%
%  Descripción de variables:
%       nelem: Cantidad de elementos
%       Ind: Matriz indicadora para insertar los nodos.
% Dim(Parejas_de_nodosx3)
%       ci, i, k, v: contadores
% nggl: Matriz numeración global de los grados de libertad. Dim(n,2).
%       glt: grados de libertad totales del sistema.
%-------------------------------------------------------------------

%Creacion de un indicador para saber si se ha creado un nuevo nodo y
%de los contadores para calcular las coordenadas del nodo interno.
ci=1;
for e=1:nelem
    for i=1:nne
        Ind(ci,1)=conecg(e,i);
        Ind(ci,2)=conecg(e,i+1);
```

```
        Ind(ci,3)=0;
        Ind(ci,4)=0;
        Ind(ci,5)=0;
        Ind(ci,6)=0;
        ci=ci+1;
    end
end

%Nodos para el elemento 1.
e=1;
for i=1:nne
    nnodos=nnodos+1;
    Ind(i,3)=Ind(i,3)+nnodos;
    nnodos=nnodos+1;
    Ind(i,4)=Ind(i,4)+nnodos;
    nnodos=nnodos+1;
    Ind(i,5)=Ind(i,5)+nnodos;
    nnodos=nnodos+1;
    Ind(i,6)=Ind(i,6)+nnodos;
end

%Nodos para los demas elementos.

k=5; nnodos=nnodos+1;
while k<=nne*nelem
    for v=1:k-1
        if (and((or((Ind(k,1)==Ind(v,1)),(Ind(k,2)==Ind(v,1))))...
                ,(or((Ind(k,1)==Ind(v,2)),(Ind(k,2)==Ind(v,2))))))
            Ind(k,3)=Ind(v,3);
            Ind(k,4)=Ind(v,4);
            Ind(k,5)=Ind(v,5);
            Ind(k,6)=Ind(v,6);
            v=k;
        end
    end

    if (Ind(k,3)==0)
        Ind(k,3)=Ind(k,3)+nnodos;
        nnodos=nnodos+1;
        Ind(k,4)=Ind(k,4)+nnodos;
        nnodos=nnodos+1;
        Ind(k,5)=Ind(k,5)+nnodos;
        nnodos=nnodos+1;
```

```
        Ind(k,6)=Ind(k,6)+nnodos;
        nnodos=nnodos+1;
    end
    k=k+1;
end

%Nodos interiores
ini=nnodos; %indicador de donde inicia el conteo de los nodos internos.

%Ampliacion de la matriz de conectividades
i=1;
for e=1:nelem
    conec(e,5)=Ind(i,3);
    conec(e,6)=Ind(i,4);
    conec(e,7)=Ind(i,5);
    conec(e,8)=Ind(i,6);
    i=i+1;
    conec(e,9)=Ind(i,3);
    conec(e,10)=Ind(i,4);
    conec(e,11)=Ind(i,5);
    conec(e,12)=Ind(i,6);
    i=i+1;
    conec(e,13)=Ind(i,3);
    conec(e,14)=Ind(i,4);
    conec(e,15)=Ind(i,5);
    conec(e,16)=Ind(i,6);
    i=i+1;
    conec(e,17)=Ind(i,3);
    conec(e,18)=Ind(i,4);
    conec(e,19)=Ind(i,5);
    conec(e,20)=Ind(i,6);
    i=i+1;
    %Conteo nodos internos.
    conec(e,21)=ini;
    ini=ini+1;
    conec(e,22)=ini;
    ini=ini+1;
    conec(e,23)=ini;
    ini=ini+1;
end

%Recalculo de los nodos por elemento
nne=N;
```

```
%Numeración global de los grados de libertad por variable nggl.
%La numeración se hace primero las r y despues las z.

nnodos=nnodos+3*nelem-1; %3 se corresponde con el # de nodos internos.

for i=1:nnodos
    nggl(i,1)=i;
    nggl(i,2)=i+nnodos;
end

%Cálculo de los grados de libertad totales glt
glt=2*nnodos;

clear k v i ci ini
```

Archivo: malla6

```
%malla6.m

%-------------------------------------------------------------------
%  Propósito:
% Calcula los nodos faltantes para un grado polinomial p=2.
%
%  Sintaxis:
% Es llamado directamente dentro del programa mecanico.m.
%
%  Descripción de variables:
%       nelem: Cantidad de elementos
%       Ind: Matriz indicadora para insertar los nodos.
%  Dim(Parejas_de_nodosx3)
%       ci, i, k, v: contadores
% nggl: Matriz numeración global de los grados de libertad. Dim(n,2).
%       glt: grados de libertad totales del sistema.
%-------------------------------------------------------------------

%Creacion de un indicador para saber si se ha creado un nuevo nodo y
%de los contadores para calcular las coordenadas del nodo interno.
ci=1;
for e=1:nelem
    for i=1:nne
        Ind(ci,1)=conecg(e,i);
        Ind(ci,2)=conecg(e,i+1);
```

```
        Ind(ci,3)=0;
        Ind(ci,4)=0;
        Ind(ci,5)=0;
        Ind(ci,6)=0;
        Ind(ci,7)=0;
        ci=ci+1;
    end
end

%Nodos para el elemento 1.
e=1;
for i=1:nne
    nnodos=nnodos+1;
    Ind(i,3)=Ind(i,3)+nnodos;
    nnodos=nnodos+1;
    Ind(i,4)=Ind(i,4)+nnodos;
    nnodos=nnodos+1;
    Ind(i,5)=Ind(i,5)+nnodos;
    nnodos=nnodos+1;
    Ind(i,6)=Ind(i,6)+nnodos;
    nnodos=nnodos+1;
    Ind(i,7)=Ind(i,7)+nnodos;
end

%Nodos para los demas elementos.

k=5; nnodos=nnodos+1;
while k<=nne*nelem
    for v=1:k-1
        if (and((or((Ind(k,1)==Ind(v,1)),(Ind(k,2)==Ind(v,1))))...
                ,(or((Ind(k,1)==Ind(v,2)),(Ind(k,2)==Ind(v,2))))))
            Ind(k,3)=Ind(v,3);
            Ind(k,4)=Ind(v,4);
            Ind(k,5)=Ind(v,5);
            Ind(k,6)=Ind(v,6);
            Ind(k,7)=Ind(v,7);
            v=k;
        end
    end

    if (Ind(k,3)==0)
        Ind(k,3)=Ind(k,3)+nnodos;
        nnodos=nnodos+1;
```

```
        Ind(k,4)=Ind(k,4)+nnodos;
        nnodos=nnodos+1;
        Ind(k,5)=Ind(k,5)+nnodos;
        nnodos=nnodos+1;
        Ind(k,6)=Ind(k,6)+nnodos;
        nnodos=nnodos+1;
        Ind(k,7)=Ind(k,7)+nnodos;
        nnodos=nnodos+1;
    end
    k=k+1;
end

%Nodos interiores
ini=nnodos; %indicador de donde inicia el conteo de los nodos internos.

%Ampliacion de la matriz de conectividades
i=1;
for e=1:nelem
    conec(e,5)=Ind(i,3);
    conec(e,6)=Ind(i,4);
    conec(e,7)=Ind(i,5);
    conec(e,8)=Ind(i,6);
    conec(e,9)=Ind(i,7);
    i=i+1;
    conec(e,10)=Ind(i,3);
    conec(e,11)=Ind(i,4);
    conec(e,12)=Ind(i,5);
    conec(e,13)=Ind(i,6);
    conec(e,14)=Ind(i,7);
    i=i+1;
    conec(e,15)=Ind(i,3);
    conec(e,16)=Ind(i,4);
    conec(e,17)=Ind(i,5);
    conec(e,18)=Ind(i,6);
    conec(e,19)=Ind(i,7);
    i=i+1;
    conec(e,20)=Ind(i,3);
    conec(e,21)=Ind(i,4);
    conec(e,22)=Ind(i,5);
    conec(e,23)=Ind(i,6);
    conec(e,24)=Ind(i,7);
    i=i+1;
    %Conteo nodos internos.
```

```
    conec(e,25)=ini;
    ini=ini+1;
    conec(e,26)=ini;
    ini=ini+1;
    conec(e,27)=ini;
    ini=ini+1;
    conec(e,28)=ini;
    ini=ini+1;
    conec(e,29)=ini;
    ini=ini+1;
    conec(e,30)=ini;
    ini=ini+1;
end

%Recalculo de los nodos por elemento
nne=N;

%Numeración global de los grados de libertad por variable nggl.
%La numeración se hace primero las r y despues las z.

nnodos=nnodos+6*nelem-1; %6 se corresponde con el # de nodos internos.

for i=1:nnodos
    nggl(i,1)=i;
    nggl(i,2)=i+nnodos;
end

%Cálculo de los grados de libertad totales glt
glt=2*nnodos;

clear k v i ci ini
```

Archivo: malla7

```
%malla7.m

%-------------------------------------------------------------------
%  Propósito:
% Calcula los nodos faltantes para un grado polinomial p=2.
%
%  Sintaxis:
% Es llamado directamente dentro del programa mecanico.m.
%
```

```
%  Descripción de variables:
%       nelem: Cantidad de elementos
%       Ind: Matriz indicadora para insertar los nodos.
% Dim(Parejas_de_nodosx3)
%       ci, i, k, v: contadores
% nggl: Matriz numeración global de los grados de libertad. Dim(n,2).
%       glt: grados de libertad totales del sistema.
%-------------------------------------------------------------------

%Creacion de un indicador para saber si se ha creado un nuevo nodo y
%de los contadores para calcular las coordenadas del nodo interno.
ci=1;
for e=1:nelem
    for i=1:nne
        Ind(ci,1)=conecg(e,i);
        Ind(ci,2)=conecg(e,i+1);
        Ind(ci,3)=0;
        Ind(ci,4)=0;
        Ind(ci,5)=0;
        Ind(ci,6)=0;
        Ind(ci,7)=0;
        Ind(ci,8)=0;
        ci=ci+1;
    end
end

%Nodos para el elemento 1.
e=1;
for i=1:nne
    nnodos=nnodos+1;
    Ind(i,3)=Ind(i,3)+nnodos;
    nnodos=nnodos+1;
    Ind(i,4)=Ind(i,4)+nnodos;
    nnodos=nnodos+1;
    Ind(i,5)=Ind(i,5)+nnodos;
    nnodos=nnodos+1;
    Ind(i,6)=Ind(i,6)+nnodos;
    nnodos=nnodos+1;
    Ind(i,7)=Ind(i,7)+nnodos;
    nnodos=nnodos+1;
    Ind(i,8)=Ind(i,8)+nnodos;
end
```

```
%Nodos para los demas elementos.

k=5; nnodos=nnodos+1;
while k<=nne*nelem
    for v=1:k-1
        if (and((or((Ind(k,1)==Ind(v,1)),(Ind(k,2)==Ind(v,1))))...
                ,(or((Ind(k,1)==Ind(v,2)),(Ind(k,2)==Ind(v,2))))))
            Ind(k,3)=Ind(v,3);
            Ind(k,4)=Ind(v,4);
            Ind(k,5)=Ind(v,5);
            Ind(k,6)=Ind(v,6);
            Ind(k,7)=Ind(v,7);
            Ind(k,8)=Ind(v,8);
            v=k;
        end
    end

    if (Ind(k,3)==0)
        Ind(k,3)=Ind(k,3)+nnodos;
        nnodos=nnodos+1;
        Ind(k,4)=Ind(k,4)+nnodos;
        nnodos=nnodos+1;
        Ind(k,5)=Ind(k,5)+nnodos;
        nnodos=nnodos+1;
        Ind(k,6)=Ind(k,6)+nnodos;
        nnodos=nnodos+1;
        Ind(k,7)=Ind(k,7)+nnodos;
        nnodos=nnodos+1;
        Ind(k,8)=Ind(k,8)+nnodos;
        nnodos=nnodos+1;
    end
    k=k+1;
end

%Nodos interiores
ini=nnodos; %indicador de donde inicia el conteo de los nodos internos.

