Manual de Lecturas del Curso:

Métodos Topológicos en Mecánica Estructural y de Medios Continuos

Fredy Vides
Centro de Innovación en Cómputo Científico e Industrial
Departamento de Matemática Aplicada
Escuela de Matemática y Ciencias de la Computación
Universidad Nacional Autónoma de Honduras

E-mail: fredy.vides@unah.edu.hn

6 de Agosto de 2019

Presentación

En este documento presentaremos algunas técnicas de cálculo de deformaciones (topológicas) continuas utilizando las herramientas computacionales FreeFem++, Calculix y Octave. Muchas de las técnicas que presentamos han sido desarrolladas en el CICCI-UNAH.

El material presentado en este documento está orientado a profesionales de ingeniería, arquitectura o biotecnología, con sólidos conocimientos de cálculo en varias variables y álgebra lineal, y que tengan interés en calcular deformaciones como las que serán estudiadas en este texto.

Los contextos de aplicación del cómputo de deformaciones que consideraremos en este material, incluyen el cálculo de deformaciones en mecánica estructural, deformaciones en mecánica de fluidos y deformaciones en entornos biológicos y micro-biológicos.

Índice general

Capítulo 1

Mallado de Materiales y Elementos Estructurales

Objetivos

1. Definir geometría correspondiente a una material o elemento estructural determinado.
2. Construir malla computacional de análisis de elemento finito de un material o elemento estructural dado.
3. Refinar la malla computacional de análisis de elemento finito de un material o elemento estructural dado.

1.1. Mallado de Materiales y Elementos Bidimensionales

Consideremos un material rectangular de dimensiones $[x_0, x_1] \times [y_0, y_1]$ cuyos bordes de contorno estan etiquetados como se muestra en la figura 1.1.

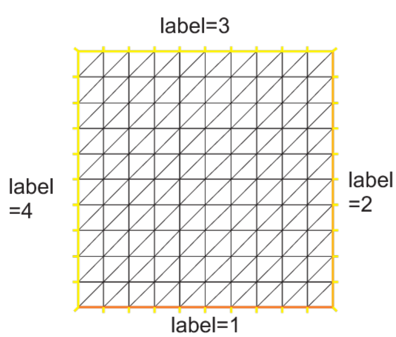

Figura 1.1: Malla rectangular básica genérica.

Puede utilizarse el comando `square` para construir una malla de análisis de este material, el código de un programa FreeFEM que puede ser usado para este propósito y que llamaremos `malla2D.edp` se muestra a continuación.

```
real x0 = 1.2;
```

```
real x1 = 2.2;
real y0 = 0;
real y1 = 2;
int n = 10;
real m = 20;
mesh Th = square(n, m, [x0+(x1-x0)*x, y0+(y1-y0)*y]);
plot(Th,wait=true);
```

Al ejecutar `malla2D.edp` con FreeFEM obtenemos la salida gráfica mostrada en la figura 1.2.

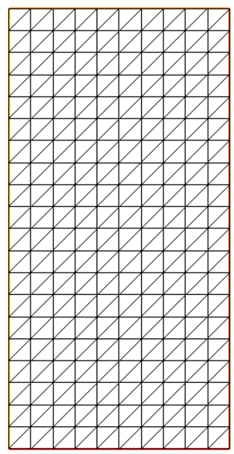

Figura 1.2: Malla rectangular básica genérica generada por `malla2D.edp`.

1.1.1. Mallado Basado en Bordes Geométricos

Utilizando el comando `buildmesh` de FreeFEM podemos construir mallas de materiales planos con base en los bordes geométricos de los materiales en estudio. Un ejemplo de esto se ilustra en el programa FreeFEM `ConstMalla2D.edp` cuyo código se presenta a continuación.

```
int upper = 1;
int others = 2;
int inner = 3;

border C01(t=0, 1){x=0; y=-1+t; label=upper;}
border C02(t=0, 1){x=1.5-1.5*t; y=-1; label=upper;}
border C03(t=0, 1){x=1.5; y=-t; label=upper;}
border C04(t=0, 1){x=1+0.5*t; y=0; label=others;}
border C05(t=0, 1){x=0.5+0.5*t; y=0; label=others;}
border C06(t=0, 1){x=0.5*t; y=0; label=others;}
border C11(t=0, 1){x=0.5; y=-0.5*t; label=inner;}
border C12(t=0, 1){x=0.5+0.5*t; y=-0.5; label=inner;}
border C13(t=0, 1){x=1; y=-0.5+0.5*t; label=inner;}

int n = 10;
```

```
plot(C01(-n) + C02(-n) + C03(-n) + C04(-n) + C05(-n)
  + C06(-n) + C11(n) + C12(n) + C13(n), wait=true);

mesh Th = buildmesh(C01(-n) + C02(-n) + C03(-n) + C04(-n) + C05(-n)
  + C06(-n) + C11(n) + C12(n) + C13(n));

plot(Th, wait=true);

cout << "Part 1 has region number " << Th(0.75, -0.25).region << endl;
cout << "Part 2 has redion number " << Th(0.25, -0.25).region << endl;
```

Al ejecutar `ConstMalla2D.edp` en FreeFEM se obtienen las salidas gráficas mostradas en la figura 1.3.

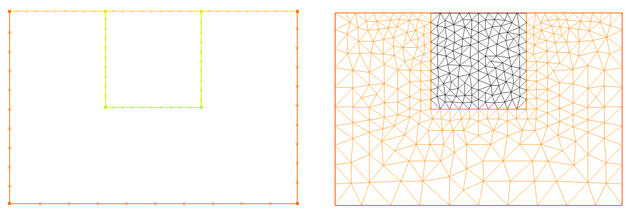

Figura 1.3: Malla rectangular basada en bordes generada por `ConstMalla2D.edp`.

Consideremos otro ejemplo de mallado de material bidimiensional basado en bordes, es este caso consideraremos tanto el mallado combinado de regiones de material como el mallado de un material *perforado*. Para este fin utilizaremos los programas `MallaMatComb2D.edp` y `MallaMatPer2D.edp`. El código de `MallaMatComb2D.edp` se presenta a continuación.

```
border a(t=0, 2*pi){x=cos(t); y=sin(t); label=1;}
border b(t=0, 2*pi){x=0.3+0.3*cos(t); y=0.3*sin(t); label=2;}
mesh ThComb = buildmesh(a(50) + b(30));
plot(a(50) + b(30),wait=true);
plot(ThComb);
```

Al ejecutar `MallaMatComb2D.edp` con FreeFEM se obtienen las salidas gráficas mostradas en la figura 1.4.

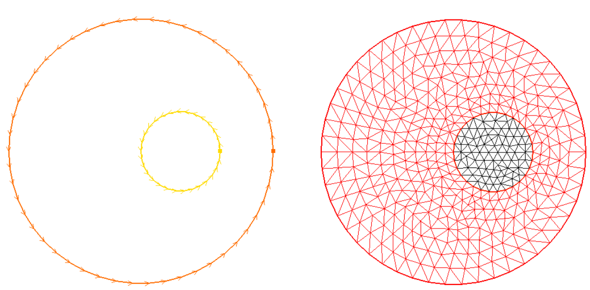

Figura 1.4: Malla bidimensional basada en bordes generada por `MallaMatComb2D.edp`.

El código de `MallaMatPer2D.edp` se presenta a continuación.

```
border a(t=0, 2*pi){x=cos(t); y=sin(t); label=1;}
border b(t=0, 2*pi){x=0.3+0.3*cos(t); y=0.3*sin(t); label=2;}
mesh ThPer = buildmesh(a(50) + b(-30));
plot(a(50) + b(-30),wait=true);
plot(ThPer);
```

Al ejecutar `MallaMatPer2D.edp` con FreeFEM se obtienen las salidas gráficas mostradas en la figura 1.5.

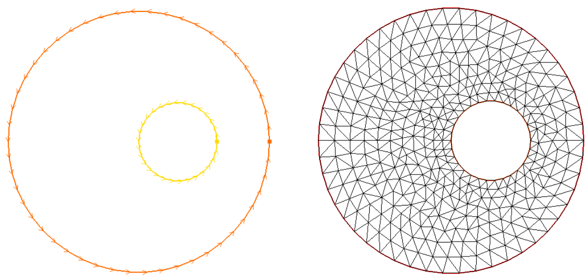

Figura 1.5: Malla bidimensional perforada basada en bordes generada por `MallaMatPer2D.edp`.

1.1.2. Refinamiento de Mallas Materiales Bidimenionales

Dada una malla $\mathcal{M}_h$ de un material bidimensional $\mathcal{M}$, utilizando el comando `adaptmesh` de FreeFEM es posible refinar la malla $\mathcal{M}_h$ obteniendo una malla $\mathcal{M}_s$ para $s \leq h$.

Ilustraremos el procedimiento de refinamiento de mallas bidimensionales utilizando el programa `MallaRef2D.edp` cuyo código se muestra a continuación.

```
mesh Th=square(2, 2);
plot(Th, wait=true);
Th = adaptmesh(Th, 1./30., IsMetric=1, nbvx=10000);
plot(Th, wait=true);
Th = adaptmesh(Th, 1./30., IsMetric=1, nbvx=10000);
plot(Th, wait=true);
```

Al ejecutar `MallaRef2D.edp` con FreeFEM obtenemos las salidas gráficas mostradas en la figura 1.6.

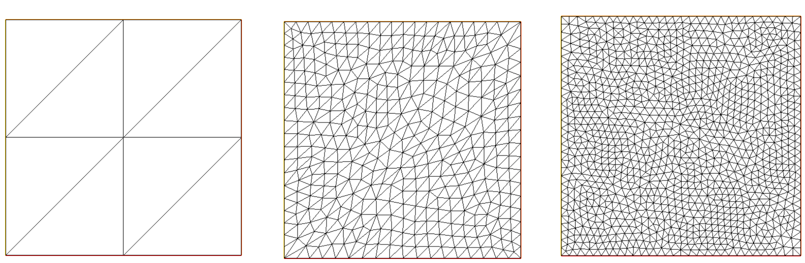

Figura 1.6: Malla bidimensional junto con dos niveles de refinamiento generados por `MallaRef2D.edp`.

1.2. Mallado de Materiales Tridimensionales

Consiederemos un material tridimensional $\mathcal{M}$ tipo paralelepípedo de dimensiones $[x_0, x_1] \times [y_0, y_1] \times [z_1, z_1]$, podemos calcular una malla tridimensional $\mathcal{M}_h$ para este material utilizando el comando `cube` de FreeFEM. Un ejeplo de uso del comando `cube` se muestra en el programa `malla3D.edp` cuyo código se muestra a continuación.

```
include "cube.idp"
int[int] Nxyz = [20, 4, 4];
real x0,x1,y0,y1,z0,z1;
x0=0;
x1=2;
y0=0;
y1=0.4;
z0=0;
z1=0.4;
real [int, int] Bxyz = [[x0, x1], [y0, y1], [z0, z1]];
int [int, int] Lxyz = [[1, 2], [2, 2], [2, 2]];
mesh3 Th = Cube(Nxyz, Bxyz, Lxyz);
plot(Th);
```

Al ejecutar `malla3D.edp` con FreeFEM se obtiene una salida gráfica como la mostrada en la figura 1.7.

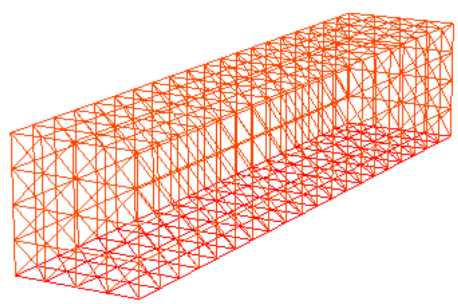

Figura 1.7: Malla tridimensional generada por `malla3D.edp`.

1.2.1. Mallado Basado en Bordes Geométricos

La construcción de mallas basadas en bordes geométricos también es posible para materiales y elementos estructurales tridimensionales. A manera de ejemplo construiremos una malla de análisis computacional para una sala regular tridimensional.

Cálculo de una Malla basada en bordes Geométricos

Consideremos un modelo de sala tridimensional como la ilustrada en la figura 1.8.

Es posible generar una malla tridimensional para la sala, utilizando el programa FreeFEM `Sala3D.edp` cuyo código se muestra a continuación.

