Indicaciones generales para entender y ampliar las capacidades de los programas contenidos en 2dBVP

En este documento se explican aspectos básicos para comprender y utilizar el programa 2dBVP para la resolución de la ecuación de Poisson en 2D mediante elementos finitos triangulares y cuadriláteros. Consta de 3 bloques: I) Explicaciones generales sobre el programa. II) Estructura de sus módulos. III) Programa de ejemplo.

El programa básico, desarrollado por Nicholas Zabaras (School Mechanical&Aerospace Eng, Univ. de Cornell), ha sido complementado por Jaime Puig-Pey (ETS Ing. Caminos, Univ. de Cantabria) en lo relativo: I) refine progresivo de malla de elementos finitos sobre el dominio: a) bordes poligonales arbitrarios, b) bordes tratables en coordenadas polares; II) cálculo y visualización de evolución de resultados a lo largo del refine, lo que permite expresar el proceso de convergencia al refinar la malla. En buena parte de los nombres de los módulos adaptados se ha añadido la terminación _refine. El nombre del archivo include_variables.m, para definir variables globales en el programa, no se ha modificado, pero se han añadido algunas instrucciones para definir variables utilizadas en el proceso de refine sucesivo. Se han incorporado funciones nuevas para gestionar mallados (InputIniGeometria.m, FunDivCuadBordes.m, FunDivTriangBordes.m, fun_Cuad_Triangulo.m, FunGeomPolar.m,...). Es conveniente leer los comentarios e instrucciones presentes en los programas, para su adecuada comprensión.

El programa principal que gestiona la ejecución global es main2D_refine.m y en él se gestiona la repetición de refines y evolución de resultados al refinar, referidos a valores solución en los nodos de la malla inicial definida por el usuario. Se ha optado utilizar carpetas diferentes, cada una con todos los módulos de programa necesarios para cada ejemplo de aplicación. Buena parte de los módulos utilizados en los diversos ejemplos son los mismos, pero con esa separación en carpetas, se evita introducir en un mismo módulo excesivos casos de datos introducidos por el usuario para ejemplos diversos.

El programa puede ejecutarse también en el entorno Octave, de dominio público.

I) Explicaciones generales sobre el programa

A: Especificación de condiciones de contorno. Entrada de datos

Un paso importante para utilizar este código es especificar las condiciones de contorno apropiadas. Para definir el contorno, se asigna un valor marcador (un número) a cada uno de los distintos bordes, que se caracteriza por tener unas mismas condiciones de contorno.

Si no se sabe el marcador de un borde, por ejemplo cuando se carga un mallado de EF del programa ANSYS, puede interesar dibujar primero el mallado , como ayuda para especificar las condiciones de borde, lo que se hace asignando a la variable carácter 'plot_boundary' el valor 'yes'.

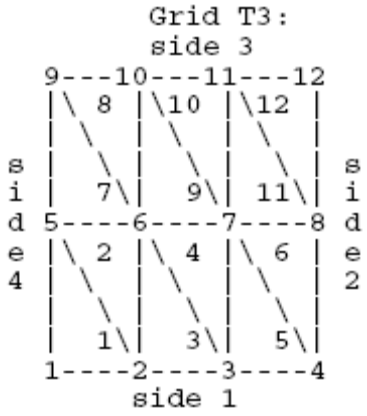
Hay una variable global: **IsBoundaryCondition**. Es un vector fila que indica si hay alguna condición de contorno en cada borde. Tiene un número de componentes igual al número de bordes. Cada componente puede tomar 2 valores:

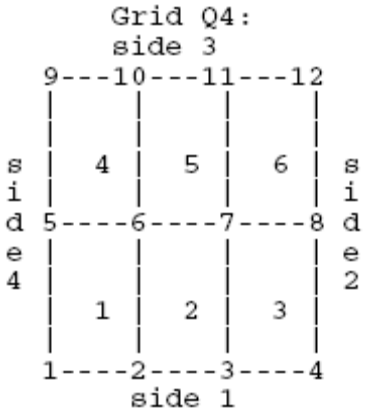
0: equivale a la condición de FLUJO NULO. Dejando en cero la componente implica que se aplica flujo nulo en ese borde.

1: Indica que en ese borde ocurre una condición ESENCIAL de contorno, nula o no, o bien una condición NATURAL no nula.

- El vector IsBoundaryCondition se especifica en la función InputData

Por ejemplo, si la malla es una caja, y las condiciones de contorno son distintas de borde a borde, se tienen 4 bordes. La longitud del vector IsBoundaryCondition es 4. La malla en caja tiene marcadores de contorno por defecto como sigue:





Si se especifica: IsBoundaryCondition=[1 1 1 1]; que tiene los siguientes significados:

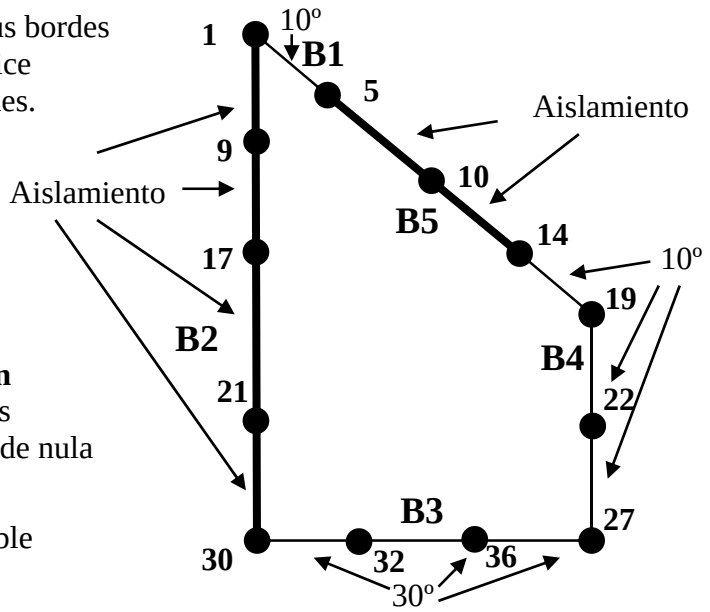
. Todos los bordes tienen condiciones de borde esenciales (nulas o no) o naturales no nulas.

. Si el valor de una condición natural de borde (flujo) es cero, se debe asignar en la correspondiente componente de IsBoundaryCondition el valor 0. Por ejemplo:

```
IsBoundaryCondition = [0 1 0 0 ];
```

significa que las bordes (sides) 1, 3 y 4 tienen condición natural de borde nula. El segundo borde tiene condición de borde de flujo no nulo o bien condición esencial de borde.

Considérese la figura de un dominio con sus bordes marcados. Los nodos indicados por su índice son los únicos que se ubican sobre los bordes.