%Ampliacion de la matriz de conectividades
i=1;
for e=1:nelem
    conec(e,5)=Ind(i,3);
    conec(e,6)=Ind(i,4);
    conec(e,7)=Ind(i,5);
```

```
conec(e,8)=Ind(i,6);
conec(e,9)=Ind(i,7);
conec(e,10)=Ind(i,8);
i=i+1;
conec(e,11)=Ind(i,3);
conec(e,12)=Ind(i,4);
conec(e,13)=Ind(i,5);
conec(e,14)=Ind(i,6);
conec(e,15)=Ind(i,7);
conec(e,16)=Ind(i,8);
i=i+1;
conec(e,17)=Ind(i,3);
conec(e,18)=Ind(i,4);
conec(e,19)=Ind(i,5);
conec(e,20)=Ind(i,6);
conec(e,21)=Ind(i,7);
conec(e,22)=Ind(i,8);
i=i+1;
conec(e,23)=Ind(i,3);
conec(e,24)=Ind(i,4);
conec(e,25)=Ind(i,5);
conec(e,26)=Ind(i,6);
conec(e,27)=Ind(i,7);
conec(e,28)=Ind(i,8);
i=i+1;
%Conteo nodos internos.
conec(e,29)=ini;
ini=ini+1;
conec(e,30)=ini;
ini=ini+1;
conec(e,31)=ini;
ini=ini+1;
conec(e,32)=ini;
ini=ini+1;
conec(e,33)=ini;
ini=ini+1;
conec(e,34)=ini;
ini=ini+1;
conec(e,35)=ini;
ini=ini+1;
conec(e,36)=ini;
ini=ini+1;
conec(e,37)=ini;
```

```
    ini=ini+1;
    conec(e,38)=ini;
    ini=ini+1;
end

%Recalculo de los nodos por elemento
nne=N;

%Numeración global de los grados de libertad por variable nggl.
%La numeración se hace primero las r y despues las z.

nnodos=nnodos+10*nelem-1; %10 se corresponde con el # de nodos
internos.

for i=1:nnodos
    nggl(i,1)=i;
    nggl(i,2)=i+nnodos;
end

%Cálculo de los grados de libertad totales glt
glt=2*nnodos;

clear k v i ci ini
```

Archivo: mallas

```
%mallas.m

%-------------------------------------------------------------------
%  Propósito:
% Asigna los nodos laterales e internos segun el grado polinomial.
%       trabaja hasta p=8
%
%  Sintaxis:
% Es llamado dentro del programa principal
%
%  Descripción de variables:
% gp: Grado polinomial.
%
%-------------------------------------------------------------------
%Calculo del # de funciones de forma para la transformacion de función
NN=4;  %Número de funciones de forma de los nodos
if gp==1
```

```
   NL=0;
   Ni=0;
else
   NL=4*(gp-1);   %Número de funciones de forma de los lados
   Ni=(gp-2)*(gp-3)/2;  %Número de funciones de forma interna
end
N=NN+NL+Ni;  %Número total de funciones de forma

%Creacion de un vector complemento necesario para otros programas.
conecg=[conec conec(:,1)];

%Decision de que subprograma llamar dependiendo del grado polinomial
% a usar

if gp==1
    malla1
else
    if gp==2
        malla2
    else
        if gp==3
            malla3
        else
            if gp==4
                malla4
            else
                if gp==5
                    malla5
                else
                    if gp==6
                        malla6
                    else
                        if gp==7
                            malla7
                        else
                            if gp==8
                                malla8
                            else
                                break
                            end
                        end
                    end
                end
```

```
            end
        end
    end
end
```

Archivo: mallas

Archivo: mallas

```
%mallas.m

%-----------------------------------------------------------------------
%  Propósito:
% Asigna los nodos laterales e internos segun el grado polinomial.
%       trabaja hasta p=8
%
%  Sintaxis:
% Es llamado dentro del programa principal
%
%  Descripción de variables:
% gp: Grado polinomial.
%
%-----------------------------------------------------------------------
%Calculo del # de funciones de forma para la transformacion de función
NN=4;  %Número de funciones de forma de los nodos
if gp==1
   NL=0;
   Ni=0;
else
   NL=4*(gp-1);   %Número de funciones de forma de los lados
   Ni=(gp-2)*(gp-3)/2;  %Número de funciones de forma interna
end
N=NN+NL+Ni;  %Número total de funciones de forma

%Creacion de un vector complemento necesario para otros programas.
conecg=[conec conec(:,1)];

%Decision de que subprograma llamar dependiendo del grado polinomial
% a usar
```

```
if gp==1
    malla1
else
    if gp==2
        malla2
    else
        if gp==3
            malla3
        else
            if gp==4
                malla4
            else
                if gp==5
                    malla5
                else
                    if gp==6
                        malla6
                    else
                        if gp==7
                            malla7
                        else
                            if gp==8
                                malla8
                            else
                                break
                            end
                        end
                    end
                end
            end
        end
    end
end
```

Archivo: Matriz-kij

```
%Matriz_kij.m   version Oct 18/04

%-------------------------------------------------------------------
%  Propósito:
% Calcula las matrices de rigidez kij de los elementos y ensambla
% la matriz global de rigidez.
```

```
%       Adicionalmente guarda los calculos de los Jacobiana para ser
%       usados por otros subprogramas.
%
%  Sintaxis:
% Es llamado directamente dentro del programa mecánico.m.
%
%  Descripción de variables:
% rt, zt: valores de r,z en los puntos de Gauss.
% intr,intz: contadores de los puntos de Gauss.
%       kijg: Matriz de rigidez global. Dim(glt,glt).
%       e: Contador del numero de elementos.
%       l: Contador del numero de nodos por elementos
%       kij: Matrices de rigidez de cada elemento. Dim(2*N,2*N).
%       CNG: Contador del numero de puntos de Gauss por elemento.
%       intr, intz: Contadores individuales de puntos de Gauss.
%       DJ: Determinante de la matriz de trasnformacion.
% i: Contador del numero de nodos por elemento.
%       cn: Contador del numero de nodos por elemento.
%       c: Indicador de la conectividad.
% r,z: Coordenadas de los nodos.
%       J: Matriz Jacobiana.
% Jinv: Inverso del Jacobiano.
%       p,w: Puntos y pesos de Legendre.
% mtxcin: Matriz cinemática. Dim(4,2*N).
%-------------------------------------------------------------------

%Abrir archivo temporales para almecenar el |J| y las matrices inversas
%de la transformacion de coordenadas.
fid1=fopen('DJ','w');
fid2=fopen('Jinv','w');
fid3=fopen('kij','w');

%Inicializa matrices de los radios y zetas evaluados en los puntos de
% Gauss.
rt=zeros(NG*NG,nelem);
zt=zeros(NG*NG,nelem);

%Inicializa la matriz de rigidez global en forma sparse.
kijg=sparse(1:glt,1:glt,0);

%Calculo de las matrices kij de cada elemento.
for e=1:nelem
   %Extraer coordenadas del elemento en los vertices.
```

```
   for l=1:4
      r(l,e)=rzs(conec(e,l),1);
      z(l,e)=rzs(conec(e,l),2);
   end
   %Calcula los r, z en puntos de Gauss.
   rt(:,e)=rt(:,e)+NIt*r(:,e);
   zt(:,e)=zt(:,e)+NIt*z(:,e);
   %inicializa la matriz de los kij de los elementos.
   kij=zeros(2*N);
   CNG=0;
   for intr=1:NG
      s=p(intr);
      for intz=1:NG
          n=p(intz);
          CNG=CNG+1;
          J=[DNst(CNG,:)*r(:,e) DNst(CNG,:)*z(:,e); DNnt(CNG,:)*r(:,e)
              DNnt(CNG,:)*z(:,e)];
          DJ=det(J);
          fwrite(fid1,DJ,'real*8'); %Guarda el determinante de la
          % matriz
          Jacobiana.
          Jinv=inv(J);
          fwrite(fid2,Jinv,'real*8'); %Guarda la matriz J inversa.
          [mtxcin]=f_matriz_cinematica(N,rt(CNG,e),Jinv,CNG);
          kij=kij+w(intr)*w(intz)*mtxcin'*D*mtxcin*DJ*rt(CNG,e);
      end
   end
   %Guarda las matrices de rigidez
   fwrite(fid3,kij,'real*8');
   %Ensamble de la matriz global
   for cn=1:nne
      c=conec(e,cn); %Extrae numeración global del nodo
      ngle(cn)=nggl(c,1); %Extrae numeración global de grados de
      libertad
      ngle(cn+nne)=nggl(c,2); %Extrae numeración global de grados de
      libertad
   end
   for i=1:2*N
       for j=1:2*N
           kijg(ngle(i),ngle(j))=kijg(ngle(i),ngle(j))+kij(i,j);
       end
   end
end
```

```
%cerrar archivo temporal
fclose(fid1);
fclose(fid2);

%Elimina variables locales
clear fid1 fid2 e r CNG J DJ Jinv kij cn c i j s n
```

Archivo: Mecanico-theta

```
%mecanico_theta.m  Version abril  - 2005

%-------------------------------------------------------------------------
%  Propósito:
% Calcula el comportamiento mecánico del plato durante el proceso de
% coccion, bajo el modelo elástico y tipo "creep", y resolución con
% el método implicito.
%
%  Sintaxis:
% Es llamado directamente dentro del programa principal.
%
%  Descripción de variables:
% a: Vector de desplazamiento nodales. Dim(nggl).
% conit: Contador de iteracciones.
% D: Matriz constitutiva del material. Dim(4,4).
% delta_t: Incremento de tiempo. [s]
% delta_Ec: Incremento de la deformación creep. Dim((4,nelem).
% Ec: Matríz de deformaciones por el comportamiento creep. Dim(4,nelem).
% error_tension: Error entre la tension elástica y la creep.
%       Eu: matriz de deformaciones elásticas. Dim(4,nelem).
% fe: Vector de fuerzas de gravedad. Dim(nelem).
% t: Tiempo inicial.
% t_final: Tiempo final de la modelación.
% tension: Tensión originada por el comportamiento elástico. Dim((4,nelem).
% tension_Ec: Tensión ocasionado por el componente Creep. Dim((4,nelem).
% Vel_Ec: Velocidad de deformación creep en un tiempo t. Dim((4,nelem).
%-------------------------------------------------------------------------

%Calcular la matriz D de propiedades del material
D=E/(1+u)/(1-2*u)*[1-u u u 0; u 1-u u 0; u u 1-u 0; 0 0 0 (1-2*u)/2];

%Calcula los puntos y pesos de Gauss a ser utilizados y los almacena en los vectores
```

```
[p,w]=leg_pesos(NG);

%Calculo de las N y N' en los puntos de Gauss
[NI,DNs,DNn]=f_form(gp,N,p,NG);

%Extraccion de las funciones de forma y sus derivadas de los vertices, para calcular
%la transformacion de las coordenadas.
NIt=NI(1:4,:)';
DNst=DNs(1:4,:)';
DNnt=DNn(1:4,:)';

%Aplicación de restricciones
rest=zeros(2*nnodos,1); %Crea un vector de 0 para especificar restricciones
for i=1:ngrest
    if grest(i,2)==1
        rest(grest(i,1),1)=rest(grest(i,1),1)+1; %Restringe el grado de libertad en z
    else
        rest(nnodos+grest(i,1),1)=rest(nnodos+grest(i,1),1)+1;
    end
end

%Calculo de la matriz de rigidez del sistema.
fprintf('   Cálculo de la matriz de rigidez y del vector de carga\n')
Matriz_kij

%Calculo del vector de fuerza global
vector_qa

%Primera solución elástica del sistema
fprintf('   Solución Elástica del Sistema\n')

[ae,flag,coef_pena,residuale,iteraccionese,normae]=sol_gmres(Fzg,nnodos,rest,glt);

sprintf('\n  Flag:  %6i',flag)

%Calculo de la norma de energía en la solución elástica
bilineal
NEE=sqrt(0.5*BUU);

%Calculo de reacciones
for i=1:ngrest
     if grest(i,2)==1
         R(i)=coef_pena*ae(grest(i,2));
```

```
     else
         R(i)=coef_pena*ae(grest(i,2)+nnodos);
    end
end
R

%Calculo de las deformaciones y tensiones en el tiempo t=0.
[Eu,tension]=Def_tens(nelem,D,conec,ae,gp,nne,N,NG);

%Actualizacion coordenadas por el comportamiento elastico
ra=[ae(1:nodos_iniciales,1) ae(nnodos+1:nnodos+nodos_iniciales,1)];
rzsd=rzs+ra;
rzsinter=rzs; %Conservacion de las coordenadas sinterizadas

theta=.6; %constante para el método theta.