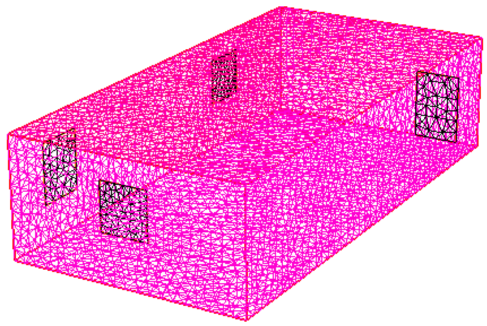

Figura 1.8: Modelo geométrico tridimensional de sala.

```
load "msh3"
load "tetgen"
load "medit"
int n= 2;
border a01(t=0,20.0) {x=0+t;     y=0;     label=1;};   // (0,0)  => (20,0)
border a02(t=0,5.0)  {x=20.;  y=0+t;  label=1;};     // (20,0) => (20,5)
border a03(t=0,20.0) {x=20-t;    y=5.;     label=1;};   // (20,5) => (0,5)
border a04(t=0,5.0)  {x=0;  y=5.-t;  label=1;};   // (0,5)  => (0,0)
mesh right = buildmesh (  a01(20*n) + a02(5*n) + a03(20*n) + a04(5*n)); // 20m
int nnn=right(0,1.5).region;
border b01(t=0,2.0)  {x=4.+t;    y=2.;    label=2;};   // (4,2)  => (6,2)
border b02(t=0,2.0)  {x=6.;     y=2.+t; label=2;};    // (6,3)  => (6,7)
border b03(t=0,2.0)  {x=6-t;     y=4.;     label=2;};      // (6,7)  => (4,7)
border b04(t=0,2.0)  {x=4.;     y=4.-t;   label=2;};
border b11(t=0,2.0)  {x=16.+t;    y=1.5;      label=3;};    // (16,3)  => (18,3)
border b12(t=0,2.5)  {x=18.;     y=1.5+t;  label=3;};   // (18,3)  => (18,4)
border b13(t=0,2.0)  {x=18-t;       y=4.;     label=3;};    // (18,4)  => (16,4)
border b14(t=0,2.5)  {x=16.;       y=4.-t;   label=3;};
mesh left= buildmesh (b01(3*n)  + b02(3*n)  + b03(3*n) +b04(3*n)+
         b11(2*n) + b12(3*n)  + b13(2*n)  + b14(3*n)
         +a01(20*n) + a02(5*n) + a03(20*n) + a04(5*n)
);
border c01(t=0,20.0) {x=0.+t;     y=0.;      label=1;};   // (0,0)  =>(20,0)
border c02(t=0,10.0) {x=20.;     y=0+t;   label=1;};   // (20,0) =>(20,10)
border c03(t=0,20.0) {x=20.-t;     y=10.;    label=1;};   // (20,10)=>(0,10)
border c04(t=0,10.0) {x=0.;         y=10.-t;  label=1;};   // (0,10) => (0,0)
mesh top= buildmesh (  c01(20*n)+ c02(10*n)+ c03(20*n)+ c04(10*n)
);
border a11(t=0,10.0) {x=0+t;   y=0;      label=1;};       // (0,0)  => (10,0)
border a12(t=0,5.0)  {x=10.;     y=0+t;  label=1;};       // (10,0) => (10,5)
border a13(t=0,10.0) {x=10-t;   y=5.;    label=1;};       // (10,5) => (0,5)
border a14(t=0,5.0)  {x=0;        y=5.-t;   label=1;};
mesh back= buildmesh (  b01(3*n)  + b02(3*n)  + b03(3*n)  + b04(3*n)+
```

```
                                        a11(10*n) + a12(5*n) + a13(10*n) + a14(5*n)
                                   );
border ad11(t=0,7.0) {x=0+t;   y=0;      label=1;};       // (0,0)  => (7,0)
border ad12(t=7.0,9.0)  {x=t;     y=0;  label=2;};       // (7,0) => (9,0)
border ad13(t=9.0,10.0) {x=t;   y=0;    label=1;};       // (9,0) => (10,0)
border ad14(t=0,5.0)  {x=10;        y=t;   label=1;};       // (10,0)  => (10,5)
border ad15(t=0,10.0)  {x=10-t;   y=5;   label=1;};       // (10,5)  => (0,5)
border ad16(t=0,5.0)  {x=0;   y=5-t;   label=1;};       // (0,5)  => (0,0)
border adoor1(t=0,3.0)  {x=9;   y=t;   label=2;};       // (9,0)  => (9,3)
border adoor2(t=0,2.0)  {x=9-t;   y=3;   label=2;};       // (9,3)  => (7,3)
border adoor3(t=0,3.0)  {x=7;   y=3-t;   label=2;};       // (7,3)  => (7,0)
mesh front = buildmesh (  adoor1(3*n) + adoor2(2*n)+
 adoor3(3*n)+
                           ad11(7*n) + ad12(2*n) + ad13(n) + ad14(5*n)
+ad15(10*n) + ad16(5*n)
);
border cd01(t=0,20.0) {x=0.+t;     y=0.;      label=1;};   // (0,0)  =>(20,0)
border cd02(t=0,10.0) {x=20.;     y=0+t;   label=1;};   // (20,0) =>(20,10)
border cd03(t=0,20.0) {x=20.-t;     y=10.;    label=1;};   // (20,10)=>(0,10)
border cd041(t=0,1) {x=0.;         y=10.-t;  label=1;};   // (0,10) => (0,9)
border cd042(t=0,2) {x=0.;         y=9.-t;  label=1;};   // (0,9) => (0,7)
border cd043(t=0,7.0) {x=0.;         y=7.-t;  label=1;};   // (0,7) => (0,0)
mesh floor= buildmesh (  cd041(n)+cd042(2*n)+ cd043(7*n)+
                                        cd01(20*n)+ cd02(10*n)+ cd03(20*n)
);
int[int] refFront=[0,20,4,30];
meshS Front = movemesh23(front,transfo=[0,x,y],label=refFront,orientation=1);
int[int] refRight=[0,20];
meshS Right= movemesh23(right,transfo=[x,10.,y],label=refRight,orientation=1);
int[int] refBack=[0,20,4,60];
meshS Back = movemesh23(back,transfo=[20.,x,y],label=refBack,orientation=-1);
int[int] refLeft=[0,20,4,50,8,40];
meshS Left = movemesh23(left,transfo=[x,0,y],label=refLeft,orientation=-1);
int[int] refFloor=[0,20];
meshS Floor= movemesh23(floor,transfo=[x,y,0.],label=refFloor,orientation=1);
int[int] refTop=[0,21];
meshS Top = movemesh23(top,transfo=[x,y,5.],label=refTop,orientation=-1);
meshS Thsalle=Right+Left+Back+Top+Floor+Front;
plot (Thsalle, cmm="Sala",wait=1);
mesh3 Th2 = tetg(Thsalle,switch="pqaAAYYQ");
plot (Th2,cmm="Room 3D  ",wait=1);
medit("Room3D",Th2);
```

Al ejecutar `sala3D.edp` con FreeFEM se obtienen las salidas gráficas mostradas en la figura 1.9.

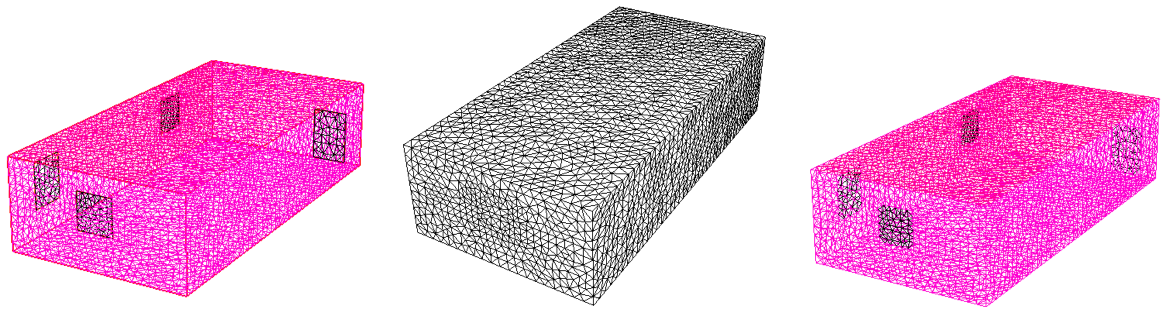

Figura 1.9: Malla tridimensional basada en bordes para la sala.

Cálculo de una Malla de un Material Tridimensional Perforado

Consideremos una pieza de material perforado como la descrita por la figura 1.10.

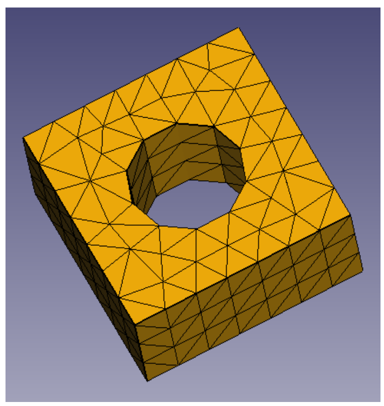

Figura 1.10: Representación geométrica de pieza de material tridimensional perforado.

Podemos calcular una malla de análisis computacional para esta pieza utilizando el programa FreeFEM `MallaPer3D.edp` coyo código se presenta a continuación.

```
load "msh3"
load "medit"
searchMethod=1;
verbosity=1;
real a=1, d=0.5, h=0.5;
border b1(t=0.5,-0.5) {x=a*t; y=-a/2; label=1;};
border b2(t=0.5,-0.5) {x=a/2; y=a*t; label=2;};
border b3(t=0.5,-0.5) {x=a*t; y=a/2; label=3;};
border b4(t=0.5,-0.5) {x=-a/2; y=a*t; label=4;};
border i1(t=0,2*pi) {x=d/2*cos(t); y=-d/2*sin(t); label=7;};
int nnb=7, nni=10;
mesh Th=buildmesh(b1(-nnb)+b3(nnb)+b2(-nnb)+b4(nnb)+i1(nni));
int nz=3;
int[int] rup=[0,5], rlow=[0,6], rmid=[1,1,2,2,3,3,4,4,7,7], rtet=[0,41];
```

```
func zmin=0;
func zmax=h;
mesh3 Th3=buildlayers(Th, nz, zbound=[zmin,zmax],
reftet=rtet,reffacemid=rmid, reffaceup=rup, reffacelow=rlow);
plot(Th3,wait=1);
medit("Th3",Th3);
```

Al ejecutar `MallaPer3D.edp` con FreeFEM se obtienen las salidas gráficas mostradas en la figura 1.11.

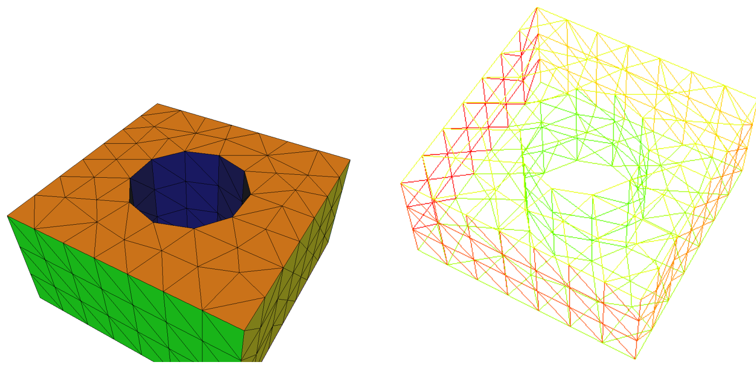

Figura 1.11: Malla tridimensional basada en bordes para la pieza.

1.2.2. Mallado Basado en Bordes Geométricos de Superficies en Medios Continuos Tridimensionales

Consideremos una supeficie $\mathcal{T}$ de material en un medio continuo tridimensional como la mostrada en la figura 1.12.

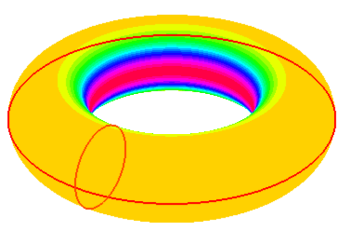

Figura 1.12: Superficie de material $\mathcal{T}$ en un medio continuo tridimensional.

Podemos contruir una malla de análisis para esta superficie con el programa FreeFEM `MallaSuper3D.edp` cuyo código se presenta a continuación.

```
load "msh3"
load "tetgen"
real R = 3, r=1;
real h = 0.2; //
int nx = R*2*pi/h;
```

```
int ny = r*2*pi/h;
func torox= (R+r*cos(y*pi*2))*cos(x*pi*2);
func toroy= (R+r*cos(y*pi*2))*sin(x*pi*2);
func toroz= r*sin(y*pi*2);
meshS ThS=square3(nx,ny,[torox,toroy,toroz]) ;
mesh3 Th3=tetg(ThS,switch="paAAQYY"); //,nbofregions=1,regionlist=domain);
plot(Th3,wait=1);
```

Al ejecutar `MallaSuper3D.edp` con FreeFEM obtenemos una salida gráfica como la mostrada en la figura 1.13.

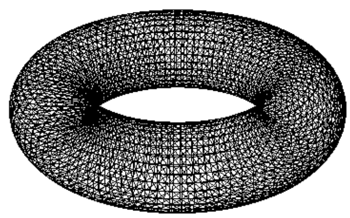

Figura 1.13: Malla superficial $\mathcal{T}_h$ del material $\mathcal{T}$.

Capítulo 2

Aproximación de Deformaciones Estáticas en Mecánica Estructural

Objetivos

1. Deducir e Interpretar los modelos genéricos de Navier para la predicción de la deformación de materiales lineales.
2. Clasificar modelos computacionales estructurales sólidos en términos de sus características de deformación.
3. Identificar el modelo computacional que mejor describe la deformación estática de un elemento estructural dado.
4. Calcular numéricamente de forma eficiente la deformación estaática aproximada de un elemento estructural dado utilizando FreeFem++ y GNU Octave.
5. Calcular numéricamente de forma eficiente la deformación estaática aproximada de un elemento estructural dado utilizando CalculiX y FreeCAD.

2.1. Modelos de Deformación de Navier

Consideremos un medio continuo $\mathcal{E} \subseteq \mathbf{R}^n$ para $n = 1, 2$. Dado un material $\mathcal{M} \subseteq \mathcal{E}$, una deformación estática $\mathcal{M}$ en $\mathcal{E}$ es una función continua $\mathcal{D} : \mathcal{M} \to \mathcal{E}$, determinada en términos de sus coordenadas como $x_j = \mathcal{D}_j(X_1, \ldots, X_n)$, $1 \leq j \leq n$ para $n = 2, 3$. Las coordenadas x_j se denominan coordenadas espaciales, y las coordenadas X_j se denominan coordenadas materiales.

Ejemplo: A manera de ejemplo en un medio continuo $\mathcal{E} \subseteq \mathbf{R}^3$ podemos considerar una viga de perfil I en voladizo como la descrita en la figura 2.1.

Utilizando técnicas matriciales de cómputo de deformación podemos calcular una aproximación $(\tilde{x}_1, \tilde{x}_2, \tilde{x}_3) = \tilde{\mathcal{D}}(X_1, X_2, X_3)$ de una deformación arbitraria $(x_1, x_2, x_3) = \mathcal{D}(X_1, X_2, X_3)$ del arreglo de vigas, obteniendo entre otras, transformaciones continuas como las mostradas en la figura 2.2.