Hay 5 bordes:
B1: nodos 5-1
B2: nodos 1-9-17-21-30
B3: nodos 30-32-36-27
B4: nodos 27-22-19-14
B5: nodos 14-10-5
El vector de bordes **IsBoundaryCondition** será en este ejemplo: [1,0,1,1,0], ya que los únicos bordes con condición natural de borde nula (aislamiento) son el **B2** y el **B5**.

Los nodos de borde se indican con la variable tipo 'estructura' **BoundaryNodes.Nodes**

En este ejemplo, esta variable indicando los nodos en cada borde deberá tomar los valores:

```
BoundaryNodes(1).Nodes=[1,5];
     BoundaryNodes(2).Nodes=[1,9,17,21,30];
BoundaryNodes(3).Nodes=[30,32,36,27];
     BoundaryNodes(4).Nodes=[27,14,19,22];
BoundaryNodes(5).Nodes=[14,10,5];
```

% NO IMPORTA EL ORDEN DE LOS NODOS EN CADA BORDE NI LA ORDENACION DE LOS BORDES

En la función **InputIniGeometria** se especifican:

* Los nodos, por sus coordenadas (x,y), en la variable **Nodes**, matriz de 2 filas.
El nodo de índice i se localiza en la columna i de Nodes. En la primera fila de Nodes están las coordenadas x de los nodos y en la segunda las coordenadas y.

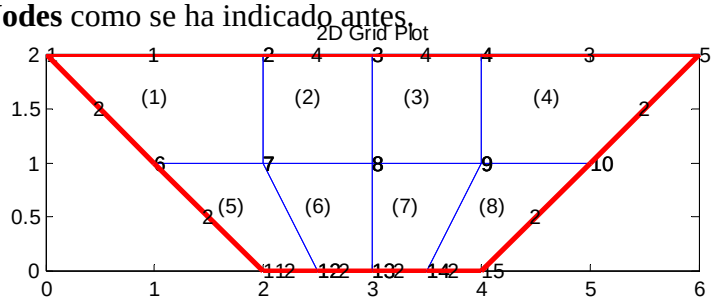
* Los elementos finitos, pueden ser triángulos o cuadriláteros (un solo tipo para una discretización dada), y no necesariamente regulares. Se elige malla triangular o cuadrilátera en **InputGrid_refine**. Los cuadriláteros deben tener todos sus ángulos interiores no mayores que 180 grados sexagesimales.

Se ha de procurar que tanto triángulos como cuadriláteros tengan los ángulos interiores ni muy pequeños ni muy grandes, para reducir problemas numéricos.

Los elementos se definen dando los índices de los nodos que configuran sus vértices, recorriendo el contorno del elemento EN ORDEN ANTIHORARIO, desde cualquiera de sus vértices.
Se almacenan en la matriz **Elems**, de tantas filas como vértices (3, triángulo; 4, cuadrilátero) y tantas columnas como elementos. El elemento k está en la columna k de Elems.

* Los bordes en las variable **BoundaryNodes.Nodes** como se ha indicado antes.

Ejemplo:



En el dominio inicial con forma de trapecio de la figura, hay 4 bordes,
con 8 elementos cuadriláteros y 15 nodos.

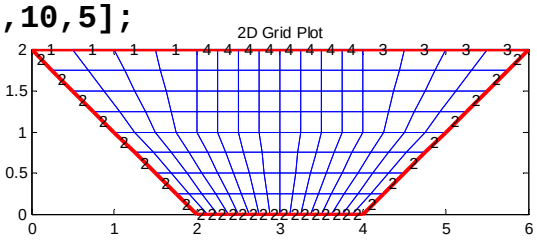
En la función InputData se asigna:

```
Is BundaryCondition=[1,0,1,0]
```

pues los bordes B1 y B3 no tienen condición de flujo normal nulo, y los B2 y B3 sí.

En la función InputIniGeometria se asignan:

```
Nodes(1,1:5)=[0,2:4,6];Nodes(2,1:5)=2*ones(1,5);
Nodes(1,6:10)=1:5;Nodes(2,6:10)=1*ones(1,5);
Nodes(1,11:15)=2:0.5:4;Nodes(2,11:15)=zeros(1,5);
Elems=[1,6,7,2;2,7,8,3;3,8,9,4;4,9,10,5;...
       6,11,12,7;7,12,13,8;8,13,14,9;9,14,15,10];
Elems=Elems'; % cada col índices de vértices de un elemento
BoundaryNodes(1).Nodes=[1,2];
BoundaryNodes(2).Nodes=[1,6,11,12,13,14,15,10,5];
BoundaryNodes(3).Nodes=[5,4];
BoundaryNodes(4).Nodes=[4,3,2];
```



El programa adaptado permite refinar sucesivamente
la malla inicial, realizando ejecuciones consecutivas con los mallados refinados . Obsérvese la malla cuadrilátera refinada mediante 2 refines.

En el programa principal **main2D_refine** se asigna a la variable nrefine el número de refines. Si se asigna el valor cero a nrefine, se ejecuta una sola vez con el mallado inicial. Para borde de dominio poligonal, la numeración de los nodos no se modifica a lo largo de los sucesivos refines, sino que se van añadiendo los nuevos nodos en el proceso. Al final de la ejecución, se escribe una tabla con la evolución de los valores de la variable escalar incógnita en los nodos del mallado inicial a lo largo de los sucesivos refines, para mostrar la convergencia, en su caso, de la solución numérica.

En cada refine, de cada cuadrilátero o triángulo surgen otros cuatro, uniendo los puntos medios de los lados de un elemento triangular, y los puntos medios de lados opuestos de uno cuadrilátero.

Es frecuente que para encajar una mallado con elementos cuadriláteros, sea conveniente introducir algunos elementos de geometría triangular, para ajustar con los bordes poligonales.
Se pueden utilizar cuadriláteros que podríamos llamar ‘límites’ o ‘degenerados’, actuando con la ubicación de los nodos.

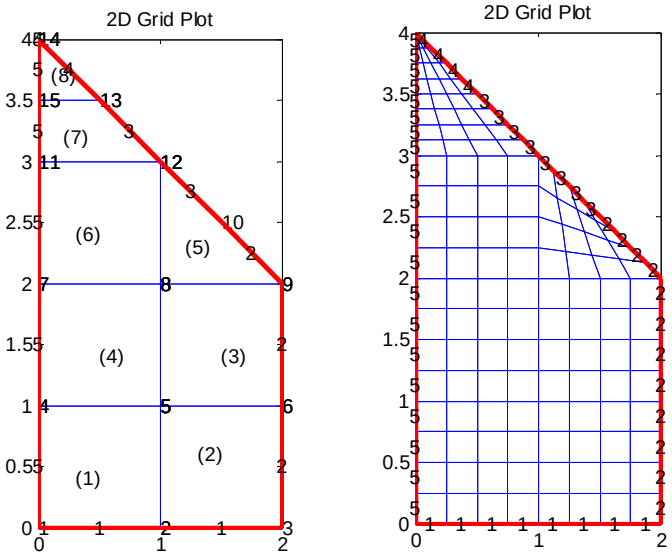
En la figura del dominio con forma de trapecio recto que sigue, se representa media sección de una casa con tejado, y elementos cuadriláteros en que se observa lo siguiente:

. El elemento 8 tiene forma triangular:
nodos 15, 13, 14.