%Inicializar el vector de desplazamiento creep y el primer desplazamiento.
ac=zeros(glt,1);
a0=zeros(glt,1);

%Inicio de la iteracción para el comportamiento Creep.
t=0;
iteraccion_creep=0;
fid5=fopen('rt','r');
despla0=zeros(t_final/delta_t+1,1);
tensioneff0=zeros(t_final/delta_t+1,1);
velec0=zeros(t_final/delta_t+1,1);
residualcreep=zeros(t_final/delta_t+1,1);
contador=0;
while t<=t_final
   contador=contador+1; %Contador para poder almacenar datos.
   iteraccion_creep=iteraccion_creep+1;
   disp(sprintf('\nVoy en la iteraccion:  %6i',iteraccion_creep))

   [Vel_Ec,tension_eff]=vel_creep(nelem,A,AA,tension,NG);
   delta_Ec=Vel_Ec*delta_t*theta;

   %Calculo del vector de cargas asociado a creep
   vector_qc
   %Calculo de los desplazamientos creep
   [atheta,flag,coef_pena,residualc,iteraccionesc,normac]=sol_gmres(qcg,nnodos,rest,g
   a=(atheta-(1-theta)*a0)/theta;
   %Actualizacion del desplazamiento creep acumulado
```

```
ac=ac+a;

%Valores del punto cero
despla0(contador,1)=ac(nnodos+1,1);
tensioneff0(contador,1)=tension_eff(1,1);
velec0(contador,1)=Vel_Ec(3,1);
residualcreep(contador,1)=residualc;

%-------------------------------------------------------------
%Actualizacion de coordenadas
ra=[a(1:nodos_iniciales,1) a(nnodos+1:nnodos+nodos_iniciales,1)];
rzsd=rzsd+ra;
rzs=rzsd; %Se necesita reasignar esta matriz para poder calcular [K]

%Recalculo de la matriz de rigidez del sistema.
Matriz_kij

%Solución elastica del sistema.
[ae,flag,coef_pena,residualef,iteraccionesef,normaef]=sol_gmres(Fzg,nnodos,rest,gl

disp(sprintf('  Flag:  %6i',flag))

%Calculo de la tension elastica del sistema actualizado
[Eu,tension]=Def_tens(nelem,D,conec,ae,gp,nne,N,NG);

a0=a; %Nuevo punto de arranque para la iteraccion
disp(sprintf('  t = :  %6i',t))
t=t+delta_t;

%Grafico de la tension efectiva en la ultima iteraccion
if t==t_final+delta_t
    figure(2)
    subplot(2,2,1)
    hold on
    grid on
    xlabel('Radio [m]'), ylabel('Tension [Pa]')
    title ('Distribucion tension en la direccion radial')
    subplot(2,2,2)
    hold on
    grid on
    xlabel('Radio [m]'), ylabel('Tension [Pa]')
    title ('Distribucion tension en la direccion angular')
    subplot(2,2,3)
```

```
hold on
grid on
xlabel('Radio [m]'), ylabel('Tension [Pa]')
title ('Distribucion tension en la direccion axial')
subplot(2,2,4)
hold on
grid on
xlabel('Radio [m]'), ylabel('Tension eff [Pa]')
title ('Distribucion tension efectiva')
CT=0; %Contador de puntos de Gauss
for e=1:nelem
    CNG=0;
    for intr=1:NG
        s=p(intr);
        for intz=1:NG
            n=p(intz);
            CNG=CNG+1;
            CT=CT+1;
            subplot(2,2,1)
            plot (rt(CNG,e),tension(1,CT),'*b');
            subplot(2,2,2)
            plot (rt(CNG,e),tension(2,CT),'*r');
            subplot(2,2,3)
            plot (rt(CNG,e),tension(3,CT),'*g');
            subplot(2,2,4)
            plot (rt(CNG,e),tension_eff(CT),'*k');
            hold on
        end
    end
end
figure(5)
subplot(2,2,1)
hold on
grid on
xlabel('Radio [m]'), ylabel('Vel_Ec [1/s]')
title ('Distribucion Vel. Def direccion radial')
subplot(2,2,2)
hold on
grid on
xlabel('Radio [m]'), ylabel('Vel_Ec [1/s]')
title ('Distribucion Vel. Def direccion angular')
subplot(2,2,3)
hold on
```

```
       grid on
       xlabel('Radio [m]'), ylabel('Vel_Ec [1/s]')
       title ('Distribucion Vel. Def direccion  axial')
       subplot(2,2,4)
       hold on
       grid on
       xlabel('Radio [m]'), ylabel('Vel_Ec [1/s]')
       title ('Distribucion Vel. Def direccion rz')
       CT=0; %Contador de puntos de Gauss
       for e=1:nelem
           CNG=0;
           for intr=1:NG
               s=p(intr);
               for intz=1:NG
                   n=p(intz);
                   CNG=CNG+1;
                   CT=CT+1;
                   subplot(2,2,1)
                   plot (rt(CNG,e),Vel_Ec(1,CT),'*b');
                   subplot(2,2,2)
                   plot (rt(CNG,e),Vel_Ec(2,CT),'*r');
                   subplot(2,2,3)
                   plot (rt(CNG,e),Vel_Ec(3,CT),'*g');
                   subplot(2,2,4)
                   plot (rt(CNG,e),Vel_Ec(4,CT),'*k');
                   hold on
               end
           end
       end
   end
end

%Guarda las coordenadas desplazadas y los desplazamientos
save rzsd rzsd -ascii;
save desplazamientos ac -ascii;

%Cálculo de la norma de energía en la solución creep
ae=ac; %Reasigna el valor de ae para poder usar el programa bilineal.
bilineal
NEC=sqrt(0.5*BUU);

%Muestra los resultados de la iteracciones de la solucion iterada
[residuale residualc residualef];
```

```
[iteraccionese iteraccionesc iteraccionesef];
normae;
normac;
normaef;
%Grafica de la norma del error en la ultima iteraccion
figure(4)
plot (normac,'+-b');
grid on
xlabel('Numero iteracciones [#]'), ylabel('norma')
title ('Comportamiento de la norma del error creep bajo el modelo de residuos minimos
%Grafica del residual en la ultima iteraccion
```

Archivo: modelaplato

```
modelaplato%modela_plato.m  Version Oct 18/04

%%-----------------------------------------------------------------
%  Propósito:
%       Programa general para modelar el comportamiento de un plato
% cerámico.
% Incluye la sinterización y la modelación mecánica.
% El programa calcula las coordenas sinterizadas de un plato
% de n elementos cuadrilateros, como un problema axi-simétrico
% basado en el principio de conservación de la masa. La modelación
% mecánica se basa en el metod theta, y utiliza un modelo
%       "Creep deformation" basado en La ley de Norton-Bailey.
%
%  Sintaxis:
%     Se escribe directamente en la ventana de matlab el nombre del
% programa.
%
%  Descripción de variables:
% conec: Vector de conexiones entre elementos.
% conecg: Vector de conexiones entre elementos ampliado para graficar.
%       dc: Densidad en crudo del plato ceramico [kg/m3].
%       ds: Densidad del plato ceramico sinterizado [kg/m3].
%       E: Modulo de Young [Pa]
%       Fzg: Vector global de fuerzas masicas. Dim(glt,1).
%       glt: Grados de libertad totales.
%       ig: Indice de pérdidas por ignición de la pieza cerámica [%].
%       kijg: Matriz global de rigidez. Dim(glt,glt).
% nelem: # de elementos en el sistema.
%       nne: Numero de nodos por elemento.
```

```
%       NI, DNn, DNs: Funciones de forma y sus derivadas evaluados en
        %puntos de Gauss.
%                       Dim(N,NG).
%       N: Numero de funciones de forma.
%       NG: Cantidad de puntos de Gauss.
%       nggl: Matriz de numeracion de los grados de libertad totales.
        %Dim(glt,2).
%       p, w: Puntos y pesos de Legendre.
%       qcg: Vector global de cargas asociado al comportamiento creep.
        %Dim(glt,1).
% rz: Matriz de las coordenas en crudo r,z.
% rzs: matriz global de coordenadas sinterizadas.
% rzsd: matriz global de coordenadas sinterizadas y modeladas
        %mecánicamente.
% rzreal: matriz global de coordenadas de la pieza real.
%       subdir: Subdirectorio donsde están los datos del problema.
%       u: Coeficiente de Poisson <0.5.
%-------------------------------------------------------------------
%clear all
%clc

warning off MATLAB:break_outside_of_loop


%Establecimiento de las variables globales.
global kijg Fzg qcg D NI DNn DNs rz rzs rzsd rt zt nggl

%Permite entrar el nombre del subdirectorio donde están los datos
%del problema.
%subdir=input('Subdirectorio de datos: ','s');

%Lectura de las propiedades mecánicas del material
fprintf('\n\n Lectura de las densidades de la pieza cruda y
 sinterizada\n')
lee_propiedades

%Lectura de las coordenadas nodales del sistema.
fprintf('\n Lectura de las coordenadas nodales\n')
lee_nodos

%Lectura de las conectividades de los elementos.
fprintf('\n Lectura de las conectividades de los elementos\n')
lee_elem
```

```
%Lectura de los parametros matemáticos.
fprintf('\n Lectura de los parámetros matemáticos\n')
lee_parametros

%Cálculo de los elementos másicos en crudo.
fprintf('\n Cálculo de los elementos másicos crudos\n')
m_crudo

%Cálculo de las coordenadas del elemento sinterizado.
sinter

%Asignacion de nodos segun el grado polinomial.
mallas

%Lectura de restricciones.
fprintf('\n Lectura de las restricciones\n')
lee_rest

%Calculo del comportamiento mecánico.
fprintf('\n Comportamiento Mecánico\n')
mecanico_theta

%Gráfico de la pieza en crudo.
figure(3)

%Matriz necesaria para graficar solo los vertices
conecv=[conec(:,1) conec(:,2) conec(:,3) conec(:,4) conec(:,1)];

for e=1:nelem
   plot (rz(conecv(e,:),1),rz(conecv(e,:),2),'g')
   axis equal
   hold on
end
grid on

%Grafico de la pieza sinterizada.
for e=1:nelem
   plot (rzsinter(conecv(e,:),1),rzsinter(conecv(e,:),2),'b')
end
grid on

%Grafico de la pieza completa modelada.
```

```
for e=1:nelem
   plot (rzsd(conecv(e,:),1),rzsd(conecv(e,:),2),'-r')
end

%-------------------------------------------------------------
%Grafico de la pieza real.

%Nombre del subdirectorio donde están los datos del plato real.
subdir_real='platoreal';

%Apertura archivo elementos_real.txt
fname=strcat(subdir_real,'\elementos_real.txt');
fid=fopen(fname,'rt');

%Leer titulo y número de elementos
nelem_real=fscanf(fid,'%i',1);
tit=fscanf(fid,'%s',6);

%lee nodos de cada elemento.
for i=1:nelem_real
   n=fscanf(fid,'%i',1);
   fscanf(fid,'%s',4);
   for j=1:4
      conecr(n,j)=fscanf(fid,'%i',1);
   end
end

%Cierra archivo
fclose(fid);

clear tit fid i n j

%-------------------------------------------------------------

%Apertura del archivo nodos.txt
fname=strcat(subdir_real,'/nodos_real.txt');
fid=fopen(fname,'rt');

%Lee titulo y número de nodos
nnodos_real=fscanf(fid,'%i',1);
peana=fscanf(fid,'%i',1);
tit=fscanf(fid,'%s',4);
```

```
%lectura de los nodos
for i=1:nnodos_real
   n=fscanf(fid,'%i',1);
   rz_real(n,1)=fscanf(fid,'%g',1);
   rz_real(n,2)=fscanf(fid,'%g',1);
   tit=fscanf(fid,'%g',1);
end

rz_real=rz_real/1000; %Las coordenadas de cosmos están en [mm],
hay que
%pasarlas a [m]

%Cierre de archivo
fclose(fid);

%Borrar variables locales
clear tit fname fid i n

%-----------------------------------------------------------------
%Traslado de la figura al mismo punto de la peana
for i=1:ngrest
     if grest(i,2)==0
         rpeana=grest(i,1);
     end
 end

if rz_real(peana,2)>rz_sinter(rpeana,2)
    dis_peana=rz_real(peana,2)-rz_sinter(rpeana,2);
else
    dis_peana=rz_sinter(rpeana,2)-rz_real(peana,2);
end
rz_real(:,2)=rz_real(:,2)-dis_peana;

%Grafica de la pieza real.
conecr=[conecr(:,1) conecr(:,2) conecr(:,3) conecr(:,4) conecr(:,1)];

for e=1:nelem_real
   plot (rz_real(conecr(e,:),1),rz_real(conecr(e,:),2),'m--')
   axis equal
   hold on
end
grid on
xlabel('r [m]'), ylabel('z [m]')
```

```
title ('Modelacion del comportamiento de un plato ceramico en coccion')
%-------------------------------------------------------------------------
%Comparacion de los resultados con los datos experimentales
Variacion_radial_maxima=(max(rzsd(:,1))-max(rz_real(:,1)))/
max(rz_real(:,1))*100;
Variacion_axial_punto00=(rzsd(1,2)-rz_real(1,2))/rz_real(1,2)*100;
Variacion_axial_maxima=(max(rzsd(:,2))-max(rz_real(1,2)))/
max(rz_real(1,2))*100;
%cd('resultados');
%cd('extensionh');
%cd(subdir);
%save spacework;
%saveas(1,'def_p0');
%saveas(2,'dis_tension');
%saveas(3,'plato');
%saveas(4,'error');
%saveas(5,'dis_vel');
%saveas(6,'Tension_eff');
%saveas(7,'velocidad_creep');
%cd ..
%cd ..
%cd ..
```

Archivo: P1

```
P1%P1.m

%-------------------------------------------------------------------------
%  Propósito:
% Subprograma para calcular las funciones de forma y sus derivadas
% con
% grado polinomial p=1.
%
%  Sintaxis:
% Es llamado dentro de la función F_form
%
%  Descripción de variables:
% FN: Vector simbolico de Funciones de forma a usar
% SDNs: Vector simbólico de derivadas con respecto a la variable
% local s.
% SDNn: Vector simbólico de derivadas con respecto a la variable
% local n.
% NI: Vector de funciones de forma evaluadas en el punto de Gauss
```