2.1.1. Modelos Matriciales de Navier

Dados $T > 0$ y una deformación $\mathcal{D} : \mathcal{M} \times [0, T] \to \mathcal{E}$ de un material $\mathcal{M}$ en un medio continuo tridimensional $\mathcal{E}$, aplicando hipótesis de elasticidad lineal, tendremos que la deformación $\mathcal{D}$ puede

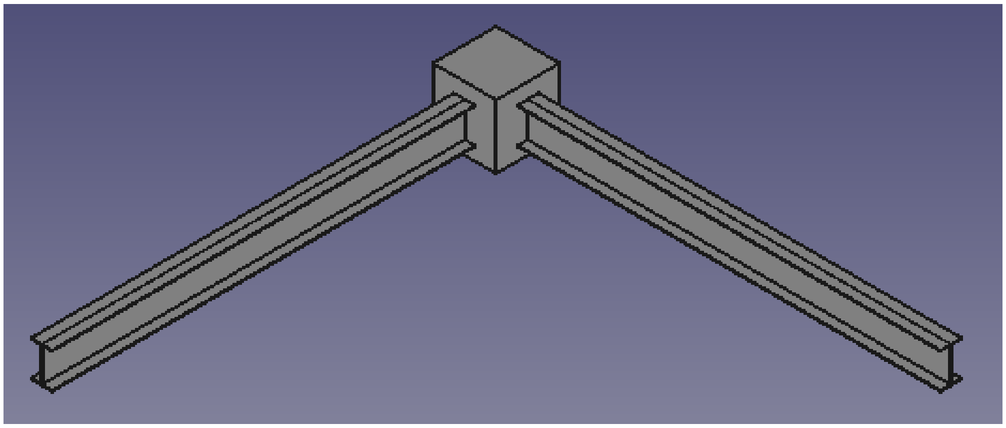

Figura 2.1: Arreglo de vigas de perfil I.

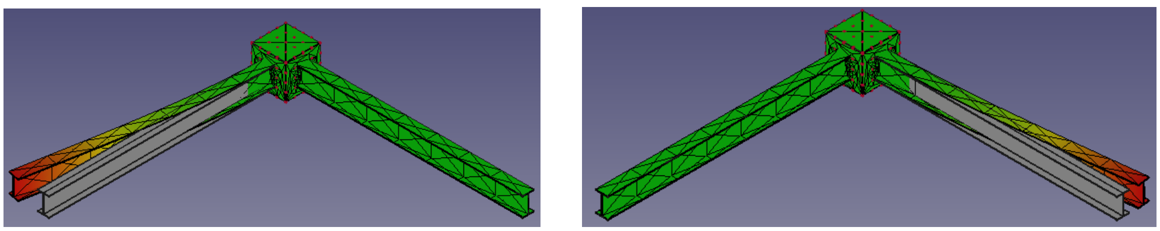

Figura 2.2: Deformaciones estáticas aproximadas del arreglo de vigas de perfil I.

describirse en la forma:

$$\mathcal{D}(\mathbf{X}, t) = \mathbf{X} + \mathbf{u}(\mathbf{X}, t) \tag{2.1}$$

para un tiempo t arbitrario en el intervalo $[0, T]$, con $\mathbf{u} \approx \mathbf{0}$. Las hipótesis antes mencionadas implican que los gradientes de deformación material $\mathbf{F}$ y de deformación espacial $\mathbf{F}^\dagger$ determinados por las ecuaciones

$$\begin{aligned}
\mathbf{F} &= \frac{\partial \mathbf{x}}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial x_1}{\partial X_1} & \frac{\partial x_1}{\partial X_2} & \frac{\partial x_1}{\partial X_3} \\ \frac{\partial x_2}{\partial X_1} & \frac{\partial x_2}{\partial X_2} & \frac{\partial x_2}{\partial X_3} \\ \frac{\partial x_3}{\partial X_1} & \frac{\partial x_3}{\partial X_2} & \frac{\partial x_3}{\partial X_3} \end{bmatrix} \\
\mathbf{F}^\dagger &= \mathbf{F}^{-1}
\end{aligned}$$

satisfacen las condiciones

$$\mathbf{F} = \mathbf{1}_3 = \mathbf{F}^\dagger$$

donde $\mathbf{1}_3$ es el tensor identidad determinado por la expresión.

$$\mathbf{1}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Con base en las ecuaciones e hipótesis, al calcular el gradiente de la deformación descrita por la ecuación (2.1) obtenemos la siguiente regla de transformación.

$$\mathbf{F} = \frac{\partial \mathcal{D}(\mathbf{X}, t)}{\partial \mathbf{X}} = \frac{\partial \mathbf{X}}{\partial \mathbf{X}} + \frac{\partial \mathbf{u}(\mathbf{X}, t)}{\partial \mathbf{X}} = \mathbf{1}_3 + \mathbf{G} \tag{2.2}$$

donde G es el gradiente de desplazamiento determinado por la ecuación.

$$\mathbf{G} = \frac{\partial \mathbf{u}(\mathbf{X},t)}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial u_1}{\partial X_1} & \frac{\partial u_1}{\partial X_2} & \frac{\partial u_1}{\partial X_3} \\ \frac{\partial u_2}{\partial X_1} & \frac{\partial u_2}{\partial X_2} & \frac{\partial u_2}{\partial X_3} \\ \frac{\partial u_3}{\partial X_1} & \frac{\partial u_3}{\partial X_2} & \frac{\partial u_3}{\partial X_3} \end{bmatrix} \tag{2.3}$$

Aplicando nuevamente las hipótesis de elasticidad lineal tendremos que los tensores materiales y espaciales de deformación colapsan aproximadamente a una representación de la forma

$$\varepsilon(\mathbf{X},t) = \frac{1}{2}\left(\mathbf{G} + \mathbf{G}^\top\right) \tag{2.4}$$

donde $\mathbf{G}$ es el gradiente de desplazamiento determinado por (2.3). El tensor $\varepsilon(\mathbf{X},t)$ se denomina tensor infinitesimal de deformación.

Si además de las hipótesis de elasticidad lineal añadimos hipótesis de isotropía al material $\mathcal{M}$ en estudio, al aplicar simetrías mecánicas correspondientes al tensor de constantes elásticas $\mathcal{C}$ que resuelve el problema de conversión determinado por la ley generalizada de Hooke

$$\sigma = \mathcal{C} : \varepsilon \tag{2.5}$$

para el tensor de tensión $\sigma(\mathbf{X},t)$, tendremos que $\sigma(\mathbf{X},t)$ puede calcularse utilizando la ecuación.

$$\sigma = \lambda \mathrm{tr}(\varepsilon)\mathbf{1}_3 + 2\mu\varepsilon \tag{2.6}$$

donde

$$\begin{cases} \lambda = \frac{\nu E}{(1+\nu)(1-2\nu)} \\ \mu = \frac{E}{2(1+\nu)} \end{cases} \tag{2.7}$$

son las constantes de Lamé del material $\mathcal{M}$ determinadas por el módulo de Elasticidad (de Young) E y la taza de Poisson ν de $\mathcal{M}$.

Con base en la ley de conversión (2.6) tendremos que la ecuación de Cauchy para el balance de la cantidad de movimiento toma la forma.

$$\nabla \cdot \sigma(\mathbf{x},t) + \rho_0 \mathbf{b}(\mathbf{x},t) = \rho_0 \partial_t^2 \mathbf{u}(\mathbf{x},t) \tag{2.8}$$

Al sustituir la ecuación (2.6) en (2.8) obtenemos la ecuación de Navier para el desplazamiento $\mathbf{u}(\mathbf{x},t)$ del material $\mathcal{M}$, la cual estará determinada por la expresión.

$$\begin{cases} (\lambda+\mu)\nabla(\nabla\cdot\mathbf{u}) + \mu\nabla^2\mathbf{u} + \rho_0\mathbf{b} = \rho_0\partial_t^2\mathbf{u}(\mathbf{x},t) \\ \mathbf{u}(\mathbf{x},t) = \mathbf{u}^*(\mathbf{x},t), \mathbf{x} \in \partial\mathcal{M} \\ \mathbf{u}(\mathbf{x},0) = u_0(\mathbf{x}) \\ \partial_t\mathbf{u}(\mathbf{x},t) = u_1(\mathbf{x}) \end{cases} \tag{2.9}$$

Para llevar a cabo el análisis matricial del material $\mathcal{M}$ calculamos la matriz (estructural) de Navier $\mathbf{N}_{\lambda,\mu}$ determinada por las ecuaciones:

$$\mathbf{N}_{\lambda,\mu} = (\lambda+\mu)\mathbf{L}_h + \mu\mathbf{K}_h \tag{2.10}$$

donde $\mathbf{L}_h \approx_h \nabla\nabla\cdot$ y $\mathbf{K}_h \approx_h \nabla^2$ son aproximaciones en diferencias finitas de los operadores correspondientes en (2.9). Si denotamos por $\mathbf{b}_h$ la representación del vector de fuerzas másicas $\mathbf{b}$ en la malla de diferencias finitas $\mathcal{M}_h$ del material $\mathcal{M}$, la ecuación (2.9) puede representarse aproximadamente por la expresión.

$$\mathbf{N}_{\lambda,\mu}\mathbf{u}_h + \rho_0\mathbf{b}_h = \rho_0\frac{d\mathbf{u}_h}{dt^2} \tag{2.11}$$

2.2. Método de Elementos Finitos para el Cómputo de Deformaciones Mecánicas

Como ya lo hemos observado los bojetos sólidos se deforman bajo la acción de fuerzas aplicadas a ellos: Bajo estas acciones, un punto en un material $\mathcal{M}$ en un medio continuo $\mathcal{E}$, localizado originalmente en (x, y, z) se convierte en (X, Y, Z) luego de cierto tiempo, el vector de desplazamiento estará dado por la fórmula $\mathbf{u} = (u_1, u_2, u_3) = (X - x, Y - y, Z - z)$. Cuando $\mathbf{u} \approx 0$ tal como se apreció anteriormente,

$$\sigma_{ij}(\mathbf{u}) = (\lambda \nabla \cdot \mathbf{u})\mathbf{1} + 2\mu\varepsilon(\mathbf{u}) \tag{2.12}$$

donde $\varepsilon(\mathbf{u}) = (1/2)(\mathbf{F} + \mathbf{F}^\top)$, para λ, μ definidas en (2.7).

Consideremos nuevamente la ecuación (2.8) de control de deformación de un material $\mathcal{M}$ en un medio continuo $\mathcal{E}$. Utilizando teoría de distribuciones podemos reformular la ecuación (2.8) de forma débil obtienendo la expresión,

$$\int_{\mathcal{M}} \sigma(\mathbf{u}) : \varepsilon(\mathbf{v})\; dx + \int_{\mathcal{M}} \mathbf{v} \cdot \mathbf{b}\; dx = 0; \tag{2.13}$$

donde : denota el producto escalar de tensores definido por $\mathbf{a} : \mathbf{b} = \sum_{i,j} a_{ij} b_{ij}$. Tenemos que la forma variacional de (2.13) estará dada por la expresión.

$$\int_{\mathcal{M}} \lambda \nabla \cdot \mathbf{u} \nabla \cdot \mathbf{v} + 2\mu\varepsilon(\mathbf{u}) : \varepsilon(\mathbf{v})\; dx + \int_{\mathcal{M}} \mathbf{v} \cdot \mathbf{b}\; dx = 0; \tag{2.14}$$

Para llevar a cabo el análisis matricial del material $\mathcal{M}$, consideramos una malla $\mathcal{M}_h \subseteq \mathcal{M}$ del material $\mathcal{M}$ determinada por una colección de puntos de referencia en el material y por un grafo que determina la conectividad entre estos puntos, luego calculamos la matriz (estructural) de rigidez (de Navier) $\mathbf{K}_{\lambda,\mu}$ determinada por las ecuaciones:

$$\mathbf{K}_{\lambda,\mu} = (\mathcal{A}(\hat{\mathbf{u}}_j, \hat{\mathbf{u}}_k)) \tag{2.15}$$

determinado por la forma variacional discretizada

$$\mathcal{A}(\hat{\mathbf{u}}_j, \hat{\mathbf{u}}_k) = \int_{\mathcal{M}_h} \lambda \nabla \cdot \hat{\mathbf{u}}_j \nabla \cdot \hat{\mathbf{u}}_k + 2\mu\varepsilon(\hat{\mathbf{u}}_j) : \varepsilon(\hat{\mathbf{u}}_k)\; dx \tag{2.16}$$

donde las funciones $\hat{\mathbf{u}}_1, \ldots, \hat{\mathbf{u}}_{M_V}$ son elementos genéricos del conjunto funciones de cálculo y de prueba, en los espacios de análisis computacional de elementos finitos $\mathcal{V}_h$ definidos por la expresión

$$\mathcal{V}_h = \{\mathbf{u}_h |\; \mathbf{u}_h = u_1\hat{\mathbf{u}}_1 + \cdots + u_{M_V}\hat{\mathbf{u}}_{M_V}\} \tag{2.17}$$

y que están determinados por la malla $\mathcal{M}_h$ del material $\mathcal{M}$. Si denotamos por $\mathbf{b}_h$ la representación del vector de cargas $\mathbf{b}$ en la malla de elementos finitos $\mathcal{M}_h$ del material $\mathcal{M}$, la ecuación (2.9) puede representarse aproximadamente por la expresión.

$$\mathbf{K}_{\lambda,\mu}\mathbf{u}_h + \rho_0\mathbf{b}_h = \rho_0\mathbf{M}\frac{d\mathbf{u}_h}{dt^2} \tag{2.18}$$

donde el vector $\mathbf{b}_h$ y la matrix de masa $\mathbf{M}$, están determinados por las ecuaciones.

$$\left\{ \begin{array}{l} \mathbf{b}_h = (\mathcal{F}(\hat{\mathbf{u}}_j)), \\ \mathbf{M} = (\mathcal{I}(\hat{\mathbf{u}}_j, \hat{\mathbf{u}}_k)) \end{array} \right. \tag{2.19}$$

y donde las formas variacionales $\mathcal{F}$ e $\mathcal{I}$ están determinadas por las ecuaciones.

$$\begin{aligned} \mathcal{F}(\hat{\mathbf{u}}_j) &= \int_{\mathcal{M}_h} \hat{\mathbf{u}}_j \cdot \mathbf{b}\, dx \\ \mathcal{I}(\hat{\mathbf{u}}_j, \hat{\mathbf{u}}_k) &= \int_{\mathcal{M}_h} \hat{\mathbf{u}}_j \cdot \hat{\mathbf{u}}_k\, dx \end{aligned} \tag{2.20}$$

2.2.1. Cómputo de Deflexión Estática de Elementos Estructurales con Elementos Finitos

Para calcular la deformación estática de un material $\tilde{\mathcal{M}}$ cuyas características mecánicas estructurales son representadas aproximadamente por una ecuación de la forma (2.11), bajo la hipótesis de que $\partial_t \mathbf{u}_h = \mathbf{0} = \partial_t^2 \mathbf{u}_h$, basta resolver el sistema de ecuaciones lineales.

$$\mathbf{N}_{\lambda,\mu}\mathbf{u}_h = -\rho_0 \mathbf{b}_h \tag{2.21}$$

Cómputo de Deflexión de una viga Sólida en Voladizo

Consideremos una viga 3D en voladizo de $5 \times 1 \times 1\ m^3$ cuyos coeficientes mecánicos se especifican más adelante. Consideraremos la fuerza másica correspondiente a la gravedad como la uúnica fuerza actuando sobre el cuerpo sólido.