¿Cómo encajarlo en la estructura cuadrilátera?

Simplemente tomando,
en el elemento 8 correspondiente
el nodo 14 con referencia duplicada

(ver 8ª columna de la matriz Elems).



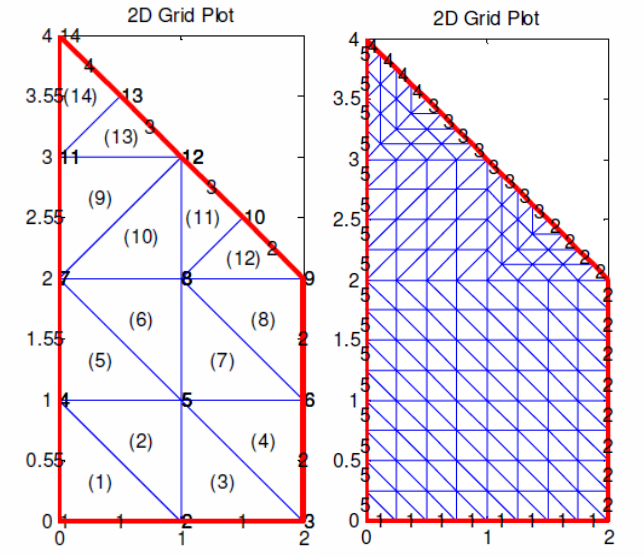

```
Elems=[1,2,5,4;5,2,3,6;...
6,9,8,5;5,8,7,4;8,9,10,12;...
 7,8,12,11;11,12,13,15;...
 13,14,14,15];% hace coincidir nodo 14 consigo mismo
```

A su vez, el elemento 5 tiene forma triangular, nodos 5,9,10,12. ¿Cómo encajar en la estructura cuadrilátera?

Simplemente tomando el nodo 10 sobre el lado 9-12, aquí se ha hecho en el punto medio (ver 5ª columna de la matriz Elems). Obsérvese en las anteriores figuras el estado del mallado 'cuadrilátero' tras 2 refines de la malla inicial

No es deseable introducir estas situaciones límite con mallados cuadriláteros, de introducir un nodo duplicado o introducir un nodo en un punto intermedio de un lado. Si se hacen sucesivos refines del mallado de elementos finitos, la forma de los sucesivos elementos refinados, triángulos con nodo doble o cuadriláteros con un nodo en posición intermedia de un lado, van adquiriendo una forma menos 'regular', lo que puede tener une incidencia negativa en la estabilidad del proceso numérico. De todos modos, se puede observar en varios ejemplos que el proceso numérico a los niveles de refine realizados, no presenta alteraciones destacables.

Por otra parte, con mallas triangulares,
se muestran los mallados inicial y
tras dos refines de la malla inicial.



Después de especificar si hay alguna condición de borde en el vector IsBoundaryCondition, hay que modificar las dos siguientes funciones:

`specifyEssBCValue.m`: define el valor de las condiciones de borde ESENCIALES.
`specifyNaturalBCValue.m`: define el valor de las condiciones NATURALES de borde.

1. Para Especificar ls condiciones ESENCIALES de contorno:

Hay que asignar valores a dos vectores globales del problema: debc y ebcVals.
% **debc** -- vector de 'degrees of freedom', grados de libertad con 'essentialboundary conditions' condiciones esenciales de contorno. En el caso de una sola variable escalar incógnita (temperatura, potencial), el grado de libertad coincide con el índice del correspondiente nodo de la malla.
% **ebcVals** -- vector con los valores de las condiciones esenciales de contorno.

Como puede haber muchos nodos con condiciones de borde esenciales, para evitar errores se deja al programa que defina automáticamente estos dos vectores.

La idea básica es como sigue:

```
for i = 1:length(IsBoundaryCondition) % se recorren los bordes
    if ( IsBoundaryCondition(i) == 1)
        call function 'fillEssBC'  % recorriendo todos los nodos sobre ese borde i;
    end
end
```

El programa comprueba en primer lugar los valores en el vector IsBoundaryCondition.

Si el valor indica que el borde tiene condición de borde, entonces el ciclo recorrerá todos los nodos de ese borde. Para cada nodo, llama a la siguiente función:

```
function [dof, vals] = specifyEssBCValue(nodeID, sideInd)
```

En ella hay dos argumentos de entrada: **nodeID** -- número global de nodo
sideInd -- número de borde, indicando en qué borde se encuentra este nodo

La razón para tener nodeID es que las condiciones ESENCIALES de borde pueden depender de las coordenadas. Si tenemos el número global del nodo, podemos recoger las coordenadas de la matriz Nodes.

La razón para tener la variable sideInd es que puede haber varios bordes que tienen condiciones esenciales de borde. Si se tiene un número de borde, se pueden separar o distinguir los diversos bordes.

PARA EL PROGRAMA PARA EL CAMPO ESCALAR (por ej. CONDUCCION DEL CALOR), las variables **dof** (Degree Of Fredom= Grado de Libertad) y **vals** son VARIABLES ESCALARES. Sin embargo, se ha hecho una programación un poco más general, de modo que se puedan utilizar la mayoría de las funciones en problemas con múltiples grados de libertad por nodo (por e. desplazamientos en las direcciones x-, y-, z- en un problema de deformación).

Las explicaciones que siguen utilizan esta notación general, que han de interpretarse en este sentido.

La salida o respuesta (output) dof es un vector fila -- en realidad devuelve parte del vector debc. El programa ensamblará automáticamente este vector dentro del vector global: debc

La salida o respuesta (output) vals es un vector columna -- en realidad devuelve parte del vector ebcVals. El programa ensamblará automáticamente este vector dentro del vector global: ebcVals.

Ejemplo:

(1). Supóngase un grado de libertad por nodo (ESTE ES SIEMPRE EL CASO PARA CAMPOS ESCALARES) y que se tienen condiciones ESENCIALES de borde en los lados 1 y 3.
Sea 2 el valor sobre el lado 1 y sea y el valor sobre el lado 3.
Entonces el código es:

```
dof = ndof*(nodeID - 1) +1;
```

Recuérdese la fórmula que calcula el grado de libertad global:

global dof = ndof * (global node number - 1) + i,
donde i = 1,2,...ndof.