```
% DNs: Vector de derivadas con respecto a la variable local s.
% DNn: Vector de derivadas con respecto a la variable local n.
% nfn: contador del número de funciones de forma
%--------------------------------------------------------------------

%Funciones de forma expresadas simbolicamente
FN(1)=sym('0.25*(1-s)*(1-n)');
FN(2)=sym('0.25*(1+s)*(1-n)');
FN(3)=sym('0.25*(1+s)*(1+n)');
FN(4)=sym('0.25*(1-s)*(1+n)');

%Calculo simbólico de las derivadas parciales de las funciones de forma
SDNs(1)=sym('-.25*(1-n)');
SDNn(1)=sym('-.25*(1-s)');
SDNs(2)=sym('.25*(1-n)');
SDNn(2)=sym('-.25*(1+s)');
SDNs(3)=sym('.25*(1+n)');
SDNn(3)=sym('.25*(1+s)');
SDNs(4)=sym('-.25*(1+n)');
SDNn(4)=sym('.25*(1-s)');

%Evaluación de las funciones de forma NI y sus derivadas en los puntos
% de Gauss,
%como vectores filas
CNG=0;
for intr=1:NG
    s=p(intr);
    for intz=1:NG
        n=p(intz);
        CNG=CNG+1;
        for nfn=1:N
            NI(nfn,CNG)=eval(FN(nfn));
            DNs(nfn,CNG)=eval(SDNs(nfn));
            DNn(nfn,CNG)=eval(SDNn(nfn));
        end
    end
end
clear CNG
```

Archivo: P2

```
%P2.m
```

```
%-------------------------------------------------------------------
%  Propósito:
% Subprograma para calcular las funciones de forma y sus derivadas
% con
% grado polinomial p=2.
%
%  Sintaxis:
% Es llamado dentro de la función F_form
%
%  Descripción de variables:
% FN: Vector simbolico de Funciones de forma a usar
% SDNs: Vector simbólico de derivadas con respecto a la variable
% local s.
% SDNn: Vector simbólico de derivadas con respecto a la variable
% local n.
% NI: Vector de funciones de forma evaluadas en el punto de Gauss
% DNs: Vector de derivadas con respecto a la variable local s.
% DNn: Vector de derivadas con respecto a la variable local n.
% nfn: contador del número de funciones de forma
%-------------------------------------------------------------------

%Funciones de forma nodales expresadas simbolicamente
FN(1)=sym('0.25*(1-s)*(1-n)');
FN(2)=sym('0.25*(1+s)*(1-n)');
FN(3)=sym('0.25*(1+s)*(1+n)');
FN(4)=sym('0.25*(1-s)*(1+n)');

%Funciones de forma laterales expresadas simbolicamente
FN(5)=sym('0.25*sqrt(1.5)*(1-n)*(s^2-1)');
FN(6)=sym('0.25*sqrt(1.5)*(1-s)*(n^2-1)');
FN(7)=sym('0.25*sqrt(1.5)*(1+n)*(s^2-1)');
FN(8)=sym('0.25*sqrt(1.5)*(1+s)*(n^2-1)');

%Calculo simbólico de las derivadas parciales de las funciones de forma
SDNs(1)=sym('-.25*(1-n)');
SDNn(1)=sym('-.25*(1-s)');
SDNs(2)=sym('.25*(1-n)');
SDNn(2)=sym('-.25*(1+s)');
SDNs(3)=sym('.25*(1+n)');
SDNn(3)=sym('.25*(1+s)');
SDNs(4)=sym('-.25*(1+n)');
SDNn(4)=sym('.25*(1-s)');
SDNs(5)=sym('0.25*sqrt(1.5)*(1-n)*2*s');
```

```
SDNn(5)=sym('-0.25*sqrt(1.5)*(s^2-1)');
SDNs(6)=sym('-0.25*sqrt(1.5)*(n^2-1)');
SDNn(6)=sym('0.25*sqrt(1.5)*(1-s)*2*n');
SDNs(7)=sym('0.25*sqrt(1.5)*(1+n)*2*s');
SDNn(7)=sym('0.25*sqrt(1.5)*(s^2-1)');
SDNs(8)=sym('0.25*sqrt(1.5)*(n^2-1)');
SDNn(8)=sym('0.25*sqrt(1.5)*(1+s)*2*n');


%Evaluación de las funciones de forma NI y sus derivadas en los puntos
% de Gauss,
%como vectores filas
CNG=0;
for intr=1:NG
    s=p(intr);
    for intz=1:NG
        n=p(intz);
        CNG=CNG+1;
        for nfn=1:N
            NI(nfn,CNG)=eval(FN(nfn));
            DNs(nfn,CNG)=eval(SDNs(nfn));
            DNn(nfn,CNG)=eval(SDNn(nfn));
        end
    end
end
clear CNG
```

Archivo: p3

```
%p3.m

%-------------------------------------------------------------------
%  Propósito:
% Subprograma para calcular las funciones de forma y sus derivadas
% con
% grado polinomial p=3.
%
%  Sintaxis:
% Es llamado dentro de la función F_form
%
%  Descripción de variables:
% FN: Vector simbolico de Funciones de forma a usar
% SDNs: Vector simbólico de derivadas con respecto a la variable
```

```
% local s.
% SDNn: Vector simbólico de derivadas con respecto a la variable
% local n.
% NI: Vector de funciones de forma evaluadas en el punto de Gauss
% DNs: Vector de derivadas con respecto a la variable local s.
% DNn: Vector de derivadas con respecto a la variable local n.
% nfn: contador del número de funciones de forma
%-------------------------------------------------------------------

%Funciones de forma nodales expresadas simbolicamente
FN(1)=sym('0.25*(1-s)*(1-n)');
FN(2)=sym('0.25*(1+s)*(1-n)');
FN(3)=sym('0.25*(1+s)*(1+n)');
FN(4)=sym('0.25*(1-s)*(1+n)');

%Funciones de forma laterales expresadas simbolicamente
FN(5)=sym('0.25*sqrt(1.5)*(1-n)*(s^2-1)');
FN(6)=sym('0.25*sqrt(1.5)*(1-s)*(n^2-1)');
FN(7)=sym('0.25*sqrt(1.5)*(1+n)*(s^2-1)');
FN(8)=sym('0.25*sqrt(1.5)*(1+s)*(n^2-1)');
FN(9)=sym('0.25*sqrt(2.5)*(1-n)*(s^3-s)');
FN(10)=sym('0.25*sqrt(2.5)*(1-s)*(n^3-n)');
FN(11)=sym('0.25*sqrt(2.5)*(1+n)*(s^3-s)');
FN(12)=sym('0.25*sqrt(2.5)*(1+s)*(n^3-n)');


%Calculo simbólico de las derivadas parciales de las funciones
%de forma
SDNs(1)=sym('-.25*(1-n)');
SDNn(1)=sym('-.25*(1-s)');
SDNs(2)=sym('.25*(1-n)');
SDNn(2)=sym('-.25*(1+s)');
SDNs(3)=sym('.25*(1+n)');
SDNn(3)=sym('.25*(1+s)');
SDNs(4)=sym('-.25*(1+n)');
SDNn(4)=sym('.25*(1-s)');
SDNs(5)=sym('0.25*sqrt(1.5)*(1-n)*2*s');
SDNn(5)=sym('-0.25*sqrt(1.5)*(s^2-1)');
SDNs(6)=sym('-0.25*sqrt(1.5)*(n^2-1)');
SDNn(6)=sym('0.25*sqrt(1.5)*(1-s)*2*n');
SDNs(7)=sym('0.25*sqrt(1.5)*(1+n)*2*s');
SDNn(7)=sym('0.25*sqrt(1.5)*(s^2-1)');
SDNs(8)=sym('0.25*sqrt(1.5)*(n^2-1)');
```

```
SDNn(8)=sym('0.25*sqrt(1.5)*(1+s)*2*n');
SDNs(9)=sym('0.25*sqrt(2.5)*(1-n)*(3*s^2-1)');
SDNn(9)=sym('-0.25*sqrt(2.5)*(s^3-s)');
SDNs(10)=sym('-0.25*sqrt(2.5)*(n^3-n)');
SDNn(10)=sym('0.25*sqrt(2.5)*(1-s)*(3*n^2-1)');
SDNs(11)=sym('0.25*sqrt(2.5)*(1+n)*(3*s^2-1)');
SDNn(11)=sym('0.25*sqrt(2.5)*(s^3-s)');
SDNs(12)=sym('0.25*sqrt(2.5)*(n^3-n)');
SDNn(12)=sym('0.25*sqrt(2.5)*(1+s)*(3*n^2-1)');

%Evaluación de las funciones de forma NI y sus derivadas en los puntos
% de Gauss,
%como vectores filas
CNG=0;
for intr=1:NG
    s=p(intr);
    for intz=1:NG
        n=p(intz);
        CNG=CNG+1;
        for nfn=1:N
            NI(nfn,CNG)=eval(FN(nfn));
            DNs(nfn,CNG)=eval(SDNs(nfn));
            DNn(nfn,CNG)=eval(SDNn(nfn));
        end
    end
end
clear CNG
```

Archivo: P4

```
%P4.m

%-----------------------------------------------------------------------
%  Propósito:
% Subprograma para calcular las funciones de forma y sus derivadas
% con grado polinomial p=3.
%
%  Sintaxis:
% Es llamado dentro de la función F_form
%
%  Descripción de variables:
% FN: Vector simbolico de Funciones de forma a usar
% SDNs: Vector simbólico de derivadas con respecto a la variable
```

```
%  local s.
% SDNn: Vector simbólico de derivadas con respecto a la variable
% local n.
% NI: Vector de funciones de forma evaluadas en el punto de Gauss
% DNs: Vector de derivadas con respecto a la variable local s.
% DNn: Vector de derivadas con respecto a la variable local n.
% nfn: contador del número de funciones de forma
%-------------------------------------------------------------------

%Funciones de forma nodales expresadas simbolicamente
FN(1)=sym('0.25*(1-s)*(1-n)');
FN(2)=sym('0.25*(1+s)*(1-n)');
FN(3)=sym('0.25*(1+s)*(1+n)');
FN(4)=sym('0.25*(1-s)*(1+n)');

%Funciones de forma laterales expresadas simbolicamente
FN(5)=sym('0.25*sqrt(1.5)*(1-n)*(s^2-1)');
FN(6)=sym('0.25*sqrt(1.5)*(1-s)*(n^2-1)');
FN(7)=sym('0.25*sqrt(1.5)*(1+n)*(s^2-1)');
FN(8)=sym('0.25*sqrt(1.5)*(1+s)*(n^2-1)');
FN(9)=sym('0.25*sqrt(2.5)*(1-n)*(s^3-s)');
FN(10)=sym('0.25*sqrt(2.5)*(1-s)*(n^3-n)');
FN(11)=sym('0.25*sqrt(2.5)*(1+n)*(s^3-s)');
FN(12)=sym('0.25*sqrt(2.5)*(1+s)*(n^3-n)');
FN(13)=sym('1/16*sqrt(3.5)*(1-n)*(5*s^4-6*s^2+1)');
FN(14)=sym('1/16*sqrt(3.5)*(1+s)*(5*n^4-6*n^2+1)');
FN(15)=sym('1/16*sqrt(3.5)*(1+n)*(5*s^4-6*s^2+1)');
FN(16)=sym('1/16*sqrt(3.5)*(1+s)*(5*n^4-6*n^2+1)');

%Funciones de forma interna
FN(17)=sym('3/8*(s^2-1)*(n^2-1)');


%Calculo simbólico de las derivadas parciales de las funciones de forma
SDNs(1)=sym('-.25*(1-n)');
SDNn(1)=sym('-.25*(1-s)');
SDNs(2)=sym('.25*(1-n)');
SDNn(2)=sym('-.25*(1+s)');
SDNs(3)=sym('.25*(1+n)');
SDNn(3)=sym('.25*(1+s)');
SDNs(4)=sym('-.25*(1+n)');
SDNn(4)=sym('.25*(1-s)');
SDNs(5)=sym('0.25*sqrt(1.5)*(1-n)*2*s');
```