Podemos calcular la deflexión de la viga utilizando FreeFem++ y GNU Octave trabajando de forma combinada con los programas `Beam3D.edp` y `Beam3D.m` descritos a continuación.

Programa FreeFem++ `Beam3D.edp`:

```
include "cube.idp"
include "ffmatlib.idp"

//Parametros
int[int] Nxyz = [20, 5, 5];
real [int, int] Bxyz = [[0., 5.], [0., 1.], [0., 1.]];
int [int, int] Lxyz = [[1, 2], [2, 2], [2, 2]];

real E = 21.5e4;
real sigma = 0.29;
real gravity = -0.05;

// Mallado
mesh3 Th = Cube(Nxyz, Bxyz, Lxyz);

// Funciones re resolucion y de de prueba
fespace Vhs(Th,P1);
Vhs u,ux,uy,uz;

fespace Vh(Th, [P1, P1, P1]);
Vh [u1, u2, u3];
Vh [v1, v2, v3];

// Macros
```

```
real sqrt2 = sqrt(2.);
macro epsilon(u1, u2, u3) [
dx(u1), dy(u2), dz(u3),
(dz(u2) + dy(u3))/sqrt2,
(dz(u1) + dx(u3))/sqrt2,
(dy(u1) + dx(u2))/sqrt2] //
macro div(u1, u2, u3) (dx(u1) + dy(u2) + dz(u3)) //

// Coeficientes mecanicos

real mu = E/(2*(1+sigma));
real lambda = E* sigma/((1+sigma)*(1-2* sigma));

// Soluci\'on del problema de deflexion estructural

solve Lame ([u1, u2, u3], [v1, v2, v3])
= int3d(Th)(
lambda*div(u1, u2, u3)*div(v1, v2, v3)
+ 2.*mu*( epsilon(u1, u2, u3)'*epsilon(v1, v2, v3) )
)
- int3d(Th)(
gravity*v3
)
+ on(1, u1=0, u2=0, u3=0)
;

// Calculo del desplazamiento absoluto
u=sqrt(u1*u1+u2*u2+u3*u3);
ux=u1;
uy=u2;
uz=u3;

// Visualizacion de Resultados
real dmax = u[].max;

// Visualizacion de mallas de referencia y deformacion

real coef = 0.3/dmax;
int[int] ref2 = [1, 0, 2, 0];
mesh3 Thm = movemesh3(Th, transfo=[x+u1*coef,y+u2*coef,z+u3*coef],label=ref2);
Thm = change(Thm, label=ref2);

plot(Th, Thm, wait=true, cmm="coef amplification = "+coef);

cout << endl;
cout << "max displacement = " << dmax << endl;
cout << endl;
```

```
// Almacenamiento de resultados para visualizacion en GNU Octave

savemesh(Th,"beam3d.mesh");
savemesh(Thm,"beam3d_def.mesh");
ffSaveVh(Th,Vhs,"beam3dvh.txt");
ffSaveData(u,"beam3dpot.txt");
ffSaveData3(ux,uy,uz,"beam3dvec.txt");
```

Programa GNU Octave `Beam3D.m`:

```
% 3D beam deformation problem
%
% Author: F. Vdies <fredy.vides@unah.edu.hn>
% Created: 2019-08-03
%
% Copyright (C) 2019
%
% This program is free software: you can redistribute it and/or modify it
% under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% This program is distributed in the hope that it will be useful, but
% WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with this program.  If not, see
% <https://www.gnu.org/licenses/>.
%

clear all;
addpath('ffmatlib');

[p,b,t,nv,nbe,nt,labels]=ffreadmesh('beam3d.mesh');
[p_def,b_def,t_def,nv_def,nbe_def,nt_def,labels_def]=ffreadmesh('beam3d_def.mesh');
[vh]=ffreaddata('beam3dvh.txt');
[u]=ffreaddata('beam3dpot.txt');
[Ex,Ey,Ez]=ffreaddata('beam3dvec.txt');

subplot(211);

ylabel('y');
xlabel('x');
zlabel('z');

ffpdeplot3D(p,b,t,'VhSeq',vh,...
```

```
'XYZStyle','monochrome');
shading interp;
lighting gouraud;
camlight('headlight');

axis equal;

subplot(212);

ylabel('y');
xlabel('x');
zlabel('z');

ffpdeplot3D(p_def,b_def,t_def,'VhSeq',vh,...
'XYZData',u,'ColorMap',jet,...
'ColorBar','on','BoundingBox','on',...
'Mesh','on');
shading interp;
lighting gouraud;
camlight('headlight');

axis equal;
```

El procedimiento computacional de cómputo es el siguiente:

1. Ejecutar `Beam3D.edp` con FreeFem++.

   - La salida gráfica principal se muestra en la figura 2.3.

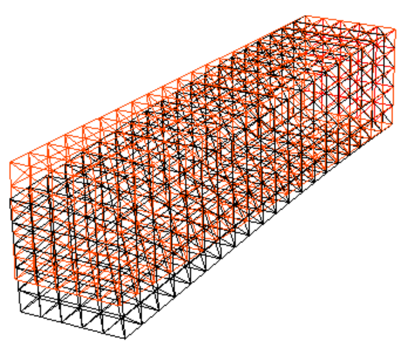

Figura 2.3: Aproximación de la deformación estática de la viga 3D en voladizo

2. Ejecutar `Beam3D.m` con GNU Octave.

   - La salida gráfica principal se muestra en la figura 2.4.

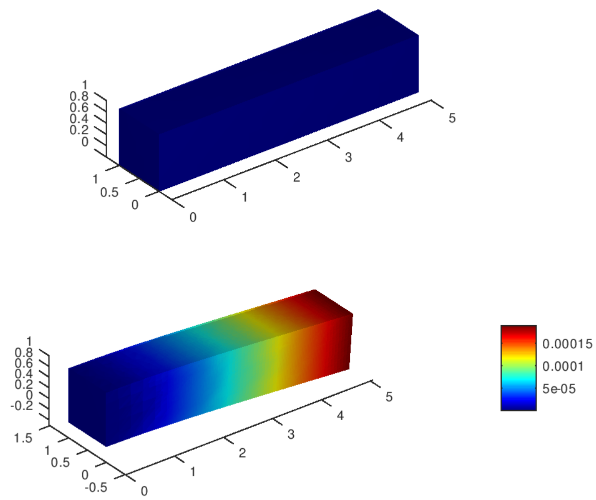

Figura 2.4: Aproximación de la deformación estática de la viga 3D en voladizo

2.2.2. Cómputo de Respuesta Mecánica de Elementos Estructurales con Elementos Finitos

Para calcular la respuesta mecánica de un material $\tilde{\mathcal{M}}$ cuyas características mecánicas estructurales son representadas aproximadamente por una ecuación de la forma (2.11), bajo la hipótesis de equilibrio de cargas (se omite peso propio y cargas externas del elemento extructural) y de que $\mathbf{u}_h(t) = e^{i\omega t}\hat{\mathbf{u}}_h$, basta resolver el problema de valores propios.

$$\mathbf{N}_{\lambda,\mu}\hat{\mathbf{u}}_{k,h} = -\rho_0\omega_k^2\hat{\mathbf{u}}_{k,h} = \alpha_k\hat{\mathbf{u}}_{k,h}, \;\; 1 \leq k \leq N \tag{2.22}$$

para algún entero positivo N. Las frecuencias naturales de respuesta mecánica ω_j y los periodos correspondientes están determinados por las fórmulas:

$$\begin{aligned} \omega_j &= \sqrt{\frac{|\lambda_j|}{\rho_0}}, \\ T_j &= \frac{2\pi}{\omega_j} \end{aligned}$$

Cómputo de Deflexión de una Reducción de Orden Plana de una Viga Sólida Doblemente Apoyada

Consideremos una reducción de orden en 2D de una viga sólida doblemente apoyada de $5 \times 1 \times 1\ m^3$ cuyos coeficientes mecánicos se especifican más adelante. Consideraremos la fuerza másica correspondiente a la gravedad como la uúnica fuerza actuando sobre el sub-cuerpo plano resultante.

Podemos calcular la deflexión de la viga utilizando FreeFem++ y GNU Octave trabajando de forma combinada con los programas `Beam2D.edp` y `Beam2D.m` descritos a continuación.

Programa FreeFem++ `Beam2D.edp`:

```
include "ffmatlib.idp"
```

```
// Parámetros
real E = 21e5;
real nu = 0.28;

real f = -1;

// Mallado
mesh Th = square(10, 10, [20*x,2*y-1]);

// Definición de funciones de resolución y funciones prueba.
fespace Vh(Th, P2);
Vh u, v;
Vh uu, vv;
Vh w;

// Macros
real sqrt2=sqrt(2.);
macro epsilon(u1,u2) [dx(u1),dy(u2),(dy(u1)+dx(u2))/sqrt2] //

macro div(u,v) ( dx(u)+dy(v) ) //

// Coeficientes mecánicos
real mu= E/(2*(1+nu));
real lambda = E*nu/((1+nu)*(1-2*nu));

// Resolución de problema de deflexión estructural
solve lame([u, v], [uu, vv])
   = int2d(Th)(
        lambda * div(u, v) * div(uu, vv)
      + 2.*mu * ( epsilon(u,v)' * epsilon(uu,vv) )
   )
   - int2d(Th)(
        f*vv
   )
   + on(4, u=0, v=0)+on(2, u=0, v=0)
   ;

// Visulaización
real coef=4000;
plot([u, v], wait=1, ps="lamevect.eps", coef=coef);

// Cálculo de malla de deformacion
mesh th1 = movemesh(Th, [x+u*coef, y+v*coef]);
plot(th1,wait=1,ps="lamedeform.eps");

// Salidas
real dxmin = u[].min;
real dymin = v[].min;
```

```
cout << " - dep. max x = "<< dxmin << " y=" << dymin << endl;
cout << "   dep. (20, 0) = " << u(20, 0) << " " << v(20, 0) << endl;

w=sqrt(u*u+v*v);


// Almacenamiento de resultados para visualización en Octave

savemesh(Th,"beam_2d.msh");
savemesh(th1,"beam_2d_def.msh");
ffSaveVh(Th,Vh,"beam_vh_2d.txt");
ffSaveData3(w,u,v,"beam_data_2d.txt");
```

Programa GNU Octave `Beam2D.m`:

```
% 2D beam eigenfrequency computation problem
%
% Author: Fredy Vides <fredy.vides@unah.edu.hn>
% Created: 2019-08-03
%
% Copyright (C) 2019 Fredy Vides
%
% This program is free software: you can redistribute it and/or modify it
% under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% This program is distributed in the hope that it will be useful, but
% WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with this program.  If not, see
% <https://www.gnu.org/licenses/>.
%

clear all;

addpath('ffmatlib');

[p,b,t,nv,nbe,nt,labels]=ffreadmesh('beam_2d.msh');
[p_def,b_def,t_def,n_def,nbe_def,nt_def,labels_def]=ffreadmesh('beam_2d_def.msh');
[vh]=ffreaddata('beam_vh_2d.txt');
[u,Ex,Ey]=ffreaddata('beam_data_2d.txt');
```

```
subplot(211);
ffpdeplot(p,b,t, ...
          'VhSeq',vh, ...
          'XYData',u, ...
          'Mesh','off', ...
  'ColorMap',jet,...
          'Boundary','on', ...
          'XLim',[0 25],'YLim',[-2.5 2.5], ...
          'CBTitle','U[V]', ...
          'Title','2D Patch Plot (Desplazamiento Absoluto)');

ylabel('y');
xlabel('x');
axis tight;
subplot(212);

ffpdeplot(p_def,b_def,t_def, ...
          'VhSeq',vh, ...
          'XYData',u, ...
          'Mesh','off', ...
  'ColorMap',jet,...
          'Boundary','on', ...
          'XLim',[0 25],'YLim',[-4 4], ...
          'CBTitle','U[V]', ...
          'Title','2D Patch Plot (Desplazamiento Absoluto)');
axis tight;
ylabel('y');
xlabel('x');



figure;
subplot(211);
ffpdeplot(p,b,t, ...
          'Mesh','on', ...
          'Boundary','on', ...
          'Title','Contorno y malla de la viga en 2D');

ylabel('y');
xlabel('x');
subplot(212);
ffpdeplot(p_def,b_def,t_def, ...
          'Mesh','on', ...
          'Boundary','on', ...
          'Title','Contorno y malla de la viga en 2D deformada');

ylabel('y');
xlabel('x');
```

```
axis tight;
```

El procedimiento computacional de cómputo es el siguiente:

1. Ejecutar `Beam2D.m` con FreeFem++.
   - La salida gráfica principal se muestra en la figura 2.5.

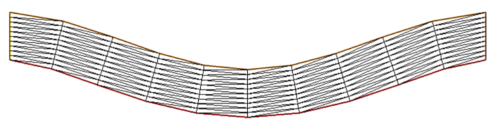

Figura 2.5: Aproximación de orden reducido plano de la deformación estática de la viga 3D doblemente apoyada

2. Ejecutar `Beam2D.m` con GNU Octave.
   - La salida gráfica principal se muestra en la figura 2.6.

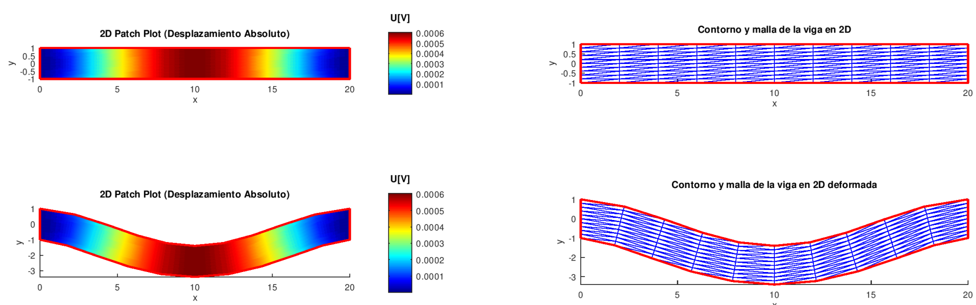

Figura 2.6: Aproximación de orden reducido plano de la deformación estática de la viga 3D doblemente apoyada

Cómputo de Respuetas Mecánicas de una Reducciñ de Orden Plana de una Viga Sólida Doblemente Apoyada

Consideremos una reducción de orden en 2D de una viga sólida doblemente apoyada de $2 \times 0{,}4 \times 0{,}4\ m^3$ cuyos coeficientes mecánicos se especifican más adelante. Consideraremos la fuerza másica correspondiente a la gravedad como la uúnica fuerza actuando sobre el sub-cuerpo plano resultante.