Puesto que se tiene un solo dof por nodo, dof es un vector con un solo valor (el número de nodo global) .

```
if ( sideInd == 1)
  vals = 2;
end
```

Así, utilizando una instrucción if, si este nodo está sobre el borde 1, sus valor de la condición esencial de borde es 2. Recordar de nuevo que ndof es 1, así que vals es un vector que contiene un solo valor..

```
if (sideInd == 3)
  y = Nodes(2,nodeID);  -- usar primero nodeID para recuperar la coordenada y.
  vals = y;                    -- asignar el valor de la condición esencial de borde
end
```

Nota: Los marcadores de borde (4 en la región con forma de caja) se utilizan sólo para identificar los bordes. Las condiciones de borde (¿Qué nodos? ¿Qué dirección? ¿Qué valores?) se deben especificar 'manualmente' por el usuario dentro de esta función (y de modo similar en la función specifyNaturalBC.m para las condiciones de borde (BC) de tipo natural). La idea es que aquí no se utilizan directamente marcadores para las condiciones de borde como puede suceder en programas comerciales.

Tras llamar a esta función specifyEssBCvalue, enlazando funciones que la llaman, fillEssBC y a ésta ApplyBC, harán que se ensamblen automáticamente dof y vals dentro de las variables globales debc y ebcVals utilizando:

```
debc = [debc,dof]; ebcVals = [ebcVals;vals];
```

Sin embargo, todavía hay que considerar un caso especial: los nodos que son comunes a dos bordes. Si tiene una condición esencial de borde, aparecerá en debc y ebcVals dos veces. Por ello se emplea la función Matlab 'unique' para encontrar sus únicas componentes.

(2) ESTA PARTE ES RELEVANTE SOLO PARA PROBLEMAS CON MULTIPLES GRADOS DE LIBERTAD POR NODO (por ej. EN UN PROBLEMA DE DEFORMACION). Supóngase que hay 2 grados de libertad por nodo, como ocurre en problemas 2D de tensión plana. Supóngase que el lado 1 está fijado en ambas direcciones, esto es, los desplazamientos en x e y son cero. Entonces, el código es así:

```
if (sideInd == 1)
   dofx =  ndof*(nodeID - 1) + 1; -- el dof global para el desplazamiento u
   dofy =  ndof*(nodeID - 1) + 2; -- el dof global para el desplazamiento v
   dof = [dofx dofy];                  -- ponerlos dentro del vector dof como resultado
   vals = [0;0];       -- recordar que debe ser un vector columna
end
```

(3) Supóngase que se tiene una condición de borde sobre esta condición, sin embargo, no es condición esencial de borde. Habrá que inicializar estos dos vectores dof y vals, pero no hacer nada, es decir, inicializar siempre esos dos vectores:

```
dof = [];  vals = [];
```

El programa ignorará automáticamente las entradas vacías.

2. Especificar condiciones naturales de borde no nulas (FLUJO) Se ha programado esta situación solo para campos escalares - en este caso la condición de borde (BC) natural general es de tipo gradiente-flujo, por medio del flujo normal que en ocasiones se denomina q_n:

FLUJO NORMAL (escalar)=**vector flujo**·**n**=-K·**grad**(u)·**n**,

con **n** la **normal unitaria exterior al dominio**, **grad**(u)=[∂u/∂x, ∂u/∂y]

K es la matriz de 'conductividad' que recoge las propiedades constitutivas del medio

El programa gestiona las condiciones naturales a través del flujo del modo siguiente:

K **grad**(u)·**n**=alpha·u+beta

escogiendo los valores para los escalares alpha y beta en correspondencia con las condiciones naturales que se den como datos del problema

Para extender el programa a otros problemas (por ej. problemas de deformación), se requiere realizar modificaciones en el tipo de BC que no se analizan aquí.

La idea básica es la misma que antes:

```
for i = 1:length(IsBoundaryCondition)
    if ( IsBoundaryCondition(i) == 1)
        call function 'ApplyNaturalBc' para recorrer todos los elementos sobre el borde i
    end
end
```

El programa mira primero el valor en el vector: IsBoundaryCondition. Si el valor indica un borde que tiene condición de borde, entonces el programa recorrerá todos los elementos que dan sobre ese borde. Para cada elemento, se llamará a la siguiente función:

```
function [alpha, beta] = specifyNaturalBCValue(sideInd,x);
```

Hay dos argumentos de entrada (inputs):

sideInd -- el mismo que anteriormente, el número del borde, a fin de separar
diferentes bordes que tienen condiciones naturales de borde.

x -- se utiliza elemento finito unidimensional para borde.

integral.x es un vector que da las coordenadas globales de los puntos de integración de Gauss sobre ese borde. x(1) es la coordenada x , mientras x(2) es la coordenada y. La razón por la que se pasan estos argumentos es que las condiciones naturales de borde pueden ser función de x e y.

La forma general que se considera en el programa para la condición natural de borde para PROBLEMAS de CAMPO ESCALAR con matriz de conductividad K diagonal de elementos k_x, k_y, es:

$$K*du/dn = k_x*\partial u/\partial x*n_x + k_y*\partial u/\partial y*n_y = alpha*u + beta$$

k_x y k_y son las conductividades en las direcciones x e y respectivamente, n_x, n_y los cosenos de la normal unitaria saliente con las direcciones x e y; los coeficientes alpha y beta pueden ser en general funciones de (x,y) en 2D o (x,y,z) en 3D.

Así, los argumentos de salida (outputs) de esta función specifyNaturalBCValue.m son alpha y beta que se usan para calcular la matriz de 'rigidez' de cada elemento en contacto con el borde y el correspondiente vector de 'cargas'. Recordar la formulación débil de los problemas.

Si alpha =0, se reduce el caso a la condición de borde del segundo tipo consistente en imponer un valor para la condición natural que no depende del valor de u en la zona de borde considerada. Si depende del valor de u en esa zona de borde, en que u es incógnita, por ejemplo en condiciones tipo $c\cdot(u-u_{ext})$, con c y u_{ext} dados, entonces alpha no sería nulo.

Por ejemplo:

Considérese la malla en caja rectangular de 4 bordes del comienzo, obsérvese el dibujo para recordar el convenio de referencia los marcadores de borde 1, 2, 3, 4.

Supóngase que se tiene $\partial u/\partial x=1$ sobre el lado 2 y $\partial u/\partial x=2$ sobre el lado 4.