```
SDNn(5)=sym('-0.25*sqrt(1.5)*(s^2-1)');
SDNs(6)=sym('-0.25*sqrt(1.5)*(n^2-1)');
SDNn(6)=sym('0.25*sqrt(1.5)*(1-s)*2*n');
SDNs(7)=sym('0.25*sqrt(1.5)*(1+n)*2*s');
SDNn(7)=sym('0.25*sqrt(1.5)*(s^2-1)');
SDNs(8)=sym('0.25*sqrt(1.5)*(n^2-1)');
SDNn(8)=sym('0.25*sqrt(1.5)*(1+s)*2*n');
SDNs(9)=sym('0.25*sqrt(2.5)*(1-n)*(3*s^2-1)');
SDNn(9)=sym('-0.25*sqrt(2.5)*(s^3-s)');
SDNs(10)=sym('-0.25*sqrt(2.5)*(n^3-n)');
SDNn(10)=sym('0.25*sqrt(2.5)*(1-s)*(3*n^2-1)');
SDNs(11)=sym('0.25*sqrt(2.5)*(1+n)*(3*s^2-1)');
SDNn(11)=sym('0.25*sqrt(2.5)*(s^3-s)');
SDNs(12)=sym('0.25*sqrt(2.5)*(n^3-n)');
SDNn(12)=sym('0.25*sqrt(2.5)*(1+s)*(3*n^2-1)');
SDNs(13)=sym('1/16*sqrt(3.5)*(1-n)*(20*s^3-12*s)');
SDNn(13)=sym('-1/16*sqrt(3.5)*(5*s^4-6*s^2+1)');
SDNs(14)=sym('-1/16*sqrt(3.5)*(5*n^4-6*n^2+1)');
SDNn(14)=sym('1/16*sqrt(3.5)*(1+s)*(20*n^3-12*n)');
SDNs(15)=sym('1/16*sqrt(3.5)*(1+n)*(20*s^3-12*s)');
SDNn(15)=sym('1/16*sqrt(3.5)*(5*s^4-6*s^2+1)');
SDNs(16)=sym('1/16*sqrt(3.5)*(5*n^4-6*n^2+1)');
SDNn(16)=sym('1/16*sqrt(3.5)*(1+s)*(20*n^3-12*n)');
SDNs(17)=sym('3/4*s*(n^2-1)');
SDNn(17)=sym('3/4*n*(s^2-1)');

%Evaluación de las funciones de forma NI y sus derivadas en los puntos
% de Gauss,
%como vectores filas
%for nfn=1:N
 %  NI(nfn)=eval(FN(nfn));
  % DNs(nfn)=eval(SDNs(nfn));
   %DNn(nfn)=eval(SDNn(nfn));
   %end

CNG=0;
for intr=1:NG
    s=p(intr);
    for intz=1:NG
        n=p(intz);
        CNG=CNG+1;
        for nfn=1:N
            NI(nfn,CNG)=eval(FN(nfn));
```

```
            DNs(nfn,CNG)=eval(SDNs(nfn));
            DNn(nfn,CNG)=eval(SDNn(nfn));
        end
    end
end
clear CNG
```

Archivo: P5

```
%P5.m

%--------------------------------------------------------------------
%  Propósito:
% Subprograma para calcular las funciones de forma y sus derivadas
%  con grado polinomial p=3.
%
%  Sintaxis:
% Es llamado dentro de la función F_form
%
%  Descripción de variables:
% FN: Vector simbolico de Funciones de forma a usar
% SDNs: Vector simbólico de derivadas con respecto a la variable
%  local s.
% SDNn: Vector simbólico de derivadas con respecto a la variable
%  local n.
% NI: Vector de funciones de forma evaluadas en el punto de Gauss
% DNs: Vector de derivadas con respecto a la variable local s.
% DNn: Vector de derivadas con respecto a la variable local n.
% nfn: contador del número de funciones de forma
%--------------------------------------------------------------------

%Funciones de forma nodales expresadas simbolicamente
FN(1)=sym('0.25*(1-s)*(1-n)');
FN(2)=sym('0.25*(1+s)*(1-n)');
FN(3)=sym('0.25*(1+s)*(1+n)');
FN(4)=sym('0.25*(1-s)*(1+n)');

%Funciones de forma laterales expresadas simbolicamente
FN(5)=sym('0.25*sqrt(1.5)*(1-n)*(s^2-1)');
FN(6)=sym('0.25*sqrt(1.5)*(1-s)*(n^2-1)');
FN(7)=sym('0.25*sqrt(1.5)*(1+n)*(s^2-1)');
FN(8)=sym('0.25*sqrt(1.5)*(1+s)*(n^2-1)');
FN(9)=sym('0.25*sqrt(2.5)*(1-n)*(s^3-s)');
```

```
FN(10)=sym('0.25*sqrt(2.5)*(1-s)*(n^3-n)');
FN(11)=sym('0.25*sqrt(2.5)*(1+n)*(s^3-s)');
FN(12)=sym('0.25*sqrt(2.5)*(1+s)*(n^3-n)');
FN(13)=sym('1/16*sqrt(3.5)*(1-n)*(5*s^4-6*s^2+1)');
FN(14)=sym('1/16*sqrt(3.5)*(1+s)*(5*n^4-6*n^2+1)');
FN(15)=sym('1/16*sqrt(3.5)*(1+n)*(5*s^4-6*s^2+1)');
FN(16)=sym('1/16*sqrt(3.5)*(1+s)*(5*n^4-6*n^2+1)');
FN(17)=sym('1/16*sqrt(4.5)*(1-n)*(7*s^5-10*s^3+3*s)');
FN(18)=sym('1/16*sqrt(4.5)*(1-s)*(7*n^5-10*n^3+3*n)');
FN(19)=sym('1/16*sqrt(4.5)*(1+n)*(7*s^5-10*s^3+3*s)');
FN(20)=sym('1/16*sqrt(4.5)*(1+s)*(7*n^5-10*n^3+3*n)');

%Funciones de forma interna
FN(21)=sym('3/8*(s^2-1)*(n^2-1)');
FN(22)=sym('sqrt(15)/8*(s^3-s)*(n^2-1)');
FN(23)=sym('sqrt(15)/8*(n^3-n)*(s^2-1)');

%Calculo simbólico de las derivadas parciales de las funciones de
% forma
SDNs(1)=sym('-.25*(1-n)');
SDNn(1)=sym('-.25*(1-s)');
SDNs(2)=sym('.25*(1-n)');
SDNn(2)=sym('-.25*(1+s)');
SDNs(3)=sym('.25*(1+n)');
SDNn(3)=sym('.25*(1+s)');
SDNs(4)=sym('-.25*(1+n)');
SDNn(4)=sym('.25*(1-s)');
SDNs(5)=sym('0.25*sqrt(1.5)*(1-n)*2*s');
SDNn(5)=sym('-0.25*sqrt(1.5)*(s^2-1)');
SDNs(6)=sym('-0.25*sqrt(1.5)*(n^2-1)');
SDNn(6)=sym('0.25*sqrt(1.5)*(1-s)*2*n');
SDNs(7)=sym('0.25*sqrt(1.5)*(1+n)*2*s');
SDNn(7)=sym('0.25*sqrt(1.5)*(s^2-1)');
SDNs(8)=sym('0.25*sqrt(1.5)*(n^2-1)');
SDNn(8)=sym('0.25*sqrt(1.5)*(1+s)*2*n');
SDNs(9)=sym('0.25*sqrt(2.5)*(1-n)*(3*s^2-1)');
SDNn(9)=sym('-0.25*sqrt(2.5)*(s^3-s)');
SDNs(10)=sym('-0.25*sqrt(2.5)*(n^3-n)');
SDNn(10)=sym('0.25*sqrt(2.5)*(1-s)*(3*n^2-1)');
SDNs(11)=sym('0.25*sqrt(2.5)*(1+n)*(3*s^2-1)');
SDNn(11)=sym('0.25*sqrt(2.5)*(s^3-s)');
SDNs(12)=sym('0.25*sqrt(2.5)*(n^3-n)');
SDNn(12)=sym('0.25*sqrt(2.5)*(1+s)*(3*n^2-1)');
```

```
SDNs(13)=sym('1/16*sqrt(3.5)*(1-n)*(20*s^3-12*s)');
SDNn(13)=sym('-1/16*sqrt(3.5)*(5*s^4-6*s^2+1)');
SDNs(14)=sym('-1/16*sqrt(3.5)*(5*n^4-6*n^2+1)');
SDNn(14)=sym('1/16*sqrt(3.5)*(1+s)*(20*n^3-12*n)');
SDNs(15)=sym('1/16*sqrt(3.5)*(1+n)*(20*s^3-12*s)');
SDNn(15)=sym('1/16*sqrt(3.5)*(5*s^4-6*s^2+1)');
SDNs(16)=sym('1/16*sqrt(3.5)*(5*n^4-6*n^2+1)');
SDNn(16)=sym('1/16*sqrt(3.5)*(1+s)*(20*n^3-12*n)');
SDNs(17)=sym('1/16*sqrt(4.5)*(1-n)*(35*s^4-30*s^2+3)');
SDNn(17)=sym('-1/16*sqrt(4.5)*(7*s^5-10*s^3+3*s)');
SDNs(18)=sym('-1/16*sqrt(4.5)*(7*n^5-10*n^3+3*n)');
SDNn(18)=sym('1/16*sqrt(4.5)*(1-s)*(35*n^4-30*n^2+3)');
SDNs(19)=sym('1/16*sqrt(4.5)*(1+n)*(35*s^4-30*s^2+3)');
SDNn(19)=sym('1/16*sqrt(4.5)*(7*s^5-10*s^3+3*s)');
SDNs(20)=sym('1/16*sqrt(4.5)*(7*n^5-10*n^3+3*n)');
SDNn(20)=sym('1/16*sqrt(4.5)*(1+s)*(35*n^4-30*n^2+3)');
SDNs(21)=sym('3/4*s*(n^2-1)');
SDNn(21)=sym('3/4*n*(s^2-1)');
SDNs(22)=sym('sqrt(15)/8*(3*s^2-1)*(n^2-1)');
SDNn(22)=sym('sqrt(15)/4*n*(s^3-s)');
SDNs(23)=sym('sqrt(15)/4*s*(n^3-n)');
SDNn(23)=sym('sqrt(15)/8*(3*n^2-1)*(s^2-1)');


%Evaluación de las funciones de forma NI y sus derivadas en los
% puntos de Gauss,
%como vectores filas
CNG=0;
for intr=1:NG
    s=p(intr);
    for intz=1:NG
        n=p(intz);
        CNG=CNG+1;
        for nfn=1:N
            NI(nfn,CNG)=eval(FN(nfn));
            DNs(nfn,CNG)=eval(SDNs(nfn));
            DNn(nfn,CNG)=eval(SDNn(nfn));
        end
    end
end
clear CNG
```

Archivo: P6

```
%P6.m

%-------------------------------------------------------------------
%  Propósito:
% Subprograma para calcular las funciones de forma y sus derivadas
%       con grado polinomial p=6.
%
%  Sintaxis:
% Es llamado dentro de la función F_form
%
%  Descripción de variables:
% FN: Vector simbolico de Funciones de forma a usar
% SDNs: Vector simbólico de derivadas con respecto a la variable
%  local s.
% SDNn: Vector simbólico de derivadas con respecto a la variable
%  local n.
% NI: Vector de funciones de forma evaluadas en el punto de Gauss
% DNs: Vector de derivadas con respecto a la variable local s.
% DNn: Vector de derivadas con respecto a la variable local n.
% nfn: contador del número de funciones de forma
%-------------------------------------------------------------------

%Funciones de forma nodales expresadas simbolicamente
FN(1)=sym('0.25*(1-s)*(1-n)');
FN(2)=sym('0.25*(1+s)*(1-n)');
FN(3)=sym('0.25*(1+s)*(1+n)');
FN(4)=sym('0.25*(1-s)*(1+n)');

%Funciones de forma laterales expresadas simbolicamente
FN(5)=sym('0.25*sqrt(1.5)*(1-n)*(s^2-1)');
FN(6)=sym('0.25*sqrt(1.5)*(1-s)*(n^2-1)');
FN(7)=sym('0.25*sqrt(1.5)*(1+n)*(s^2-1)');
FN(8)=sym('0.25*sqrt(1.5)*(1+s)*(n^2-1)');
FN(9)=sym('0.25*sqrt(2.5)*(1-n)*(s^3-s)');
FN(10)=sym('0.25*sqrt(2.5)*(1-s)*(n^3-n)');
FN(11)=sym('0.25*sqrt(2.5)*(1+n)*(s^3-s)');
FN(12)=sym('0.25*sqrt(2.5)*(1+s)*(n^3-n)');
FN(13)=sym('1/16*sqrt(3.5)*(1-n)*(5*s^4-6*s^2+1)');
FN(14)=sym('1/16*sqrt(3.5)*(1+s)*(5*n^4-6*n^2+1)');
FN(15)=sym('1/16*sqrt(3.5)*(1+n)*(5*s^4-6*s^2+1)');
FN(16)=sym('1/16*sqrt(3.5)*(1+s)*(5*n^4-6*n^2+1)');
FN(17)=sym('1/16*sqrt(4.5)*(1-n)*(7*s^5-10*s^3+3*s)');
FN(18)=sym('1/16*sqrt(4.5)*(1-s)*(7*n^5-10*n^3+3*n)');
```