Podemos calcular respuestas mecánicas de la viga de orden reducido de $2 \times 0{,}4\ m^2$ utilizando FreeFem++ y GNU Octave trabajando de forma combinada con los programas `EigBeam2D.edp` y `EigBeam2D.m` descritos a continuación.

Programa FreeFem++ `EigBeam2D.edp`:

```
include "ffmatlib.idp"

// Definición de geometría del problema

verbosity=1;
int bottombeam = 2;
border aaa(t=0.4,0)  { x=0; y=t ;label=1;};        //  borde izquierdo
border bbb(t=0,2) { x=t; y=0 ;label=bottombeam;};//  borde inferior
border ccc(t=0,0.4)  { x=2; y=t ;label=1;};       //  borde derecho
border ddd(t=0,2) { x=2-t; y=0.4; label=3;};     //  borde superior

// Definición de coeficientes mecánicos

real E = 20e5;
real sigma = 0.3;
real mu = E/(2*(1+sigma));
real lambda = E*sigma/((1+sigma)*(1-2*sigma));
real gravity = -0.05;

// Mallado

mesh Th = buildmesh( bbb(20)+ccc(5)+ddd(20)+aaa(5));

// Definición de funciones de cómputo y de prueba

fespace Vh(Th,[P1,P1]);
Vh [uu,vv], [w,s];



cout << "lambda,mu,gravity ="<<lambda<< " " << mu << " " << gravity << endl;

real shift = 0;

//Definición de forma variacional del problema

varf a([uu,vv],[w,s])=
int2d(Th)(
2*mu*(dx(uu)*dx(w)+dy(vv)*dy(s)+ ((dx(vv)+dy(uu))*(dx(s)+dy(w)))/2 )
               + lambda*(dx(uu)+dy(vv))*(dx(w)+dy(s))
        - shift* (uu*w + vv*s)
             )
  + on(1,uu=0,vv=0);

varf b([uu,vv],[w,s])=
int2d(Th)(uu*w + vv*s);

// Cómputo de matrices estructurales
```

```
matrix A= a(Vh,Vh,solver=UMFPACK);
matrix B= b(Vh,Vh,solver=CG,eps=1e-20);

// Selección del número de respuestas mecánicas

int nev=20;

// Cómputo de respuestas mecánicas

real[int] ev(nev);
Vh[int] [eV,eW](nev);


int k=EigenValue(A,B,sym=true,sigma=sigma,
value=ev,vector=eV,tol=1e-10,maxit=0,ncv=0);

k=min(k,nev);

// Visualización y almacenamiento de resultados

mesh th1;

savemesh(Th,"eig_beam_2d.msh");
ffSaveVh(Th,Vh,"eig_beam_vh_2d.txt");

real coef=1e-2;

for (int i=0;i<k;i++)
{
  [uu,vv]=[eV[i],eW[i]];
  th1 = movemesh(Th, [x+coef*uu, y+coef*vv]);
  plot(th1, wait=true);
  savemesh(th1,"eig_beam_2d_def"+i+".msh");
  ffSaveData3(uu,uu,vv,"eig_beam_data"+i+"_2d.txt");
}
```

Programa GNU Octave `EigBeam2D.m`:

```
% 2D beam eigenfrequency computation problem
%
% Author: Fredy Vides <fredy.vides@unah.edu.hn>
% Created: 2019-08-03
%
% Copyright (C) 2019 Fredy Vides
%
% This program is free software: you can redistribute it and/or modify it
% under the terms of the GNU General Public License as published by
```

```
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% This program is distributed in the hope that it will be useful, but
% WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with this program.  If not, see
% <https://www.gnu.org/licenses/>.
%

%clear all;

addpath('ffmatlib');

[p,b,t,nv,nbe,nt,labels]=ffreadmesh('eig_beam_2d.msh');
[vh]=ffreaddata('eig_beam_vh_2d.txt');

for k=0:19,
[p_def,b_def,t_def,n_def,nbe_def,nt_def,labels_def]=
ffreadmesh(['eig_beam_2d_def' num2str(k) '.msh']);
[u,Ex,Ey]=ffreaddata(['eig_beam_data' num2str(k) '_2d.txt']);

figure;

subplot(211);
ffpdeplot(p,b,t, ...
          'VhSeq',vh, ...
          'XYData',sqrt(Ex.^2+Ey.^2), ...
          'Mesh','off', ...
  'ColorMap',jet,...
          'Boundary','on', ...
          'XLim',[0 10],'YLim',[-2.5 2.5], ...
          'CBTitle','U[V]', ...
          'Title','2D Patch Plot (Desplazamiento Absoluto)');

ylabel('y');
xlabel('x');
axis equal;
subplot(212);

ffpdeplot(p_def,b_def,t_def, ...
          'VhSeq',vh, ...
          'XYData',sqrt(Ex.^2+Ey.^2), ...
          'Mesh','off', ...
  'ColorMap',jet,...
```

```
          'Boundary','on', ...
          'XLim',[0 10],'YLim',[-2 2], ...
          'CBTitle','U[V]', ...
          'Title','2D Patch Plot (Desplazamiento Absoluto)');
axis equal;
ylabel('y');
xlabel('x');



figure;
subplot(211);
ffpdeplot(p,b,t, ...
          'Mesh','on', ...
          'Boundary','on', ...
          'Title','Contorno y malla de la viga en 2D');

ylabel('y');
xlabel('x');
subplot(212);
ffpdeplot(p_def,b_def,t_def, ...
          'Mesh','on', ...
          'Boundary','on', ...
          'Title','Contorno y malla de la viga en 2D deformada');

ylabel('y');
xlabel('x');

axis tight;
endfor
```

El procedimiento computacional de cómputo es el siguiente:

1. Ejecutar `EigBeam2D.m` con FreeFem++.
    - La salida gráfica principal se muestra en la figura 2.7.

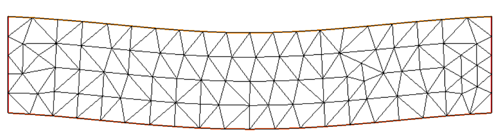

Figura 2.7: Aproximación de orden reducido plano de la respuesta mecánica de más baja frecuencia de la viga 3D doblemente apoyada

2. Ejecutar `Beam2D.m` con GNU Octave.
    - La salida gráfica principal se muestra en la figura 2.8.

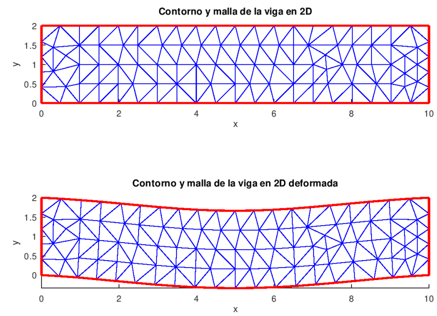

Figura 2.8: Aproximación de orden reducido plano de la respuesta mecánica de más baja frecuencia de la viga 3D doblemente apoyada

Cómputo de Respuesta Mecánica de una Viga de Concreto

Consideremos una viga de concreto genérico en voladizo de $2 \times 0{,}4 \times 0{,}4\ m^3$, apoyada en la cara donde el plano $x = 0$ interseca a la viga. Bajo hipótesis de equilibrio de cargas (omitiendo peso propio y cualquier carga estructural) por simplicidad de este ejemplo, y suponiendo además que la deformación de la viga está controlada por su componente de concreto, podemos calcular la respuesa mecánica de más baja frecuencia de este elemento estructural utilizando el siguiente código FreeFem que podemos definir con el nombre `EigBeam3D.edp`.

- **Programa FreeFem++ `EigBeam3D.edp`:**

```
include "cube.idp"
include "ffmatlib.idp"

//Parámetros
int[int] Nxyz = [20, 20, 20];
real [int, int] Bxyz = [[0., 2], [0., .4], [0., .4]];
int [int, int] Lxyz = [[1, 2], [2, 2], [2, 2]];

real E = 32000;
real sigma = .17;
real mu = E/(2*(1+sigma));
real lambda = E*sigma/((1+sigma)*(1-2*sigma));
real gravity = -0.05;

real shift = 0;

// Mallado
mesh3 Th = Cube(Nxyz, Bxyz, Lxyz);

// Cómputo de funciones de prueba
fespace Vhs(Th,P1);
Vhs u;
```

```
fespace Vh(Th, [P1, P1, P1]);
Vh [ux, uy, uz];
Vh [vx, vy, vz];
Vh [uu,vv,ww];



//Macros
real sqrt2 = sqrt(2.);
macro Epsilon(ux, uy, uz) [dx(ux), dy(uy), dz(uz),
(dz(uy)+dy(uz))/sqrt2,
(dz(ux)+dx(uz))/sqrt2,
(dy(ux)+dx(uy))/sqrt2] //
macro Divergence(ux, uy, uz) (dx(ux) + dy(uy) + dz(uz)) //

//Planteamiento del Problema Variacional de Deflexión Estática

varf A ([ux, uy, uz], [vx, vy, vz])
= int3d(Th)(
  lambda * Divergence(vx, vy, vz) * Divergence(ux, uy, uz)
+ 2. * mu * (
  Epsilon(vx, vy, vz)' * Epsilon(ux, uy, uz)
               - shift* (ux*vx + uy*vy+ uz*vz)
)
)
+ on(1, ux=0, uy=0, uz=0)
;

// Definición de matriz estructural y vector de cargas

matrix K = A(Vh, Vh, solver=sparsesolver);

varf m([ux,uy,uz],[vx,vy,vz])=
int3d(Th)(ux*vx + uy*vy+uz*vz);

matrix M= m(Vh,Vh,solver=CG,eps=1e-20);

int nev=1;

// Cómputo de respuestas mecánicas

real[int] ev(nev);
Vh[int] [eVx,eVy,eVz](nev);


int k=EigenValue(K,M,sym=true,sigma=sigma,
value=ev,vector=eVx,tol=1e-10,maxit=0,ncv=0);
```

```
k=min(k,nev);

// Visualización y almacenamiento de resultados

mesh3 th1;


savemesh(Th,"EigBeam3d.mesh");
ffSaveVh(Th,Vh,"Eigbeamvh3d.txt");

real coef=1e-1;

u=sqrt(uu*uu+vv*vv+ww*ww);

for (int i=0;i<k;i++)
{
  [uu,vv,ww]=[eVx[i],eVy[i],eVz[i]];
  th1 = movemesh(Th, [x+coef*uu, y+coef*vv,z+coef*ww]);
  plot(th1,Th,value=true,fill=true, wait=true);
  savemesh(th1,"EigBeam3ddef"+i+".mesh");
  ffSaveData3(ux,uy,uz,"EigBeamData"+i+"_3d.txt");
}
```

El procedimiento computacional para calcular la respuesta mecánica de más baja frecuencia de la viga es el siguiente.

- Ejecutar `EigBeam2D.edp` con **FreeFem++**.
- Se produce una salida gráfica como la mostrada en la figura 2.9.

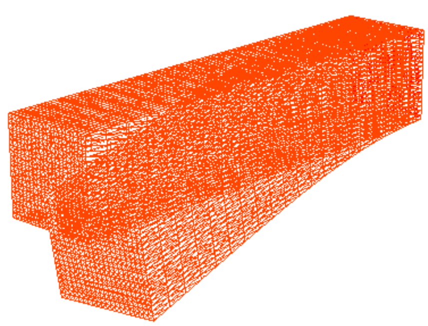

Figura 2.9: Respuesta mecánica aproximada de más baja frecuencia de la viga 3D en voladizo.

- Utilizando **Gmsh** es posible post-procesar los archivos producidos por `EigBeam3D.edp` para obtener las mallas `EigBeam3d.med` y `EigBeam3ddef0.med`.
- Utilizando **FreeCAD** es posible visualizar los archivos `EigBeam3d.med` y `EigBeam3ddef0.med` obteniendo una salida gráfica como la mostrada en la figura 2.10.

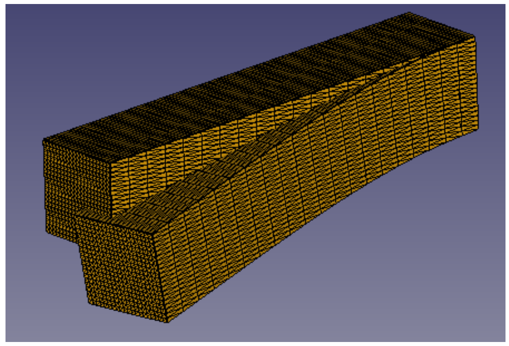

Figura 2.10: Respuesta mecánica aproximada de más baja frecuencia de la viga 3D en voladizo.