. Sobre el lado 2: $n_x=1$, $n_y=0$ => $k_x\cdot\partial u/\partial x*n_x+k_y*\partial u/\partial y*n_y= k_x*\partial u/\partial x = k_x$
luego alpha=0, beta=k_x

. Sobre el lado 4: $n_x=-1$, $n_y=0$ => $k_x*\partial u/\partial x*n_x + k_y*\partial u/\partial y*n_y=$
$= -k_x*\partial u/\partial x = -2*k_x$.
luego alpha = 0, beta = $-2*k_x$.

Por tanto el código es:

```
alpha = 0;
if (sideInd == 2)
    beta = kx;
end
if (sideInd == 4)
   beta = -2*kx;
end
```

Supóngase que se tiene condición de borde natural de flujo nulo (aislamiento). Entonces se han de inicializar estos dos valores alpha y beta, y simplemente, no hacer nada, es decir, inicializar siempre estos dos valores: alpha=0 ; beta=0;
El programa ignorará automáticamente las entradas que son cero.

Las condiciones de contorno que se presentan aquí permiten tratar casos posible para campos escalares (problemas de transferencia de calor, por ejemplo) . Para extender esto a problemas de deformación donde existe más de 1 DOF (Grado de Libertad, Degree Of Freedom) por nodo, será necesario hacer alguna modificación adicional, pues el tipo de condiciones de contorno es diferente de las aquí consideradas (aquí se prescribe el flujo normal, pero en problemas de deformación se pueden presentar diferentes componentes de tracción prescritas en direcciones diferentes).

B: ¿Dónde aparece la formulación débil? Funciones Integrands.m , Integrands4Side.m

Para modificar el programa, es necesario comprender las dos siguientes funciones: integrands.m , integrands4side.m.

1. integrands.m

Recuérdese la forma débil del método de los elementos finitos, se necesita evaluar una integral. Únicamente en casos muy simples se puede tener una forma explícita de esta integración, como ocurre en el elemento viga.
Sin embargo, para los aspectos de programación y como plantemiento general, no se conoce la forma explícita de esta integración pero sí se conoce la forma funcional de los integrandos. Por ello es adecuado realizar integración numérica (cuadratura de Gauss-Legendre) para calcular la integral. Para utilizar cuadratura de Gauss, se necesita conocer el valor de los integrandos en los puntos de Gauss correspondientes. Por ello, el papel de la función 'integrands.m' es proporcionar el valor de la 'matriz de rigidez' y el 'vector de cargas' en los puntos de integración de Gauss.

El algoritmo básico es como sigue:

for i = 1: todos los elementos en el mallado
  for j = 1 : número de puntos de integración de Gauss
    . Establecer la información básica de la función base de elementos finitos en ese punto.
      En general el propio programa se ocupará automáticamente de esta parte, no hay
      necesidad de modificar esta parte del código.
    . Llamar a la función integrands.m a fin de calcular el valor de la 'matriz de rigidez'
      y el 'vector de cargas' en este punto de Gauss.
  end
  Ensamblar la contribución del elemento a la 'matriz de rigidez' y 'vector de cargas' globales.
end

Así que, para modificar integrands.m, primero se necesita deducir la forma débil del problema que se está abordando, y establecer las matrices N y B que le corresponden (recordar las definiciones de N y B en las diapositivas de clase, para la conducción del calor y de modo similar para problemas de tensión plana), utilizando el argumento de entrada (input) 'felm' de esta función,

function [Ke,fe] = integrands(e, felm, Ke, fe);

porque toda la información sobre funciones base se ha preparado ya cuidadosamente en la estructura 'felm'.

2. integrands4side.m

Esta función sigue la misma idea que integrands.m. Recordar que cuando se utilizan condiciones de contorno o borde mixtas (que implican a u y a du/dx, etc.), hay integrales sobre el borde en la forma débil que aportan contribuciones tanto a la 'matriz de rigidez' como al 'vector de cargas'.
Se necesita realizar estas integrales sobre los lados de todos los elementos que están sobre la parte del contorno con condiciones de contorno mixtas.

En este programa, en vez de recorrer todos los elementos , sólo se recorren los elementos sobre ese borde, y se prepara la información básica de cada elemento finito para integración sobre borde 1-D (segmento) , en un problema 2D o integración 2D sobre una superficie de borde en el caso 3D.

Entonces el programa llamará a integrands4side.m para calcular la 'matriz de rigidez' de un elemento y el 'vector de cargas' en cada punto de Gauss sobre el borde que corresponda. Antes de reprogramar esta función, hay que tener en cuenta la forma débil del problema que se está resolviendo.

Ahora se puede ver por qué es importante conocer cómo se deduce la forma débil. Es la parte fundamental en la teoría de elementos finitos. La única diferencia entre problemas sobre un campo escalar es la geometría y los valores específicos de los coeficientes alpha y beta en las condiciones naturales de borde. Si el problema que se está resolviendo no está en una forma estándar, es necesario deducir la forma débil que le corresponde y modificar estas dos funciones.

Como se ha dicho antes, en problemas de deformación, se prescribe la 'tracción' (las componentes) para cada dirección x, y, z. La programación en integrands4Side.m es un poco diferente debido a la diferencia en las condiciones de contorno, pero la idea básica se mantiene.

C: Posproceso

En la función InputData se pueden elegir opciones sobre incluir ('yes') o no ('no') salidas gráficas que pueden ser de interés en el programa (malla de EF, numeración de nodos, borde del dominio, función respuesta como bandas de nivel con colores, vectores gradiente, resultado exacto si es caso,…

```
plot_mesh     = 'yes';  % What to plot. For big meshes,
plot_node     = 'yes';   % it is better not to plot node and vector.
plot_boundary = 'yes';  % Change this information as appropriate
plot_contour  = 'no';
plot_vector   = 'yes';
is_exact      = 'no';
```

Algunas de las opciones pueden generar errores dependiendo de la versión de Matlab que se utilice, y también pueden dar problemas con Octave.

En la función postprocessor_refine.m (postprocessor.m) se realiza la gestión de varias de estas salidas gráficas en las que se dibujan resultados y se hacen algunos pasos de posproceso que pueden ser dependientes del problema. En esta función se gestiona la comparación con la solución exacta del problema, caso de que exista. La solución exacta se puede dar en la función exact.m .En el programa global se proporcionan algunas funciones de utilidad:

. plotcontour.m -- Dibuja el valor/es del campo- El argumento de entrada (input) es un vector columna que se desea dibujar sobre la malla de elementos finitos. Es el valor nodal sobre cada nodo. Es muy general y se puede dibujar cualquier magnitud que se desee, por ejemplo temperatura, potencial, desplazamiento, tensión, deformación.

. plotvector.m -- dibuja el vector asociado al campo, como flujo de calor, gradiente de temperatura o presión, vector velocidad.

. makeGradient.m -- Calcula la derivada de cualquier magnitud de campo. Su input es también un vector columna con el que se desea calcular la derivada de la magnitud.