```
FN(19)=sym('1/16*sqrt(4.5)*(1+n)*(7*s^5-10*s^3+3*s)');
FN(20)=sym('1/16*sqrt(4.5)*(1+s)*(7*n^5-10*n^3+3*n)');
FN(21)=sym('1/32*sqrt(5.5)*(1-n)*(21*s^6-35*s^4+15*s^2-1)');
FN(22)=sym('1/32*sqrt(5.5)*(1-s)*(21*n^6-35*n^4+15*n^2-1)');
FN(23)=sym('1/32*sqrt(5.5)*(1+n)*(21*s^6-35*s^4+15*s^2-1)');
FN(24)=sym('1/32*sqrt(5.5)*(1+s)*(21*n^6-35*n^4+15*n^2-1)');

%Funciones de forma interna
FN(25)=sym('3/8*(s^2-1)*(n^2-1)');
FN(26)=sym('sqrt(15)/8*(s^3-s)*(n^2-1)');
FN(27)=sym('sqrt(15)/8*(n^3-n)*(s^2-1)');
FN(28)=sym('sqrt(21)/32*(5*s^4-6*s^2+1)*(n^2-1)');
FN(29)=sym('5/8*(s^3-s)*(n^3-n)');
FN(30)=sym('sqrt(21)/32*(s^2-1)*(5*n^4-6*n^2+1)');

%Calculo simbólico de las derivadas parciales de las funciones
% de forma
SDNs(1)=sym('-.25*(1-n)');
SDNn(1)=sym('-.25*(1-s)');
SDNs(2)=sym('.25*(1-n)');
SDNn(2)=sym('-.25*(1+s)');
SDNs(3)=sym('.25*(1+n)');
SDNn(3)=sym('.25*(1+s)');
SDNs(4)=sym('-.25*(1+n)');
SDNn(4)=sym('.25*(1-s)');
SDNs(5)=sym('0.25*sqrt(1.5)*(1-n)*2*s');
SDNn(5)=sym('-0.25*sqrt(1.5)*(s^2-1)');
SDNs(6)=sym('-0.25*sqrt(1.5)*(n^2-1)');
SDNn(6)=sym('0.25*sqrt(1.5)*(1-s)*2*n');
SDNs(7)=sym('0.25*sqrt(1.5)*(1+n)*2*s');
SDNn(7)=sym('0.25*sqrt(1.5)*(s^2-1)');
SDNs(8)=sym('0.25*sqrt(1.5)*(n^2-1)');
SDNn(8)=sym('0.25*sqrt(1.5)*(1+s)*2*n');
SDNs(9)=sym('0.25*sqrt(2.5)*(1-n)*(3*s^2-1)');
SDNn(9)=sym('-0.25*sqrt(2.5)*(s^3-s)');
SDNs(10)=sym('-0.25*sqrt(2.5)*(n^3-n)');
SDNn(10)=sym('0.25*sqrt(2.5)*(1-s)*(3*n^2-1)');
SDNs(11)=sym('0.25*sqrt(2.5)*(1+n)*(3*s^2-1)');
SDNn(11)=sym('0.25*sqrt(2.5)*(s^3-s)');
SDNs(12)=sym('0.25*sqrt(2.5)*(n^3-n)');
SDNn(12)=sym('0.25*sqrt(2.5)*(1+s)*(3*n^2-1)');
SDNs(13)=sym('1/16*sqrt(3.5)*(1-n)*(20*s^3-12*s)');
SDNn(13)=sym('-1/16*sqrt(3.5)*(5*s^4-6*s^2+1)');
```

```
SDNs(14)=sym('-1/16*sqrt(3.5)*(5*n^4-6*n^2+1)');
SDNn(14)=sym('1/16*sqrt(3.5)*(1+s)*(20*n^3-12*n)');
SDNs(15)=sym('1/16*sqrt(3.5)*(1+n)*(20*s^3-12*s)');
SDNn(15)=sym('1/16*sqrt(3.5)*(5*s^4-6*s^2+1)');
SDNs(16)=sym('1/16*sqrt(3.5)*(5*n^4-6*n^2+1)');
SDNn(16)=sym('1/16*sqrt(3.5)*(1+s)*(20*n^3-12*n)');
SDNs(17)=sym('1/16*sqrt(4.5)*(1-n)*(35*s^4-30*s^2+3)');
SDNn(17)=sym('-1/16*sqrt(4.5)*(7*s^5-10*s^3+3*s)');
SDNs(18)=sym('-1/16*sqrt(4.5)*(7*n^5-10*n^3+3*n)');
SDNn(18)=sym('1/16*sqrt(4.5)*(1-s)*(35*n^4-30*n^2+3)');
SDNs(19)=sym('1/16*sqrt(4.5)*(1+n)*(35*s^4-30*s^2+3)');
SDNn(19)=sym('1/16*sqrt(4.5)*(7*s^5-10*s^3+3*s)');
SDNs(20)=sym('1/16*sqrt(4.5)*(7*n^5-10*n^3+3*n)');
SDNn(20)=sym('1/16*sqrt(4.5)*(1+s)*(35*n^4-30*n^2+3)');
SDNs(21)=sym('1/32*sqrt(5.5)*(1-n)*(126*s^5-140*s^3+30*s)');
SDNn(21)=sym('-1/32*sqrt(5.5)*(21*s^6-35*s^4+15*s^2-1)');
SDNs(22)=sym('-1/32*sqrt(5.5)*(21*n^6-35*n^4+15*n^2-1)');
SDNn(22)=sym('1/32*sqrt(5.5)*(1-s)*(126*n^5-140*n^3+30*n)');
SDNs(23)=sym('1/32*sqrt(5.5)*(1+n)*(126*s^5-140*s^3+30*s)');
SDNn(23)=sym('1/32*sqrt(5.5)*(21*s^6-35*s^4+15*s^2-1)');
SDNs(24)=sym('1/32*sqrt(5.5)*(21*n^6-35*n^4+15*n^2-1)');
SDNn(24)=sym('1/32*sqrt(5.5)*(1+s)*(126*n^5-140*n^3+30*n)');
SDNs(25)=sym('3/4*s*(n^2-1)');
SDNn(25)=sym('3/4*n*(s^2-1)');
SDNs(26)=sym('sqrt(15)/8*(3*s^2-1)*(n^2-1)');
SDNn(26)=sym('sqrt(15)/4*n*(s^3-s)');
SDNs(27)=sym('sqrt(15)/4*s*(n^3-n)');
SDNn(27)=sym('sqrt(15)/8*(3*n^2-1)*(s^2-1)');
SDNs(28)=sym('sqrt(21)/32*(20*s^3-12*s)*(n^2-1)');
SDNn(28)=sym('sqrt(21)/16*n*(5*s^4-6*s^2+1)');
SDNs(29)=sym('5/8*(3*s^2-1)*(n^3-n)');
SDNn(29)=sym('5/8*(s^3-s)*(n^2-1)');
SDNs(30)=sym('sqrt(21)/16*s*(5*n^4-6*n^2+1)');
SDNn(30)=sym('sqrt(21)/32*(s^2-1)*(20*n^3-12*n)');

%Evaluación de las funciones de forma NI y sus derivadas en los
% puntos de Gauss,
%como vectores filas
CNG=0;
for intr=1:NG
    s=p(intr);
    for intz=1:NG
        n=p(intz);
```

```
        CNG=CNG+1;
        for nfn=1:N
            NI(nfn,CNG)=eval(FN(nfn));
            DNs(nfn,CNG)=eval(SDNs(nfn));
            DNn(nfn,CNG)=eval(SDNn(nfn));
        end
    end
end
clear CNG
```

Archivo: p7

```
%P7.m

%-------------------------------------------------------------------
%  Propósito:
% Subprograma para calcular las funciones de forma y sus derivadas
%   con grado polinomial p=7.
%
%  Sintaxis:
% Es llamado dentro de la función F_form
%
%  Descripción de variables:
% FN: Vector simbolico de Funciones de forma a usar
% SDNs: Vector simbólico de derivadas con respecto a la variable
%   local s.
% SDNn: Vector simbólico de derivadas con respecto a la variable
%  local n.
% NI: Vector de funciones de forma evaluadas en el punto de Gauss
% DNs: Vector de derivadas con respecto a la variable local s.
% DNn: Vector de derivadas con respecto a la variable local n.
% nfn: contador del número de funciones de forma
%-------------------------------------------------------------------
%Funciones de forma nodales expresadas simbolicamente
FN(1)=sym('0.25*(1-s)*(1-n)');
FN(2)=sym('0.25*(1+s)*(1-n)');
FN(3)=sym('0.25*(1+s)*(1+n)');
FN(4)=sym('0.25*(1-s)*(1+n)');

%Funciones de forma laterales expresadas simbolicamente
FN(5)=sym('0.25*sqrt(1.5)*(1-n)*(s^2-1)');
FN(6)=sym('0.25*sqrt(1.5)*(1-s)*(n^2-1)');
FN(7)=sym('0.25*sqrt(1.5)*(1+n)*(s^2-1)');
```

```
FN(8)=sym('0.25*sqrt(1.5)*(1+s)*(n^2-1)');
FN(9)=sym('0.25*sqrt(2.5)*(1-n)*(s^3-s)');
FN(10)=sym('0.25*sqrt(2.5)*(1-s)*(n^3-n)');
FN(11)=sym('0.25*sqrt(2.5)*(1+n)*(s^3-s)');
FN(12)=sym('0.25*sqrt(2.5)*(1+s)*(n^3-n)');
FN(13)=sym('1/16*sqrt(3.5)*(1-n)*(5*s^4-6*s^2+1)');
FN(14)=sym('1/16*sqrt(3.5)*(1+s)*(5*n^4-6*n^2+1)');
FN(15)=sym('1/16*sqrt(3.5)*(1+n)*(5*s^4-6*s^2+1)');
FN(16)=sym('1/16*sqrt(3.5)*(1+s)*(5*n^4-6*n^2+1)');
FN(18)=sym('1/16*sqrt(4.5)*(1-n)*(7*s^5-10*s^3+3*s)');
FN(19)=sym('1/16*sqrt(4.5)*(1-s)*(7*n^5-10*n^3+3*n)');
FN(20)=sym('1/16*sqrt(4.5)*(1+n)*(7*s^5-10*s^3+3*s)');
FN(21)=sym('1/16*sqrt(4.5)*(1+s)*(7*n^5-10*n^3+3*n)');
FN(24)=sym('1/32*sqrt(5.5)*(1-n)*(21*s^6-35*s^4+15*s^2-1)');
FN(25)=sym('1/32*sqrt(5.5)*(1-s)*(21*n^6-35*n^4+15*n^2-1)');
FN(26)=sym('1/32*sqrt(5.5)*(1+n)*(21*s^6-35*s^4+15*s^2-1)');
FN(27)=sym('1/32*sqrt(5.5)*(1+s)*(21*n^6-35*n^4+15*n^2-1)');
FN(31)=sym('1/32*sqrt(6.5)*(1-n)*(33*s^7-63*s^5+35*s^3-5*s)');
FN(32)=sym('1/32*sqrt(6.5)*(1-s)*(33*n^7-63*n^5+35*n^3-5*n)');
FN(33)=sym('1/32*sqrt(6.5)*(1+n)*(33*s^7-63*s^5+35*s^3-5*s)');
FN(34)=sym('1/32*sqrt(6.5)*(1+s)*(33*n^7-63*n^5+35*n^3-5*n)');

%Funciones de forma interna
FN(17)=sym('3/8*(s^2-1)*(n^2-1)');
FN(22)=sym('sqrt(15)/24*(s^3-3*s-2)*(n^2-1)');
FN(23)=sym('sqrt(15)/24*(n^3-3*n-2)*(s^2-1)');
FN(28)=sym('sqrt(21)/32*(5*s^4-6*s^2+1)*(n^2-1)');
FN(29)=sym('5/72*(s^3-3*s-2)*(n^3-3*n-2)');
FN(30)=sym('sqrt(21)/32*(s^2-1)*(5*n^4-6*n^2+1)');
FN(35)=sym('3/32*sqrt(3)*(7*s^5-10*s^3+3*s)*(n^2-1)');
FN(36)=sym('sqrt(35)/32*(5*s^4-6*s^2+1)*(n^3/3-n-2/3)');
FN(37)=sym('sqrt(35)/32*(5*n^4-6*n^2+1)*(s^2/3-s-2/3)');
FN(38)=sym('3/32*sqrt(3)*(7*n^5-10*n^3+3*n)*(s^2-1)');

%Calculo simbólico de las derivadas parciales de las funciones
% de forma
SDNs(1)=sym('-.25+.25*n');
SDNn(1)=sym('-.25+.25*s');
SDNs(2)=sym('.25-.25*n');
SDNn(2)=sym('-.25-.25*s');
SDNs(3)=sym('.25+.25*n');
SDNn(3)=sym('.25+.25*s');
SDNs(4)=sym('-.25-.25*n');
```