2.3. Análisis de Elemento Finito de Elementos Estructurales con Geometría Importada

Consideremos un material $\mathcal{M}$ sólido cuya geometría aproximada $\mathcal{M}_h$ está disponible en algún archivo de malla en cualquiera de los formatos `*.msh, *.stl o *.mesh`. Es posible pre- y post-procesar la geometría utilizando FreeCAD, Gmsh, FreeFem++ y Calculix como se mostrará en los siguientes casos de estudio.

2.3.1. Análisis de Elemento Finito de Elementos Estructurales con Geometría Importada en Gmsh-FreeCAD-FreeFem-Calculix

Deflexión Estática de una Viga Cilíndrica Doblemente Empotrada

Consideremos un material cilíndrico $\mathcal{M}$ cuya geometría aproximada $\mathcal{M}_h$ está disponible en un archivo `Cylinder3D.gmsh` cuyo código se muestra más adelante, y cuya deformación estática es controlada por la equación (2.9) bajo las hipótesis $\partial\mathbf{u} = \mathbf{0} = \partial_t^2\mathbf{u}$. El código de `Cylinder3D.gmsh` es el siguiente.

```
Mesh.Optimize = 1;

////////////////
///PARAMETERS///
////////////////
h = 1./5.; //Mesh quality
L = 20.; //Beam length
D = 1.; //Beam height
Fixed = 1; //Beam fixed label
Free = 2; //Beam free label

////////////////
///ELEMENTARY///
////////////////
//Points
p = newp;
Point(p+0) = { 0.,   0.,   0.};
```

```
Point(p+1) = { D/2., 0.,   0., h};
Point(p+2) = { 0.,   D/2., 0., h};
Point(p+3) = {-D/2., 0.,   0., h};
Point(p+4) = { 0.,  -D/2., 0., h};

Point(p+5) = { 0.,   0.,   L};
Point(p+6) = { D/2., 0.,   L, h};
Point(p+7) = { 0.,   D/2., L, h};
Point(p+8) = {-D/2., 0.,   L, h};
Point(p+9) = { 0.,  -D/2., L, h};

//Lines
l = newl;
Circle(l+0) = {p+1, p+0, p+2};
Circle(l+1) = {p+2, p+0, p+3};
Circle(l+2) = {p+3, p+0, p+4};
Circle(l+3) = {p+4, p+0, p+1};

Circle(l+4) = {p+6, p+5, p+7};
Circle(l+5) = {p+7, p+5, p+8};
Circle(l+6) = {p+8, p+5, p+9};
Circle(l+7) = {p+9, p+5, p+6};

Line(l+10) = {p+1, p+6};
Line(l+11) = {p+2, p+7};
Line(l+12) = {p+3, p+8};
Line(l+13) = {p+4, p+9};

//Line Loops
ll = newll;
Line Loop(ll+0) = {l+0, l+1, l+2, l+3};
Line Loop(ll+1) = {l+4, l+5, l+6, l+7};
Line Loop(ll+2) = {l+0, l+11, -(l+4), -(l+10)};
Line Loop(ll+3) = {l+1, l+12, -(l+5), -(l+11)};
Line Loop(ll+4) = {l+2, l+13, -(l+6), -(l+12)};
Line Loop(ll+5) = {l+3, l+10, -(l+7), -(l+13)};

//Surfaces
s = news;
Plane Surface(s+0) = {ll+0};
Plane Surface(s+1) = {ll+1};
Ruled Surface(s+2) = {ll+2};
Ruled Surface(s+3) = {ll+3};
Ruled Surface(s+4) = {ll+4};
Ruled Surface(s+5) = {ll+5};

//Surface loops
sl = newsl;
```

```
Surface Loop(sl+0) = {s+0, s+1, s+2, s+3, s+4, s+5};

//Volumes
v = newv;
Volume(v+0) = {sl+0};

//////////////
///PHYSICAL///
//////////////
//Surfaces
Physical Surface("Fixed", Fixed) = {s+0, s+1};
Physical Surface("Free", Free) = {s+2, s+3, s+4, s+5};

//Volumes
Physical Volume("Volume", 1) = {v+0};
```

Utilizando el programa **Gmsh** podemos pre-procesar el archivo `Cylinder3D.gmsh` para generar los archivos `Cylinder3D.msh, Cylinder3D.med` y `Cylinder3D.stl`. Utilizando FreeCAD podemos visualizar `Cylinder3D.med` como se muestra en la figura 2.11.

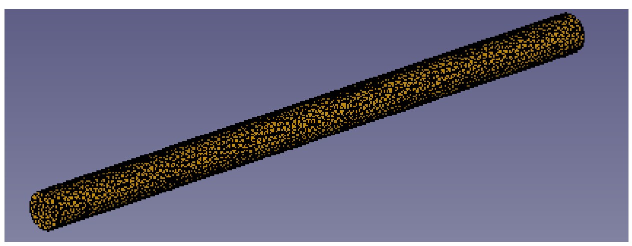

Figura 2.11: Representación de malla 3D aproximada $\mathcal{M}_h$ del material cilíndrico $\mathcal{M}$ en formato *.med.

Utilizando el FreeFem++ podemos importar y procesar la geometría $\mathcal{M}_h$ para calcular la deflexión estática del material cilíndrico utilizando el programa que puede definirse como `Cylinder3D.edp` cuyo código se muestra a continuación.

```
load "gmsh"
load "msh3"
include "ffmatlib.idp"

//Parámetros mecánicos
real Rho = 8000.; //Density
real E = 210.e9; //Young modulus
real Nu = 0.27; //Poisson ratio

real Gravity = -9.81; //Gravity

//Etiquetas de apoyo estructural
int Fixed = 1; //Beam fixed label
```

```
int Free = 2; //Beam free label
mesh3 Th = gmshload3("Cylinder3D.msh");

//Cómputo de funciones de cálculo y de prueba
func Pk = P1;
fespace Uh(Th, [Pk, Pk, Pk]);
Uh [ux, uy, uz];
Uh [vx, vy, vz];
Uh [uxp, uyp, uzp];
Uh [uxpp, uypp, uzpp];

//Macros
real sqrt2 = sqrt(2.);
macro Epsilon(ux, uy, uz) [dx(ux), dy(uy), dz(uz),
(dz(uy)+dy(uz))/sqrt2,
(dz(ux)+dx(uz))/sqrt2,
(dy(ux)+dx(uy))/sqrt2] //
macro Divergence(ux, uy, uz) (dx(ux) + dy(uy) + dz(uz)) //

//Planteamiento del Problema Variacional de Deflexión Estática

real Mu = E/(2.*(1.+Nu));
real Lambda = E*Nu/((1.+Nu)*(1.-2.*Nu));

varf vElasticity ([ux, uy, uz], [vx, vy, vz])
= int3d(Th)(
  Lambda * Divergence(vx, vy, vz) * Divergence(ux, uy, uz)
+ 2. * Mu * (
  Epsilon(vx, vy, vz)' * Epsilon(ux, uy, uz)
)
)
+ int3d(Th)(
  Rho * Gravity * vy
)
+ on(Fixed, ux=0, uy=0, uz=0)
;

// Definición de matriz estructural y vector de cargas

matrix Elasticity = vElasticity(Uh, Uh, solver=sparsesolver);
real[int] ElasticityBoundary = vElasticity(0, Uh);

// Solución de la forma matricial del problema de deflexión

ux[] = Elasticity^-1 * ElasticityBoundary;

// Post-procesamiento de visualización
```

```
real coef=1000;

//Cómputo de malla de deformación
Th = movemesh(Th, [x+coef*ux, y+coef*uy, z+coef*uz]);
[ux, uy, uz] = [ux, uy, uz];

//Visualización
plot([ux, uy, uz], value=true, cmm="u");


//Almacenamiento de Resultados
savemesh(Th,"DefCylinder3D.mesh");
ffSaveVh(Th,Uh,"Cylinder3Dvh.txt");
ffSaveData3(ux,uy,uz,"Cylinder3Dvec.txt");
```

Es posible post-procesar la malla de deformación `DefCylinder3D.mesh` utilizando el programa Gmsh para producir el archivo `DefCylinder3D.med`. Podemos visualizar `Cylinder3D.med` y `DefCylinder3D.med` en FreeCAD obteniendo gráficos como los mostrados en la figura 2.12.

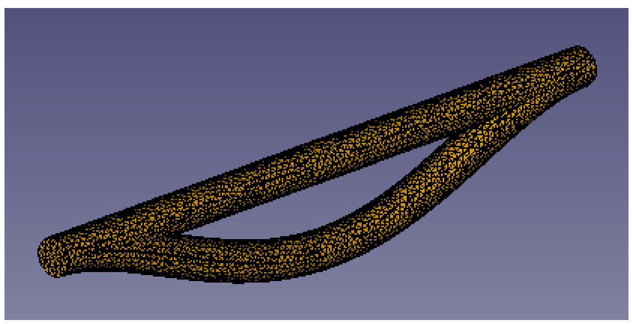

Figura 2.12: Representación 3D aproximada de la deformación $\mathcal{DM}_h$ del material cilíndrico $\mathcal{M}$ en formato *.med.

Deflexión Estática y Respuesta Mecánica del Esqueleto de Acero de un Complejo de Apartamentos de Seis Niveles

Consideremos el material $\mathcal{M}$ determinado por el esqueleto de acero (genérico) de un complejo de apartamentos de seis niveles como el que se muestra en la figura 2.13.

Utilizando FreeCAD es posible crear la malla material aproximada $\mathcal{M}_h$ para el esqueleto de acero $\mathcal{M}$ a partir de los archivos STL `ElementoA.stl` y `ElementoB.stl` cuyas representaciones gráficas pueden visualizarse con FreeCAD como se muestra en la figura 2.14.

Utilizando los módulos **Part** y **Part Design** es posible post-procesar las componentes geométricas elementales `ElementoA.stl` y `ElementoB.stl` para obtener un objeto geométrico como el mostrado en la figura 2.13.

Cómputo de Deflexión Estática con FreeCAD/Calculix

Considerando por simplicidad que la estructura se encuentra empotrada en las bases cuadradas de sus columnas, y considerando solo la carga correspondiente al peso propio de la estructura.

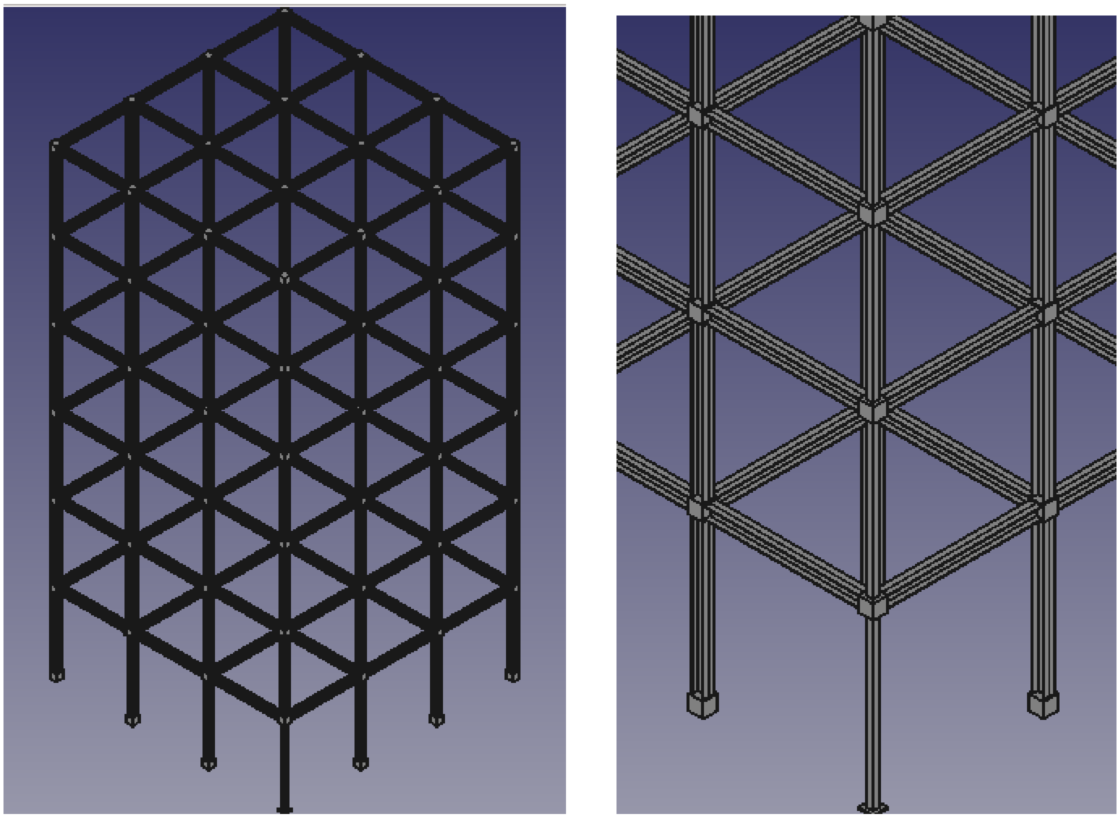

Figura 2.13: Esqueleto de acero de seis niveles $\mathcal{M}$: vista global (izquierda) y vista local (derecha).