Es importante observar que todos los inputs de las tres funciones anteriores son vectores columna cuyo tamaño es el mismo que el número de nodos y que la ordenación de los datos en este vector se corresponde con la numeración global de los nodos.

. La función trisurf de Matlab permite construir una imagen 3D de la solución de la EDP, a partir de la malla de elementos triangulares contenida en Elems, y de las coordenadas de los nodos, que están en la matriz Nodes. Si se tiene una malla cuadrilátera, se puede construir una triangular a partir de ella, trazando una diagonal en cada cuadrilátero. Seguidamente se puede representar con trisurf la superficie triangulada que interpola los vértices de la malla. El atributo colorbar de trisurf permite visualizar mediante colores los valores de la función representada en la imagen 3D. Esta imagen 3D de la solución se dibuja siempre para el caso del último refine de la malla, el más fino.

. Se puede usar la eficiente función patch() para construir una imagen 3D de la solución de la EDP . Al final de la función postprocessor_refine se pueden retocar algunas instrucciones para hacer la gestión con patch().

II) Estructura modular del programa 2DBVP

(Nicholas Zabaras, adaptado por Jaime Puig-Pey (UC, 2013))

include_variables : es script, no function. Para incluir variables globales
(añadidas a las de Zabaras por JPE: nno_ini: nº nodos malla inicial, nrefine, KontaRefine,...)

Main2D_refine **DATO nrefine: núm refines malla sobre la inicial**
include_variables
preprocessor_refine-> include_variables **DATO load_grid : 'refine' , 'no' (rectang), 'yes' (Ansys)**
InputData **DATO vector fila IsBoundaryCondition, var is_exact, var para selec plot**
include_variables
InputGrid_refine-> include_variables **DATO ElementType: 1: Q4, 2: T3**

BoxGrid_2D
loadFromGridFile (ansys')
InputIniGeometria-> include_variables **DATO: Nodes, Elems, BoundaryNodes.Nodes**
FunDivTriangBordes
FunDivCuadBordes
(FunGeomPolar , en InputIniGeometria especial, carpeta 2dBVP_EF_MallasPolares)
GenElemSurfaceIndicator-> include_variables (function interna en InputIniGeometria)
isBoundaryNode (function interna en InputIniGeometria)
PlotGrid-> include_variables
Assemple-> include_variables
gauss
FiniteElement_2D-> include_variables
integrands-> include_variables
Conductivity: matriz 'conductividades' **DATO: matriz D de Conductivity(x)**
ff : función término independiente en la EDP . **DATO: value de ff(x)**
AssembleGlobalMatrix-> include_variables
ApplyBC-> include_variables
fillEssBC-> include_variables
specifyEssBCvalue-> include_variables **DATO: valores condic. esenciales nodos**
ApplyNaturalBC-> include_variables
integrands4sid-> include_variables
specifyNaturalBCvalue-> include_variables **DATO: valores condic. naturales lados**
NodalSoln
postprocessor_refine-> include_variables
plotcontour-> include_variables
makeGradient-> include_variables
gauss
FiniteElement_2D-> include_variables
plotvector-> include_variables
exact **DATO de solución exacta y su gradiente exacto**
plotcontour-> include_variables
plotvector-> include_variables

III) Problema de Ejemplo . 2dBVP

Con el programa 2dBVP en Matlab, se considera el siguiente problema estacionario de conducción del calor en 2D:

$$-\nabla^2 u = f(x,y)$$

siendo u es una función escalar de x e y, desconocida , definida sobre el dominio cuadrado [0,1]x[0,1], y f es una función de x, y, conocida $f(x,y) = 2\pi^2 \sin(\pi x)\sin(\pi y)$

Las condiciones de borde son:

$$u(x,0) = u(x,1) = 0 \quad \frac{\partial u}{\partial x}(1,y) = -\pi \sin(\pi y), \quad \frac{\partial u}{\partial x}(0,y) = \pi \sin(\pi y)$$

. En este ejemplo, la solución analítica exacta, es: $u(x) = \sin(\pi x)\sin(\pi y)$

Las condiciones naturales en 2dBVP se introducen a través del flujo normal:
vector flujo·n=-K·**grad**(u)·**n**, con **n** la **normal unitaria exterior al dominio**, **grad**(u)=[∂u/∂x, ∂u/∂y]

El programa gestiona las condiciones naturales a través del flujo del modo siguiente:

K **grad**(u)·**n**=alpha·u+beta

En nuestro problema la matriz de 'conductividad' K asociada a las propiedades constitutivas del medio es la identidad I (2x2)
En el borde x=1, K·**grad**(u)·**n**=[-π·sin(π·y),??]·[1,0] =-π·sin(π·y)
En el borde x=0, K·**grad**(u)·**n**=[π·sin(π·y),??]·[-1,0] =-π·sin(π·y)

Obsérvese que las 2 condiciones naturales de borde son de la forma (K=I(2) o k=1 aquí):

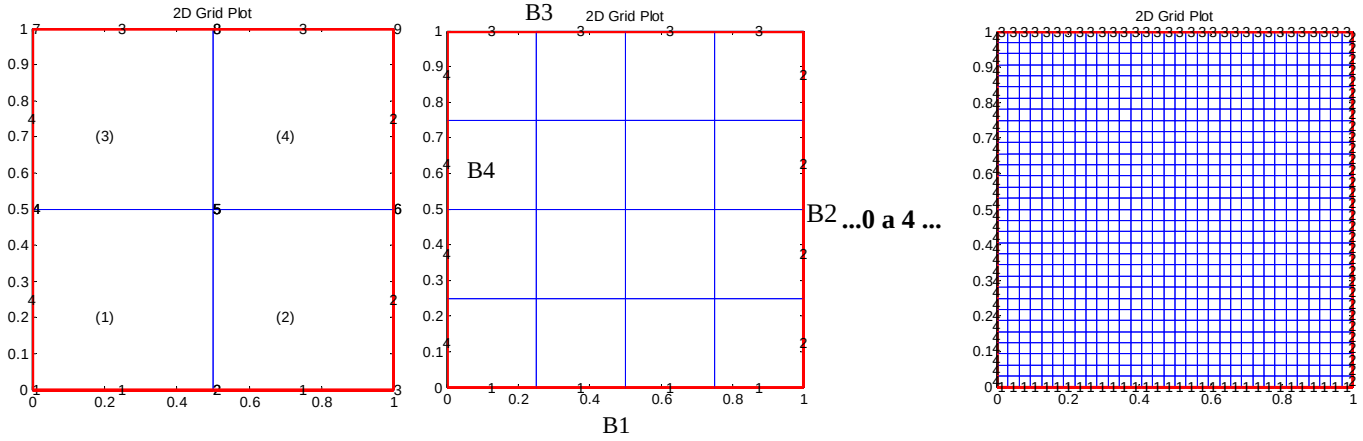
K·**grad**(u)·**n** =-π·sin(π·y)=alpha·u+beta

es decir, se tomará en las zonas de contorno con condiciones naturales: beta=-π·sin(π·y), alpha=0