```
SDNn(4)=sym('.25-.25*s');
SDNs(5)=sym('0.25*sqrt(1.5)*(1-n)*2*s');
SDNn(5)=sym('-0.25*sqrt(1.5)*(s^2-1)');
SDNs(6)=sym('-0.25*sqrt(1.5)*(n^2-1)');
SDNn(6)=sym('0.25*sqrt(1.5)*(1-s)*2*n');
SDNs(7)=sym('0.25*sqrt(1.5)*(1+n)*2*s');
SDNn(7)=sym('0.25*sqrt(1.5)*(s^2-1)');
SDNs(8)=sym('0.25*sqrt(1.5)*(n^2-1)');
SDNn(8)=sym('0.25*sqrt(1.5)*(1+s)*2*n');
SDNs(9)=sym('0.25*sqrt(1.5)*(1-n)*(3*s^2-1)');
SDNn(9)=sym('-0.25*sqrt(1.5)*(1-s)*(3*n^2-1)');
SDNs(10)=sym('-0.25*sqrt(1.5)*(n^2-1)');
SDNn(10)=sym('0.25*sqrt(1.5)*(1-s)*2*n');
SDNs(11)=sym('0.25*sqrt(1.5)*(1+n)*2*s');
SDNn(11)=sym('0.25*sqrt(1.5)*(s^2-1)');
SDNs(12)=sym('0.25*sqrt(1.5)*(n^2-1)');
SDNn(12)=sym('0.25*sqrt(1.5)*(1+s)*2*n');
SDNs(13)=sym('1/16*sqrt(3.5)*(1-n)*(20*s^3-12*s)');
SDNn(13)=sym('-1/16*sqrt(3.5)*(5*s^4-6*s^2+1)');
SDNs(14)=sym('-1/16*sqrt(3.5)*(5*n^4-6*n^2+1)');
SDNn(14)=sym('1/16*sqrt(3.5)*(1-s)*(20*n^3-12*n)');
SDNs(15)=sym('1/16*sqrt(3.5)*(1+n)*(20*s^3-12*s)');
SDNn(15)=sym('1/16*sqrt(3.5)*(5*s^4-6*s^2+1)');
SDNs(16)=sym('1/16*sqrt(3.5)*(5*n^4-6*n^2+1)');
SDNn(16)=sym('1/16*sqrt(3.5)*(1+s)*(20*n^3-12*n)');
SDNs(17)=sym('3/4*s*(n^2-1)');
SDNn(17)=sym('3/4*n*(s^2-1)');
SDNs(18)=sym('1/16*sqrt(4.5)*(1-n)*(35*s^4-30*s^2+3)');
SDNn(18)=sym('-1/16*sqrt(4.5)*(7*s^5-10*s^3+3*s)');
SDNs(19)=sym('-1/16*sqrt(4.5)*(7*n^5-10*n^3+3*n)');
SDNn(19)=sym('1/16*sqrt(4.5)*(1-s)*(35*n^4-30*n^2+3)');
SDNs(20)=sym('1/16*sqrt(4.5)*(1+n)*(35*s^4-30*s^2+3)');
SDNn(20)=sym('1/16*sqrt(4.5)*(7*s^5-10*s^3+3*s)');
SDNs(21)=sym('1/16*sqrt(4.5)*(7*n^5-10*n^3+3*n)');
SDNn(21)=sym('1/16*sqrt(4.5)*(1+s)*(35*n^4-30*n^2+3)');
SDNs(22)=sym('sqrt(15)/8*(s^2-1)*(n^2-1)');
SDNn(22)=sym('sqrt(15)/12*n*(s^3-3*s-2)');
SDNs(23)=sym('sqrt(15)/12*s*(n^3-3*n-2)');
SDNn(23)=sym('sqrt(15)/8*(n^2-1)*(s^2-1)');
SDNs(24)=sym('1/32*sqrt(5.5)*(1-n)*(126*s^5-140*s^3+30*s)');
SDNn(24)=sym('-1/32*sqrt(5.5)*(21*s^6-35*s^4+15*s^2-1)');
SDNs(25)=sym('-1/32*sqrt(5.5)*(21*n^6-35*n^4+15*n^2-1)');
SDNn(25)=sym('1/32*sqrt(5.5)*(1-s)*(126*n^5-140*n^3+30*n)');
```

```
SDNs(26)=sym('1/32*sqrt(5.5)*(1+n)*(126*s^5-140*s^3+30*s)');
SDNn(26)=sym('1/32*sqrt(5.5)*(21*s^6-35*s^4+15*s^2-1)');
SDNs(27)=sym('1/32*sqrt(5.5)*(21*n^6-35*n^4+15*n^2-1)');
SDNn(27)=sym('1/32*sqrt(5.5)*(1+s)*(126*n^5-140*n^3+30*n)');
SDNs(28)=sym('sqrt(21)/32*(20*s^3-12*s)*(n^2-1)');
SDNn(28)=sym('sqrt(21)/16*n*(s*s^4-6*s^2+1)');
SDNs(29)=sym('5/24*(s^2-1)*(n^3-3*n-2)');
SDNn(29)=sym('5/24*(s^3-3*s-2)*(n-1)');
SDNs(30)=sym('sqrt(21)/16*s*(5*n^4-6*n^2+1)');
SDNn(30)=sym('sqrt(21)/32*(s^2-1)*(20*n^3-12*n)');
SDNs(31)=sym('1/32*sqrt(6.5)*(1-n)*(231*s^6-315*s^4+105*s^2-5)');
SDNn(31)=sym('-1/32*sqrt(6.5)*(33*s^7-63*s^5+35*s^3-5*s)');
SDNs(32)=sym('-1/32*sqrt(6.5)*(33*n^7-63*n^5+35*n^3-5*n)');
SDNn(32)=sym('1/32*sqrt(6.5)*(1-s)*(231*n^6-315*n^4+105*n^2-5)');
SDNs(33)=sym('1/32*sqrt(6.5)*(1+n)*(231*s^6-315*s^4+105*s^2-5)');
SDNn(33)=sym('1/32*sqrt(6.5)*(33*s^7-63*s^5+35*s^3-5*s)');
SDNs(34)=sym('1/32*sqrt(6.5)*(33*n^7-63*n^5+35*n^3-5*n)');
SDNn(34)=sym('1/32*sqrt(6.5)*(1+s)*(231*n^6-315*n^4+105*n^2-5)');
SDNs(35)=sym('3/32*sqrt(3)*(35*s^4-30*s^2+3)*(n^2-1)');
SDNn(35)=sym('3/16*sqrt(3)*(7*s^5-10*s^3+3*s)');
SDNs(36)=sym('sqrt(35)/32*(20*s^3-12*s)*(n^3/3-n-2/3)');
SDNn(36)=sym('sqrt(35)/32*(2*n/3-1)*(5*s^4-6*s^2+1)');
SDNs(37)=sym('sqrt(35)/32*(2*s/3-1)*(5*n^4-6*n^2+1)');
SDNn(37)=sym('sqrt(35)/32*(20*n^3-12*n)*(s^3/3-s-2/3)');
SDNs(38)=sym('3/16*sqrt(3)*(7*n^5-10*n^3+3*n)');
SDNn(38)=sym('3/32*sqrt(3)*(35*n^4-30*n^2+3)*(s^2-1)');

%Evaluación de las funciones de forma NI y sus derivadas en los
% puntos de Gauss,
%como vectores filas
CNG=0;
for intr=1:NG
    s=p(intr);
    for intz=1:NG
        n=p(intz);
        CNG=CNG+1;
        for nfn=1:N
            NI(nfn,CNG)=eval(FN(nfn));
            DNs(nfn,CNG)=eval(SDNs(nfn));
            DNn(nfn,CNG)=eval(SDNn(nfn));
        end
    end
end
```

```
clear CNG
```

Archivo: poly-itg

```
%poly_itg.m

%-------------------------------------------------------
%  Propósito:
% Integración de un polinomio.
% Encontrar los coeficientes del polinomio integrado.
%
%  Sintaxis:
% py=poly_itg(po)
%
%  Descripción de variables:
% po=coeficientes del polinomio ha integrar
% py=coeficientes del polinomio ya integrado
%-------------------------------------------------------

function py=poly_itg(po)
n=length(po);
py=[po.*[n:-1:1].^(-1),0];
```

Archivo: sinter

```
%sinter.m

%-------------------------------------------------------
%  Propósito:
% Determinar coordenadas sinterizadas.
%
%  Sintaxis:
% Es llamado directamente dentro del programa principal.
%
%  Descripción de variables:
% Ib: Vector indicador de si las coordenadas han sido movidas al nuevo
% punto sinterizado. Dim(nnodos).
% BN: Primer valor del coeficiente de contracción.
% e: Contador del número de elementos.
% m: Contador del número de iteracciones.
% B: Vector de coeficientes de contracción. Dim(nelem).
% tol_B: Tolerancia para la iteracción de Beta.
% crs y czs: coordenadas movidas.
```

```
% rz: Martriz de coordenadas en crudo. Dim(nnodos,2).
% I: El valor de la integral numerica.
% fb1,fb2,fbd: Funcion evaluada en ptos 1,2; y su derivada para
%  Newton Rapson.
% rzs: Matriz de coordenadas sinterizadas. Dim(nnodos,2).
%-------------------------------------------------------

%Calculo de las coordenadas sinterizadas.
dh=1;
if dh==1
   x=1;
else
   x=nelem;
end

Ib=zeros(nnodos,1);
B=ones(nelem,1);
BN=ones(nelem,1);
B=Bsup*B;

for e=1:x
   m=0;
   error=tol_B+1;
   for l=1:nne
      if Ib(conec(e,l),1)==0;
         crs(l,1)=B(e)*rz(conec(e,l),1);
         czs(l,1)=B(e)*rz(conec(e,l),2);
      else
         crs(l,1)=rz(conec(e,l),1);
         czs(l,1)=rz(conec(e,l),2);
      end
   end
   I=integrar(crs',czs',tol,it);
   fb1=mc(e)-ds*I;
   while abs(error)>tol_B
      m=m+1;
      if m>it break; end
      for l=1:nne
         if Ib(conec(e,l),1)==0;
            crs(l,1)=(B(e)+delta_B)*rz(conec(e,l),1);
            czs(l,1)=(B(e)+delta_B)*rz(conec(e,l),2);
         else
            crs(l,1)=rz(conec(e,l),1);
```

```
            czs(l,1)=rz(conec(e,l),2);
         end
      end
      I=integrar(crs',czs',tol,it);
      fb2=mc(e)-ds*I;
      fbd=(fb2-fb1)/delta_B;
      BN=B(e)-fb1/fbd;
      error=BN(e)-B(e);
      B(e)=BN(e);
      fb1=fb2;
   end
   for l=1:nne
      if Ib(conec(e,l),1)==0;
         rzs(conec(e,l),1)=crs(l,1);
         rzs(conec(e,l),2)=czs(l,1);
         Ib(conec(e,l),1)=1;
      end
   end
end

rzs=B(1)*rz;
rz_sinter=rzs;

%Guarda coordenadas sinterizadas
save rz_sinter rzs -ascii

clear l e
```

Archivo: sol-gmres

```
%sol_gmres.m
function [a,flag,coef_pena,residual,iteracciones,norma]=sol_
        gmres(b,nnodos,rest,glt)

%-------------------------------------------------------------------
%  Propósito:
% Aplica las restricciones, penaliza la matriz y calcula la solucion
% del sistema mediante el metodo generalizado de residuos minimos
%
%  Sintaxis:
% function [a,flag,coef_pena,residual,iteracciones,norma]=sol_gmres
        %(b,nnodos,rest,glt).
%
```

```
%  Descripción de variables:
% a: Desplazamiento.
%       b: Vector de fuerzas.
% coef_pena: factor de penalización.
%       flag: indicador de la converencia.
%       glt: Cantidad de grados de libertad del sistema
%       iteracciones: Numero maximo de ietarcciones
% nnodos: # total de nodos del sistema.
%       norma: Norma de la ultima ireaccion.
%       residual: residual de la iteraccion
% rest: Vector de restricciones.
%-------------------------------------------------------------------
global kijg

%factor de penalización
coef_pena=full(max(max(kijg)))*1e4;

for i=1:2*nnodos
   if rest(i)==1
      kijg(i,i)=coef_pena+kijg(i,i); %Asigna un valor grande al
       elemento
      %kii que se
      %corresponde con el grado de libertad restringido
   end
end

%Llama la funcion para gmres [Metodo de los residuales minimos]
[a,flag,residual,iteracciones,norma]=gmres(b,glt/2,1e-1,glt/2);
%El 3er parametro es la Tolerancia convergencia.
```

Archivo: vector-Fz

```
%vector_Fz.m

%-------------------------------------------------------------------
%  Propósito:
% Calcular el vector de fuerzas másicas de cada elemento.
%
%  Sintaxis:
% Es llamado directamente dentro del programa mecanico.m
%
%  Descripción de variables:
% peso: Vector de pesos de los elementos. Dim(nelem).
```

```
% e: contador de los elementos.
% r,z: Coordenadas de los nodos.
% Fz: Vector de fuerzas de gravedad. Dim(2*N).
%
%-----------------------------------------------------------------