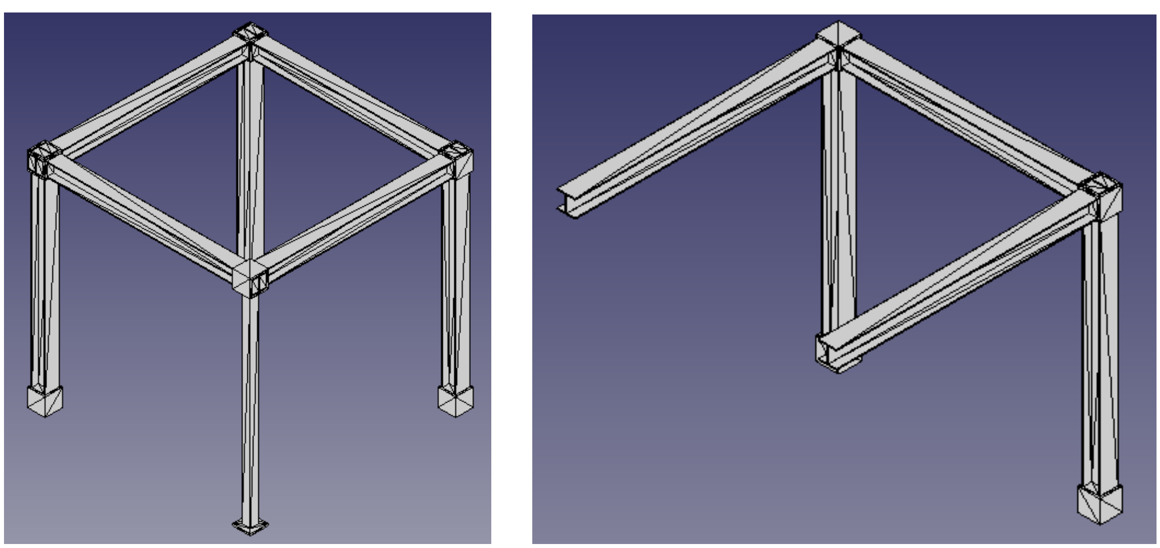

Figura 2.14: Componentes geométricas elementales: `ElementoA.stl` (izquierda) y `ElementoB.stl` (derecha).

Podemos aproximar la deflexión estática de la estructura $\mathcal{M}$ utilizando el siguiente procedmiento.

1. Utilizar el módulo **FEM** de FreeCAD para generar el archivo `EsqueletoDeAcero6.inp` que se bosqueja a continuación.

```
** written by FreeCAD inp file writer for CalculiX,Abaqus meshes
** highest dimension mesh elements only.

** Nodes
*Node, NSET=Nall
1, -4190, 2000, 8.4e-14
2, -4190, 1990, 2.27e-13
3, -4190, 1990, 190
4, -4190, 2000, 190
5, -4380, 2000, 0
6, -4380, 1990, 2.27e-13
7, -4380, 1990, 190
8, -4360, 1990, 20
9, -4360, 1990, 30
10, -4290, 1990, 30
11, -4290, 1990, 160
12, -4360, 1990, 160
13, -4360, 1990, 170
14, -4210, 1990, 170
15, -4210, 1990, 160
16, -4280, 1990, 160
17, -4280, 1990, 30
18, -4210, 1990, 30
19, -4210, 1990, 20
20, -4380, 2000, 190
21, -4190, 2190, 8.4e-14
22, -4190, 2190, 190
23, -4190, 2100, 30
24, -4190, 2170, 30
25, -4190, 2170, 20
26, -4190, 2020, 20
27, -4190, 2020, 30
28, -4190, 2090, 30
29, -4190, 2090, 160
30, -4190, 2020, 160
31, -4190, 2020, 170
32, -4190, 2170, 170
33, -4190, 2170, 160
34, -4190, 2100, 160
35, -4380, 2190, 0
36, -4210, 2170, 0
37, -4360, 2170, 0
38, -4360, 2160, 0
39, -4290, 2160, 2.8e-14
40, -4290, 2030, 2.8e-14
```

```
41, -4360, 2030, 0
42, -4360, 2020, 0
43, -4210, 2020, 0
44, -4210, 2030, 8.5e-14
45, -4280, 2030, 5.7e-14
46, -4280, 2160, 5.7e-14
47, -4210, 2160, 8.5e-14
48, -4360, -10, 20
49, -4360, -10, 30
50, -4290, -10, 30
..........................
103340, 5856, 5723, 66554, 5721, 107062, 106596, 98060, 99999, 19955, 98832
103341, 5723, 5856, 66554, 5472, 107062, 98060, 106596, 100146, 106380, 99211
103342, 5723, 5856, 61427, 5721, 107062, 67837, 61445, 19955, 99999, 61444
103343, 5856, 5723, 61427, 5472, 107062, 61445, 67837, 106380, 100146, 100145

** Define element set Eall
*ELSET, ELSET=Eall
Evolumes



***********************************************************
** Element sets for materials and FEM element type (solid, shell, beam, fluid)
** written by write_element_sets_material_and_femelement_type function
*ELSET,ELSET=SolidMaterialSolid
Evolumes

***********************************************************
** Node sets for fixed constraint
** written by write_node_sets_constraints_fixed function
** FemConstraintFixed
*NSET,NSET=FemConstraintFixed
245,
246,
247,
248,
..............................................................................
54235,
54236,
54237,
54238,
54239,

***********************************************************
** Materials
** written by write_materials function
** Young's modulus unit is MPa = N/mm2
```

```
** Density's unit is t/mm^3
** FreeCAD material name: Steel-Generic
** SolidMaterial
*MATERIAL, NAME=SolidMaterial
*ELASTIC
200000, 0.300
*DENSITY
7.900e-09

***********************************************************
** Sections
** written by write_femelementsets function
*SOLID SECTION, ELSET=SolidMaterialSolid, MATERIAL=SolidMaterial

***********************************************************
** At least one step is needed to run an CalculiX analysis of FreeCAD
** written by write_step_begin function
*STEP
*STATIC


***********************************************************
** Fixed Constraints
** written by write_constraints_fixed function
** FemConstraintFixed
*BOUNDARY
FemConstraintFixed,1
FemConstraintFixed,2
FemConstraintFixed,3


***********************************************************
** Self weight Constraint
** written by write_constraints_selfweight function
** ConstraintSelfWeight
*DLOAD
Eall,GRAV,9810,0.0,0.0,-1.0


***********************************************************
** Outputs --> frd file
** written by write_outputs_types function
*NODE FILE
U
*EL FILE
S, E

***********************************************************
```

```
** written by write_step_end function
*END STEP

***********************************************************
** CalculiX Input file
** written by write_footer function
**   written by    --> FreeCAD 0.18.3.
**   written on    --> Tue Aug 13 16:30:06 2019
**   file name     --> AEF_Stat_Geo_Conectada_Problema_1_Cont.FCStd
**   analysis name --> Analysis
**
**
**
**   Units
**
**   Geometry (mesh data)        --> mm
**   Materials (Young's modulus) --> N/mm2 = MPa
**   Loads (nodal loads)         --> N
**
```

2. Utilizar Calculix/FreeCAD para resolver el problema descrito por `EsqueletoDeAcero6.inp`.

   a) Escribir en terminal:

   ```
   usuario@computer:$ export OMP_NUM_THREADS=4
   usuario@computer:$ cgx -c EsqueletoDeAcero6.inp
   ```

   Obtenemos una salida gráfica como la mostrada en la figura 2.15.

   b) Ejecutar en terminal:

   ```
   usuario@computer:$ ccx EsqueletoDeAcero6

   ************************************************************

   CalculiX Version 2.11, Copyright(C) 1998-2015 Guido Dhondt
   CalculiX comes with ABSOLUTELY NO WARRANTY. This is free
   software, and you are welcome to redistribute it under
   certain conditions, see gpl.htm

   ************************************************************

   You are using an executable made on So 31. Jul 13:26:31 CEST 2016

     The numbers below are estimated upper bounds

     number of:

      nodes:       107062
      elements:       103343
   ```

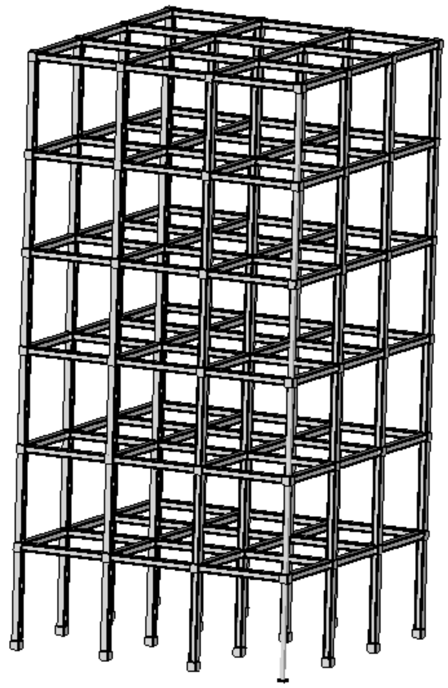

Figura 2.15: Salida gráfica del pre-procesador cgx de Calculix.

```
one-dimensional elements:             0
two-dimensional elements:             0
integration points per element:             4
degrees of freedom per node:             3
layers per element:            1

distributed facial loads:             0
distributed volumetric loads:             1
concentrated loads:            0
single point constraints:          657
multiple point constraints:            1
terms in all multiple point constraints:             1
tie constraints:            0
dependent nodes tied by cyclic constraints:            0
dependent nodes in pre-tension constraints:            0

sets:            5
terms in all sets:       385736

materials:            1
constants per material and temperature:            2
temperature points per material:            1
plastic data points per material:            0
```

```
  orientations:            0
  amplitudes:            2
  data points in all amplitudes:           2
  print requests:           0
  transformations:           0
  property cards:           0


STEP            1

Static analysis was selected

Decascading the MPC's

Determining the structure of the matrix:
number of equations
320529
number of nonzero lower triangular matrix elements
11350740

Using up to 4 cpu(s) for the stress calculation.

Using up to 4 cpu(s) for the symmetric stiffness/mass contributions.

Factoring the system of equations using the symmetric spooles solver
Using up to 4 cpu(s) for spooles.

Using up to 4 cpu(s) for the stress calculation.


Job finished
```

c) Post-procesar el archivo `EsqueletoDeAcero6.frd` generado por **Calculix** utilizando **FreeCAD**. Obtenemos las salidas gráficas mostradas en la figura 2.16.

d) Obtenemos los siguientes valores:

- $\mathbf{u}_{Max} \approx 0{,}12\ mm$
- $\sigma_{C,Max} \approx 1844{,}82\ kPa$

Cómputo de Deflexión Estática con FreeCAD/Calculix

Considerando nuevamente por simplicidad que la estructura se encuentra empotrada en las bases cuadradas de sus columnas, y bajo hipótesis de equilibrio de cargas. Podemos aproximar la deflexión estática de la estructura $\mathcal{M}$ utilizando el siguiente procedmiento.

1. Utilizar el módulo **FEM** de FreeCAD para generar el archivo `EsqueletoDeAceroRM6.inp` que se bosqueja a continuación.