Obsérvese que el flujo normal en 2dBVP es positivo si sale del dominio, pues entonces tendrá la misma orientación que la normal exterior (la que apunta hacia el exterior del dominio)

Resolución:

Vamos a ir introduciendo los DATOS específicos para este problema en los diversos módulos del programa 2DBVP. Se realizarán 4 refines de la malla inicial, es decir, 5 ejecuciones sucesivas



La malla inicial será de 4 elementos cuadriláteros.
Se realizarán 4 refines, es decir, 5 ejecuciones, respectivamente de 2x2, 4x4, 8x8, 16x16, 32x32=1024 elementos cuadriláteros.
En el programa principal, archivo `Main2D_refine`, se asignará:

```
nrefine=4;
```

Atendiendo a las condiciones de contorno o borde, se consideran 4 bordes del dominio. El **vector fila IsBoundaryCondition**: tendrá 4 componentes. Las 4 componentes valdrán 1, pues en cada uno de ellos

hay una condición esencial de contorno o una condición natural de flujo no nula. Si hubiera una condición de flujo nulo en un borde, en su componente en IsBoundaryCondition se pondría el valor 0.

. Entonces en la función `InputData`

```
function InputData
... ... ....
IsBoundaryCondition = [1 1 1 1];
```

También en la función `InputData` se especifica si el problema tiene solución exacta, para comparar :

```
is_exact      = 'yes';
```

Y también en la función `InputData` se especifican valores para variables de selección de dibujos a realizar por el programa:

```
plot_mesh     = 'yes';  % What to plot. For big meshes,
plot_node     = 'yes';   % it is better not to plot node and vector.
plot_boundary = 'yes';  % Change this information as appropriate
plot_contour  = 'yes';
plot_vector   = 'yes';
```

. Se introducirá el tipo de elemento finito para la malla en la función `InputGrid_refine`

```
function InputGrid_refine
... ... ...
ElementType = 1; nen=4;   % 1, nen=4: Cuadri4       2: Tri3, nen=3
```

En esta función se asigna automáticamente el número de dimensiones espaciales:

```
nsd = 2;
```

. En la función `InputIniGeometria`

```
function InputIniGeometria
```

. Se dan las coordenadas cartesianas de los nodos en la malla inicial:

```
Nodes=[0,0;1/2,0;1,0;0,1/2;1/2,1/2;1,1/2;0,1;1/2,1;1,1];
Nodes=Nodes';
```

Debe quedar una matriz de dos filas y las coordenadas x e y de cada nodo en columna

. Se dan los elementos por los índices globales de sus nodos (EN ORDEN ANTIHORARIO!!):

```
Elems=[1,2,5,4;2,3,6,5;4,5,8,7;5,6,9,8];Elems=Elems';
```

El elemento i: i-ésima columna, con los índices globales de sus nodos recorridos en sentido antihorario.

. Se dan los nodos (índices globales) que componen cada borde, en la estructura `BoundaryNodes`

```
       BoundaryNodes(1).Nodes=[1,2,3]; % No importa orden
       BoundaryNodes(2).Nodes=[3,6,9];
       BoundaryNodes(3).Nodes=[9,8,7];
       BoundaryNodes(4).Nodes=[7,4,1];
```

. Se da la matriz de 'conductividad' del medio en la función:

```
function D = Conductivity(x)
% Sets up the `conductivity' matrix (2x2 for 2D) or (3x3 for 3D)
% at point x=(x(1),x(2),x(3)).
```

En este caso es la matriz constante 2x2:

```
D = [1  0;...
     0  1]; % Esto para Problema Ejemplo 2DBVP. Otros casos,
adaptar.
```

. Se da el término independiente de la EDP: $-\nabla^2 u = f(x,y)$ en la función `function`

```
value=ff(x)
value = 2*pi^2*sin(pi*x(1))*sin(pi*x(2));% ejemplo muestra, 2DBVP
```

. Se dan las condiciones esenciales de borde en la función `specifyEssBCValue`

```
function [dof, vals] = specifyEssBCValue(nodeID, sideInd)
... ... ...
if ( sideInd == 1 || sideInd == 3)   %Ej prueba calor dominio
cuadrado
                                       % Example problem 2D BVP
     dof = ndof*(nodeID - 1) + 1;   % Here, we only have one dof
equal
                                  % to nodeID (ndof=1, dof=nodeID).
     vals = 0;           % dof and vals are scalars for this
program!
 end
```

. Se dan las condiciones naturales de borde en la función `specifyNaturalBCValue`

```
function [alpha, beta] = specifyNaturalBCValue(sideInd,x);
... ... ...
alpha = 0;                          % initialize the two output
variables
beta  = 0;

% Ejemplo prueba base de Zabaras, Example 2D BVP
if ( sideInd == 2 || sideInd == 4 ) % Add the NC value below
        beta  = -pi*sin(pi*x(2));
end
```

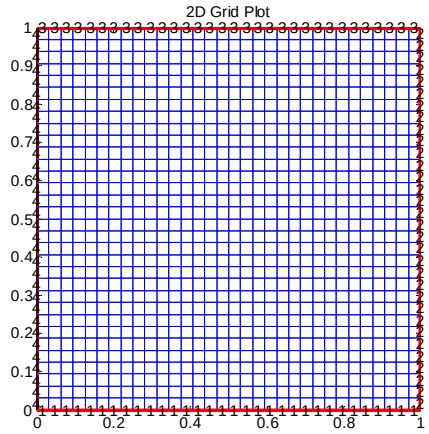
. Se da la solución exacta si se conoce, así como su gradiente en la función `exact`

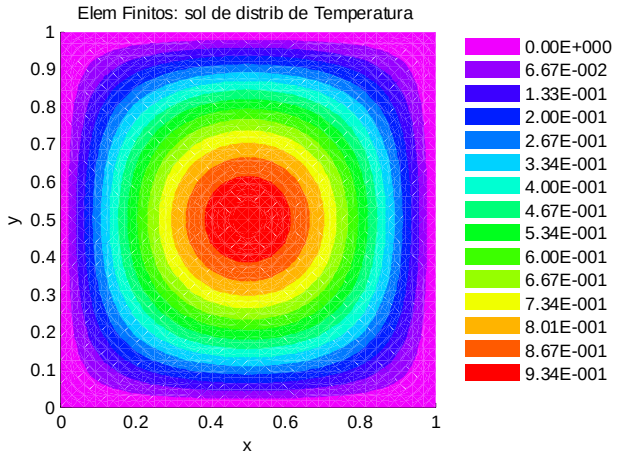
```
function [u dudx dudy] = exact ( x )
… … …
% Este es el ejemplo de muestra 2D BVP
 u    =      sin ( pi * x(1) ) * sin ( pi * x(2) ) ;
 dudx = pi * cos ( pi * x(1) ) * sin ( pi * x(2) ) ;
 dudy = pi * sin ( pi * x(1) ) * cos ( pi * x(2) );
```