%Calcula los puntos y pesos de Gauss a ser utilizados y los almacena en
    %los vectores p,w
[p,w]=leg_pesos(NG);

%Abrir archivo temporal "Fz" para almacenar los vectores de carga
fid=fopen('Fz','w');

%Abrir archivo temporal "peso" para lectura de cargas gravitacionales
fid1=fopen('peso','r');
peso=fread(fid1,[nelem,1],'real*8')
fclose(fid1);


%Calculo de los qai y qbi de cada elemento.
for e=1:nelem
   %Extraer coordenadas del elemento
   for l=1:nne
      r(l)=rzs(conec(e,l),1);
      z(l)=rzs(conec(e,l),2);
   end
   Fz=zeros(2*N,1); %Inicializa el Vector de fuerzas másicas.

   for intr=1:NG
      s=p(intr);
      for intz=1:NG
         n=p(intz);
         rt=0.25*[(1-s)*(1-n) (1+s)*(1-n) (1+s)*(1+n) (1-s)*(1-n)]*r';
         zt=0.25*[(1-s)*(1-n) (1+s)*(1-n) (1+s)*(1+n) (1-s)*(1-n)]*z';
         JDNs=[-0.25*(1-n) 0.25*(1-n) 0.25*(1+n) -0.25*(1+n)];
         JDNn=[-0.25*(1-s) -0.25*(1+s) 0.25*(1+s) 0.25*(1-s)];
         J=[JDNs*r' JDNs*z'; JDNn*r' JDNn*z'];
         [NI,DNs,DNn]=f_form(gp,N,n,s,w); %Llama funcion para N y N'
         NI=[zeros(N,1);NI'];
         Fz=Fz+w(intr)*w(intz)*peso(e)*NI*det(J)*rt;
      end
   end
   fwrite(fid,Fz,'real*8'); %Guarda los vectores de fuerza
```

```
end

%cerrar archivo temporal
fclose(fid);

%Elimina variables locales
clear fid e intr intz
```

Archivo: vector-qa

```
%vector_qa.m  Version Oct 18/04

%-------------------------------------------------------------------
%  Propósito:
% Calcular el vector de fuerzas másicas de cada elemento y ensamblar
%       el vector global de fuerzas.
%
%  Sintaxis:
% Es llamado directamente dentro del programa mecanico_theta.m
%
%  Descripción de variables:
%       Fzg: Vector global de fuerzas masicas. Dim(glt,1).
%       e: Contador del numero de elementos.
% Fz: Vector de fuerzas de gravedad. Dim(2*N).
%       CNG: Contador del numero de puntos de Gauss totales.
%       intr, intz: Contadores individuales de puntos de Gauss.
%       DJ: Determinante de la matriz de trasnformacion.
% i: Contador del numero de nodos por elemento.
%       c: Indicador de la conectividad.
%-------------------------------------------------------------------

%Abrir archivo donde estan los Jinv
fid=fopen('DJ','r');

%Inicializa el vector global de fuerzas.
Fzg=zeros(glt,1);

%Calculo de los qai de cada elemento.
for e=1:nelem
    Fz=zeros(N,1); %Inicializa el Vector de fuerzas másicas.
    CNG=0;
    for intr=1:NG
        for intz=1:NG
```

```
            CNG=CNG+1;
            DJ=fread(fid,[1,1],'real*8');
            Fz=Fz+w(intr)*w(intz)*NI(:,CNG)*DJ*rt(CNG,e);
        end
    end

    %Ensamble de la matriz global
    for i=1:nne
        c=conec(e,i); %Extrae numeración global del nodo
        Fzg(nggl(c,2),1)=Fzg(nggl(c,2),1)+Fz(i,1);
    end
end
%Aplicacion de fuerza de gravedad y la densidad del material
Fzg=-9.8*ds*Fzg;
%cerrar archivo temporal
fclose(fid);

%Elimina variables locales
clear fid i e c intr intz CNG DJ
```

Archivo: vector-qc

```
%vector_qc.m  Version Oct 18/04
%
%-------------------------------------------------------------------
%  Propósito:
% Calcular el vector de cargas asociadas a la deformacion Creep y
%       ensamblar el el vector global.
%
%  Sintaxis:
% Es llamado directamente dentro del programa mecanico_theta.m
%
%  Descripción de variables:
%       qcg: Vector global de cargas creep. Dim(glt,1).
%       C: Contador del numero total de puntos de Gauss usados.
%       e: Contador del numero de elementos.
%       CNG: Contador del numero de puntos de Gauss por elemento.
%       intr, intz: Contadores individuales de puntos de Gauss.
%       DJ: Determinante de la matriz de trasnformacion.
% i: Contador del numero de nodos por elemento.
%       cn: Contador del numero de nodos por elemento.
%       c: Indicador de la conectividad.
% e: contador de los elementos.
```

```
% r,z: Coordenadas de los nodos.
%
%--------------------------------------------------------------------
%Abrir archivo temporales donde esta el |J| y las matrices inversas
%de la transformacion de coordenadas
fid1=fopen('DJ','r');
fid2=fopen('Jinv','r');

%Inicializa el vector de cargas creep.
qcg=zeros(glt,1);


%Calculo de los qci de cada elemento.
C=0;
for e=1:nelem
    qc=zeros(2*N,1); %Inicializa el vector de carga debido a la
    deformacion
    %inicial Ec.
    CNG=0;
    for intr=1:NG
      for intz=1:NG
         CNG=CNG+1;
         C=C+1;
         DJ=fread(fid1,[1,1],'real*8');
         Jinv=fread(fid2,[2,2],'real*8');
         [mtxcin]=f_matriz_cinematica(N,rt(CNG,e),Jinv,CNG);
         qc=qc+w(intr)*w(intz)*mtxcin'*D*delta_Ec(:,C)*DJ*rt(CNG,e);
      end
   end

   %Ensamble del vector global.
   for cn=1:nne
       c=conec(e,cn); %Extrae numeración global del nodo
       ngle(cn)=nggl(c,1); %Extrae numeración global de grados de
       libertad
       ngle(cn+nne)=nggl(c,2); %Extrae numeración global de grados de
       %libertad
   end
   for i=1:2*N
       qcg(ngle(i),1)=qcg(ngle(i),1)+qc(i,1);
   end
end
```

```
%cerrar archivo temporal
fclose(fid1);
fclose(fid2);

%Elimina variables locales
clear fid1 fid2 intr intz e l CNG C i cn
```

Archivo: vel-creep

```
%function Vel_Ec

%-----------------------------------------------------------------
%  Propósito:
%     Determinar la velocidad de deformación tipo creep.
%
%  Sintaxis:
%     [Vel_Ec]=vel_creep(nelem,A,tension).
%
%  Variable Description:
%     Vel_Ec: Matriz de velocidad de deformación creep. Dim(4,nelem).
% nelem: Número de elementos del sistema
% A: Constante 1 del modelo real de Norton.
%       AA: Constante 2 del modelo real de Norton.
% tension: Vector de esfuerzos resultantes. Dim(nelem,nelem).
% Calculados en el centroide de cada elemento.
% tension_media: Tensión media. [N/m2].
% e: Contador del número de elementos.
% sij: Matriz de los esfuerzo "Deviatoric". Dim(4,nelem).
% tension_eff: Vector de esfuerzo efectivos calculados por la ley
%  multi-axial. Dim(e)
%-----------------------------------------------------------------

function [Vel_Ec,tension_eff]=vel_creep(nelem,A,AA,tension,NG)

%Inicialización de vectores y matrices.
CNG=nelem*NG*NG;
tension_media=zeros(CNG,1);
Vel_Ec=zeros(4,CNG);

%Calculo de la tensión media y los esfuerzos "Deviatoric"
for e=1:CNG
   tension_media(e,1)=tension_media(e,1)+(tension(1,e)+tension(2,e)+
   tension(3,e))/3;
```

```
   sij(1,e)=tension(1,e)-tension_media(e,1);
   sij(2,e)=tension(2,e)-tension_media(e,1);
   sij(3,e)=tension(3,e)-tension_media(e,1);
   sij(4,e)=tension(4,e);

   %Calculo de la tensión efectiva
   tension_eff(e,1)=(1/sqrt(2))*sqrt((tension(1,e)-tension(2,e))^2+...
      (tension(1,e)-tension(3,e))^2+(tension(2,e)-tension(3,e))^2+6*
        (tension(4,e))^2);

   %Calculo de la velocidad de deformacion creep.
   Vel_Ec(:,e)=Vel_Ec(:,e)+3/2*(A*(tension_eff(e,1)+AA))*sij(:,e);
end
```

APÉNDICE F

Datos

Archivo: Elementos.txt

```
20
4
  Elem    Eg  Mp    Rc  Ec                    NODES
     1     1   1     1  -1          1 2 3 4
     2     1   1     1  -1         2 5 6 3
     3     1   1     1  -1         5 7 8 6
     4     1   1     1  -1          7 9 10 8
     5     1   1     1  -1        9 11 12 10
     6     1   1     1  -1         12 13 14 15
     7     1   1     1  -1        11 16 13 12
     8     1   1     1  -1         17 18 15 14
     9     1   1     1  -1         19 20 17 14
    10     1   1     1  -1         21 23 19 14
    11     1   1     1  -1         16 21 14 13
    12     1   1     1  -1         18 22 26 15
    13     1   1     1  -1        22 24 25 26
    14     1   1     1  -1        24 27 28 25
    15     1   1     1  -1        27 29 30 28
    16     1   1     1  -1        29 31 32 30
    17     1   1     1  -1        31 33 34 32
    18     1   1     1  -1        33 35 36 34
    19     1   1     1  -1        35 37 38 36
    20     1   1     1  -1       37 39 40 38
```

Archivo: Nodos.txt

40

Node	X-Coordinate	Y-Coordinate	Z-Coordinate
1	0.000000e+000	0.000000e+000	0.000000e+000
2	1.422435e+001	-5.620636e-001	0.000000e+000
3	1.307353e+001	4.662071e+000	0.000000e+000
4	0.000000e+000	4.995181e+000	0.000000e+000
5	2.298828e+001	-1.062500e+000	0.000000e+000
6	2.394559e+001	3.958527e+000	0.000000e+000
7	4.985254e+001	-3.022598e+000	0.000000e+000
8	4.691701e+001	1.948014e+000	0.000000e+000
9	6.436331e+001	-4.480865e+000	0.000000e+000
10	6.988843e+001	-6.250000e-002	0.000000e+000
11	7.574254e+001	-6.716227e+000	0.000000e+000
12	8.345032e+001	-1.328125e+000	0.000000e+000
13	8.442214e+001	-3.629352e+000	0.000000e+000
14	8.900513e+001	-5.492204e+000	0.000000e+000
15	9.701221e+001	-2.593750e+000	0.000000e+000
16	7.968874e+001	-8.127122e+000	0.000000e+000
17	9.484008e+001	-1.223046e+001	0.000000e+000
18	9.831043e+001	-1.102599e+001	0.000000e+000
19	9.009955e+001	-1.571982e+001	0.000000e+000
20	9.207818e+001	-1.465243e+001	0.000000e+000
21	8.489786e+001	-1.166907e+001	0.000000e+000
22	1.035041e+002	-1.003482e+001	0.000000e+000
23	8.841653e+001	-1.510297e+001	0.000000e+000
24	1.085805e+002	-8.556061e+000	0.000000e+000
25	1.058971e+002	-2.056110e+000	0.000000e+000
26	1.014838e+002	-2.805985e+000	0.000000e+000
27	1.149482e+002	-5.500366e+000	0.000000e+000
28	1.088820e+002	-7.613120e-001	0.000000e+000
29	1.223338e+002	-2.121448e+000	0.000000e+000
30	1.170166e+002	2.630779e+000	0.000000e+000
31	1.300867e+002	2.985990e-001	0.000000e+000
32	1.272644e+002	4.952521e+000	0.000000e+000
33	1.370086e+002	2.246079e+000	0.000000e+000
34	1.375122e+002	7.274261e+000	0.000000e+000
35	1.439834e+002	3.994770e+000	0.000000e+000
36	1.445204e+002	8.856607e+000	0.000000e+000
37	1.510452e+002	3.758605e+000	0.000000e+000
38	1.515287e+002	1.043895e+001	0.000000e+000
39	1.547458e+002	4.906196e+000	0.000000e+000

```
40          1.550254e+002          8.770492e+000          0.000000e+000
```

Archivo: Parámetros.txt

```
%Parametros_matematicos
0.0001
100
0.8
0.000001
0.0001
9000
1000
3
```

Archivo: Propiedades.txt

```
%Propiedades_mecánicas_del_material:
_Módulo_Young_[Pa]_y_Relación_de_Poisson_Constantes_modelo_creep
8e9
0.49
1.80587e-15
4.45313e-11
%Densidades_en_crudo_y_Sinterizada_en_kg/m3_y_perdidas
_por_ignicion_indice
1700
2400
0.92
```

[?????@@]