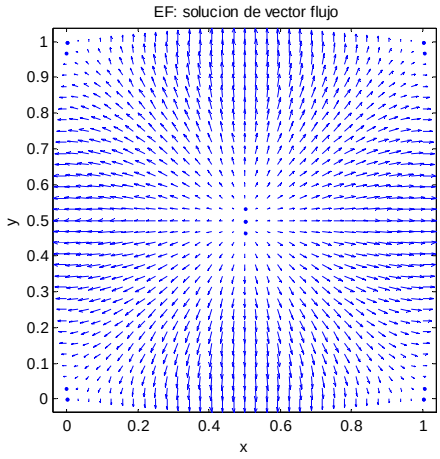
Ejecútese el programa y compruébese que se obtienen los siguientes resultados:

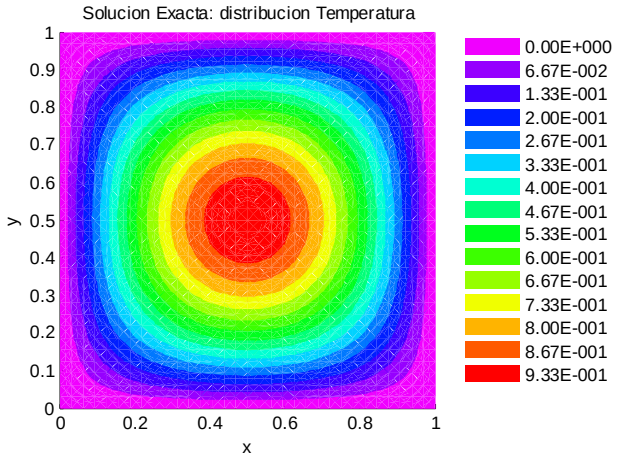
Evol =

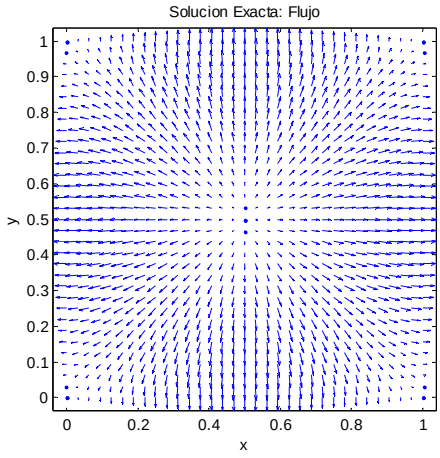
1	0.00000	0.00000	0.00000	0.00000	0.00000
2	0.00000	0.00000	0.00000	0.00000	0.00000
3	0.00000	0.00000	0.00000	0.00000	0.00000
4	0.03828	0.00196	0.00012	0.00001	0.00000
5	1.23721	1.05368	1.01300	1.00322	1.00080
6	0.03828	0.00196	0.00012	0.00001	0.00000
7	0.00000	0.00000	0.00000	0.00000	0.00000
8	0.00000	0.00000	0.00000	0.00000	0.00000
9	0.00000	0.00000	0.00000	0.00000	0.00000







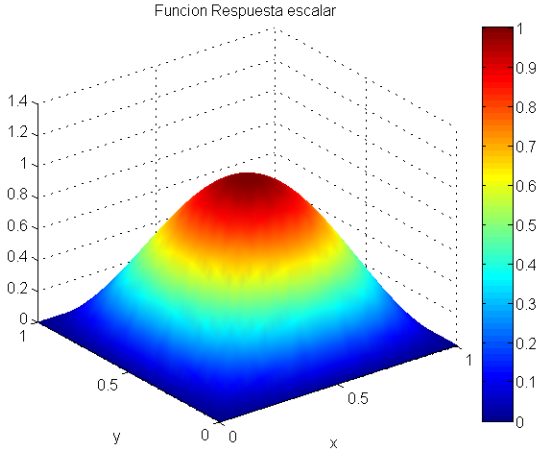




Informe de errores:

El máximo error en los valores nodales es: 8.037550e-004

El máximo error en los flujos nodales es: 2.523446e-003



La memoria utilizada por el programa, en la ventana workspace:

Workspace

File Edit View Graphics Debug Desktop Window Help

Stack: Base Select data to plot

Name	Size	Bytes	Class	Value
BoundaryElems	1x4	3072	struct (global)	<1x4 struct>
BoundaryNodes	1x4	1568	struct (global)	<1x4 struct>
ElementType	1x1	8	double (global)	1
Elems	4x1024	32768	double (global)	<4x1024 double>
Evol	9x65	1170	char	<9x65 char>
IsBoundaryCondition	1x4	32	double (global)	[1,1,1,1]
K	1089x1089	182960	double (global sparse)	<1089x1089 double>
KontaRefine	1x1	8	double (global)	4
Nodes	2x1089	17424	double (global)	<2x1089 double>
aux	1x65	130	char	' 9 0.00000 0.00000 ...
d	1089x1	8712	double (global)	<1089x1 double>
d_refine	9x5	360	double	<9x5 double>
debc	1x66	528	double (global)	<1x66 double>
ebcVals	66x1	528	double (global)	<66x1 double>
f	1089x1	8712	double (global)	<1089x1 double>
i	1x1	8	double	9
is_exact	1x3	6	char (global)	'yes'
load_grid	1x6	12	char (global)	'refine'
n_d_ini	1x1	8	double	9
ndof	1x1	8	double (global)	1
nel	1x1	8	double (global)	1024
nen	1x1	8	double (global)	4
neq	1x1	8	double (global)	1089
nno	1x1	8	double (global)	1089
nno_ini	1x1	8	double (global)	9
nrefine	1x1	8	double (global)	4
nsd	1x1	8	double (global)	2
ntriplets	1x1	8	double (global)	16384
plot_boundary	1x3	6	char (global)	'yes'
plot_contour	1x3	6	char (global)	'yes'
plot_mesh	1x3	6	char (global)	'yes'
plot_node	1x2	4	char (global)	'no'
plot_vector	1x3	6	char (global)	'yes'
r	66x1	528	double	<66x1 double>

La estructura de la matriz de 'rigidez' K, global:

>> figure, spy(K)
nz=9409 , nº de elem de K no nulos