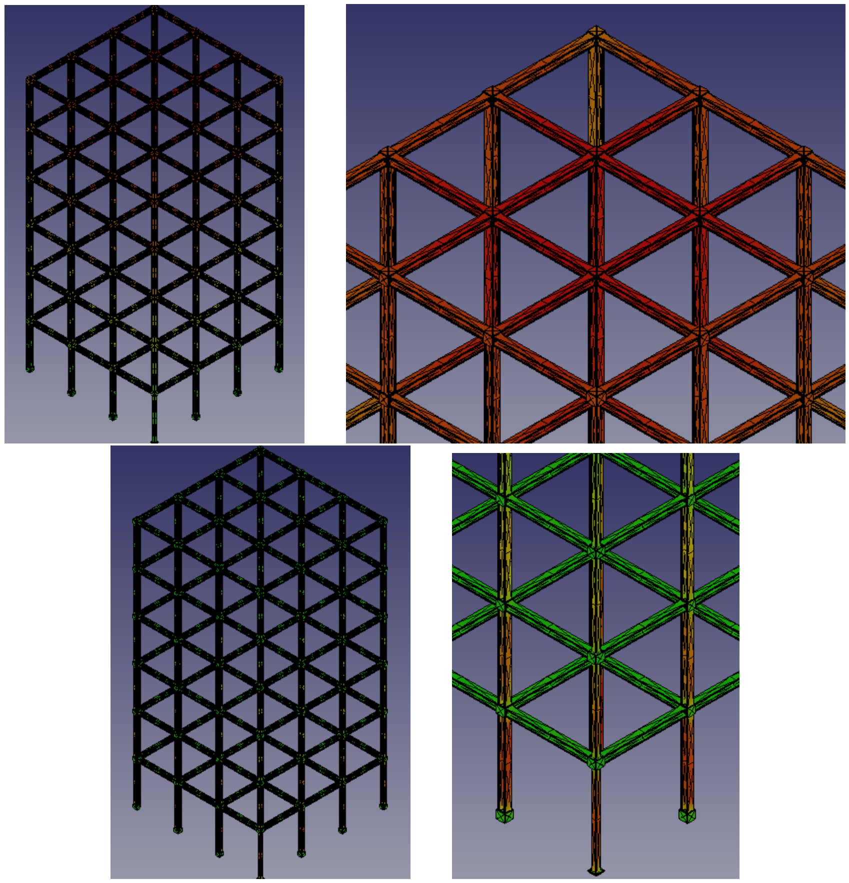

Figura 2.16: Salida gráfica de FreeCAD correspondiente al archivo post-procesado `EsqueletoDeAcero6.frd`: Desplazamiento absoluto (arriba) y Esfuerzo cortante (abajo)

```
** written by FreeCAD inp file writer for CalculiX,Abaqus meshes
** highest dimension mesh elements only.

** Nodes
*Node, NSET=Nall
1, -4190, 2000, 8.4e-14
2, -4190, 1990, 2.27e-13
3, -4190, 1990, 190
4, -4190, 2000, 190
5, -4380, 2000, 0
6, -4380, 1990, 2.27e-13
7, -4380, 1990, 190
8, -4360, 1990, 20
9, -4360, 1990, 30
10, -4290, 1990, 30
11, -4290, 1990, 160
12, -4360, 1990, 160
13, -4360, 1990, 170
14, -4210, 1990, 170
```

```
15, -4210, 1990, 160
16, -4280, 1990, 160
17, -4280, 1990, 30
18, -4210, 1990, 30
19, -4210, 1990, 20
20, -4380, 2000, 190
21, -4190, 2190, 8.4e-14
22, -4190, 2190, 190
23, -4190, 2100, 30
24, -4190, 2170, 30
25, -4190, 2170, 20
26, -4190, 2020, 20
27, -4190, 2020, 30
28, -4190, 2090, 30
29, -4190, 2090, 160
30, -4190, 2020, 160
31, -4190, 2020, 170
32, -4190, 2170, 170
33, -4190, 2170, 160
34, -4190, 2100, 160
35, -4380, 2190, 0
36, -4210, 2170, 0
37, -4360, 2170, 0
38, -4360, 2160, 0
39, -4290, 2160, 2.8e-14
40, -4290, 2030, 2.8e-14
41, -4360, 2030, 0
42, -4360, 2020, 0
43, -4210, 2020, 0
44, -4210, 2030, 8.5e-14
45, -4280, 2030, 5.7e-14
46, -4280, 2160, 5.7e-14
47, -4210, 2160, 8.5e-14
..........................................................
103202, 4816, 66610, 4815, 5172, 96564, 106701, 17828, 105579, 95974, 105685
103203, 66610, 4816, 4815, 54504, 96564, 17828, 106701, 83942, 54520, 54519
103204, 5065, 5610, 5608, 5056, 106865, 87159, 106705, 106704, 103115, 104669
103205, 5610, 5065, 5608, 5609, 106865, 106705, 87159, 19715, 98179, 19714
103206, 372, 243, 233, 244, 106868, 87952, 106126, 96973, 7071, 7072
103207, 243, 372, 233, 369, 106868, 106126, 87952, 103304, 106125, 104665
103208, 4635, 4611, 4808, 54374, 54405, 106717, 54506, 54406, 54392, 98809
103209, 4808, 4611, 4635, 4328, 106717, 54405, 54506, 90952, 101030, 51960
103210, 49641, 4234, 4233, 49664, 106930, 106130, 49647, 106720, 49670, 105992
103211, 4234, 49641, 4233, 4232, 106930, 49647, 106130, 16480, 49648, 16470
103212, 4234, 49641, 49652, 49664, 106930, 102337, 49658, 49670, 106720, 102351
103213, 49641, 4234, 49652, 4232, 106930, 49658, 102337, 49648, 16480, 49659

** Define element set Eall
```

```
*ELSET, ELSET=Eall
Evolumes



***********************************************************
** Element sets for materials and FEM element type (solid, shell, beam, fluid)
** written by write_element_sets_material_and_femelement_type function
*ELSET,ELSET=SolidMaterialSolid
Evolumes

***********************************************************
** Node sets for fixed constraint
** written by write_node_sets_constraints_fixed function
** FemConstraintFixed
*NSET,NSET=FemConstraintFixed
245,
246,
247,
248,
..................................................................................
54232,
54233,
54234,
54235,
54236,
54237,
54238,
54239,

***********************************************************
** Materials
** written by write_materials function
** Young's modulus unit is MPa = N/mm2
** Density's unit is t/mm^3
** FreeCAD material name: Steel-Generic
** SolidMaterial
*MATERIAL, NAME=SolidMaterial
*ELASTIC
200000, 0.300
*DENSITY
7.900e-09

***********************************************************
** Sections
** written by write_femelementsets function
*SOLID SECTION, ELSET=SolidMaterialSolid, MATERIAL=SolidMaterial
```

```
***********************************************************
** At least one step is needed to run an CalculiX analysis of FreeCAD
** written by write_step_begin function
*STEP
*FREQUENCY
1,0.0,1000000.0


***********************************************************
** Fixed Constraints
** written by write_constraints_fixed function
** FemConstraintFixed
*BOUNDARY
FemConstraintFixed,1
FemConstraintFixed,2
FemConstraintFixed,3


***********************************************************
** Outputs --> frd file
** written by write_outputs_types function
*NODE FILE
U
*EL FILE
S, E

***********************************************************
** written by write_step_end function
*END STEP

***********************************************************
** CalculiX Input file
** written by write_footer function
**   written by    --> FreeCAD 0.18.3.
**   written on    --> Tue Aug 13 19:05:20 2019
**   file name     --> AEF_Stat_Geo_Conectada_Problema_1_Cont.FCStd
**   analysis name --> Analysis
**
**
**
**   Units
**
**   Geometry (mesh data)        --> mm
**   Materials (Young's modulus) --> N/mm2 = MPa
**   Loads (nodal loads)         --> N
**
```

2. Utilizar Calculix/FreeCAD para resolver el problema descrito por `EsqueletoDeAceroRM6.inp`.

a) Escribir en terminal:

```
usuario@computer:$ export OMP_NUM_THREADS=4
usuario@computer:$ cgx -c EsqueletoDeAceroRM6.inp
```

Obtenemos una salida gráfica como la mostrada en la figura 2.17.

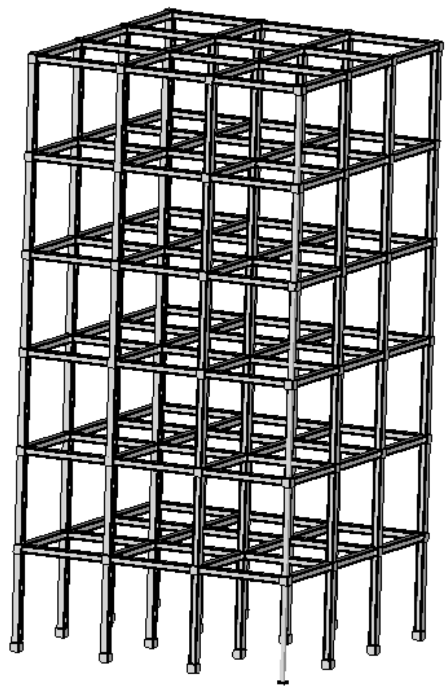

Figura 2.17: Salida gráfica del pre-procesador `cgx` de Calculix.

b) Ejecutar en terminal:

```
usuario@computer:$ ccx EsqueletoDeAceroRM6

************************************************************

CalculiX Version 2.11, Copyright(C) 1998-2015 Guido Dhondt
CalculiX comes with ABSOLUTELY NO WARRANTY. This is free
software, and you are welcome to redistribute it under
certain conditions, see gpl.htm

************************************************************

You are using an executable made on So 31. Jul 13:26:31 CEST 2016

  The numbers below are estimated upper bounds

  number of:
```

```
  nodes:       106930
  elements:       103213
  one-dimensional elements:           0
  two-dimensional elements:           0
  integration points per element:           4
  degrees of freedom per node:           3
  layers per element:           1

  distributed facial loads:           0
  distributed volumetric loads:           0
  concentrated loads:           0
  single point constraints:         657
  multiple point constraints:           1
  terms in all multiple point constraints:           1
  tie constraints:           0
  dependent nodes tied by cyclic constraints:           0
  dependent nodes in pre-tension constraints:           0

  sets:            5
  terms in all sets:       385082

  materials:            1
  constants per material and temperature:           2
  temperature points per material:           1
  plastic data points per material:           0

  orientations:           0
  amplitudes:           1
  data points in all amplitudes:           1
  print requests:           0
  transformations:           0
  property cards:           0


STEP            1

Frequency analysis was selected

Decascading the MPC's

Determining the structure of the matrix:
number of equations
320133
number of nonzero lower triangular matrix elements
11329257

Using up to 4 cpu(s) for the stress calculation.
```

```
 Using up to 4 cpu(s) for the symmetric stiffness/mass contributions.

 Factoring the system of equations using the symmetric spooles solver
 Using up to 4 cpu(s) for spooles.

 Calculating the eigenvalues and the eigenmodes

 Using up to 4 cpu(s) for the stress calculation.


*WARNING: not all frequencies in the requested interval might be found;
increase the number of requested frequencies

 Job finished
```

3. Post-procesar el archivo `EsqueletoDeAceroRM6.frd` generado por **Calculix** utilizando **FreeCAD**. Obtenemos las salidas gráficas mostradas en la figura 2.18.

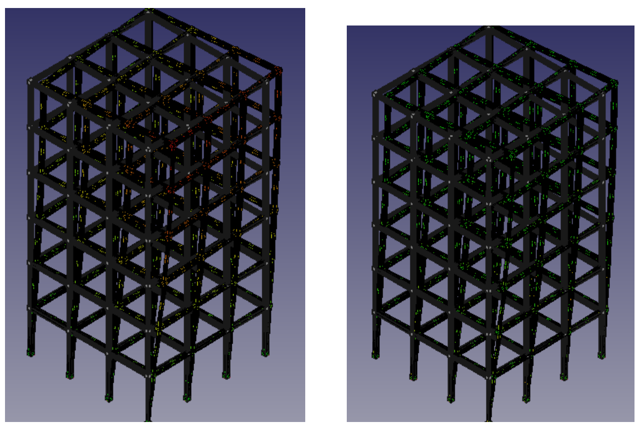

Figura 2.18: Salida gráfica de FreeCAD correspondiente al archivo post-procesado `EsqueletoDeAceroRM6.frd`: Desplazamiento absoluto (izquierda) y Esfuerzo cortante (derecha)

4. Obtenemos los siguientes valores:
   - $\omega_{min} \approx 3{,}24\ Hz$
   - $\mathbf{u}_{Max} \approx 0{,}25\ mm$
   - $\sigma_{C,Max} \approx 22{,}39\ MPa$

Capítulo 3

Aproximación de Deformaciones Estructurales Dinámicas

Objetivos

1. Deducir e Interpretar los modelos dinámicos genéricos para la predicción de la deformación de materiales lineales.

2. Deducir e Interpretar los modelos dinámicos genéricos para la predicción de la deformación de fluidos.

3. Identificar el modelo computacional que mejor describe la deformación dinámica de un elemento estructural dado.

4. Identificar el modelo computacional que mejor describe la deformación dinámica de un fluido dado.

5. Calcular numéricamente de forma eficiente la deformación dinámica aproximada de un elemento estructural o fluido dado utilizando FreeFem++ y GNU Octave.

6. Calcular y/o clasificar numéricamente de forma eficiente la deformación dinámica aproximada de un elemento estructural o fluido dado, aplicando el método de Control Cíclico de Estado Finito (CCEF) con FreeFem++ y GNU Octave.

3.1. Cálculo de Deformación Dinámica de Modelos Matriciales

Consideremos un modelo de deformación dinámica de un material Ω, definido en términos de las matrices estructurales de elementos finitos de Ω y un parámetro $0 \leq \theta \leq 1$ en la forma.

$$\begin{aligned}(\mathbf{M} + \theta\tau\mathbf{A})\mathbf{u}^{n+1} &= \{\mathbf{M} - (1-\theta)\tau\mathbf{A}\}u^n + \tau\left\{\theta\mathbf{f}^{n+1} + (1-\theta)\mathbf{f}^n\right\} \\ \mathbf{M} = (m_{ij}), \quad m_{ij} &= \mathcal{I}(\hat{\mathbf{u}}_i, \hat{\mathbf{u}}_j), \qquad \mathbf{A} = (a_{ij}), \quad a_{ij} = \mathcal{A}(\hat{\mathbf{u}}_i, \hat{\mathbf{u}}_j)\end{aligned} \tag{3.1}$$

Los modelos de la forma (3.1) son denominados modelos de deformación matricial dinámicos en este documento.

3.2. Cálculo de Deformación Dinámica de Modelos de Navier: Ondas Materiales

Para calcular un historial de deformación en un intervalo de tiempo $[0, T]$ con $T > 0$, para un material $\tilde{\mathcal{M}}$ cuyas características mecánicas estructurales son representadas aproximadamente por una ecuación de la forma (2.11), bajo la hipótesis de equilibrio de cargas (se omite peso propio y cargas externas del elemento extructural), basta resolver el sistema de ecuaciones diferenciales ordinarias.

$$\begin{cases} \mathbf{u}_h'(t) = \mathbf{v}_h(t) \\ \mathbf{v}_h'(t) = \frac{1}{\rho_0}\mathbf{N}_{\lambda,\mu}\mathbf{u}_h(t) \\ \mathbf{u}_h(0) = \mathbf{u}_0 \\ \mathbf{v}_h(0) = \mathbf{u}_1 \end{cases} \tag{3.2}$$

Haciendo las sustituciones

$$\mathbf{w}_h(t) = \begin{bmatrix} \mathbf{u}_h(t) \\ \mathbf{v}_h(t) \end{bmatrix}$$

y

$$\mathbf{M}_{\lambda,\nu} = \begin{bmatrix} \mathbf{0} & \mathbf{1} \\ \mathbf{N}_{\lambda,\mu} & \mathbf{0} \end{bmatrix}$$

podemos resolver (3.2) aproximadamente utilizando un esquema de Crank-Nicolson de la forma:

$$(2\mathbf{1} - h_t\mathbf{M}_{\lambda,\mu})\mathbf{w}_h(t + h_t) = (2\mathbf{1} + h_t\mathbf{M}_{\lambda,\mu})\mathbf{w}_h(t) \tag{3.3}$$

para $t \geq 0$ y una longitud de paso temporal $h_t = T/N_t > 0$ para algún entero $N_t \geq 1$.

Otro esquema factible de integración numérica de modelos diferenciales de la forma,

$$\begin{cases} \mathbf{u}_h''(t) = \frac{1}{\rho_0}\mathbf{N}_{\lambda,\mu}\mathbf{u}_h(t) \\ \mathbf{u}_h(0) = \mathbf{u}_0 \\ \mathbf{u}_h'(0) = \mathbf{u}_1 \end{cases} \tag{3.4}$$

están determinados por ecuaciones en diferencias definidas por las siguientes expresiones:

$$\begin{cases} \mathbf{u}_h(t+1) - 2\mathbf{u}_h(t) + \mathbf{u}_h(t-1) = \frac{h_t^2}{2\rho_0}\mathbf{N}_{\lambda,\mu}\mathbf{u}_h(t+1) + \frac{1}{2\rho_0}\mathbf{N}_{\lambda,\mu}\mathbf{u}_h(t-1) \\ \mathbf{u}_h(0) = \mathbf{u}_0 \\ \mathbf{u}_h(1) = \mathbf{u}_0 + h_t\mathbf{u}_1 \end{cases} \tag{3.5}$$

para $t \geq 1$ (entero).

Cálculo de Historial de Deformación Mecánica Material por Reducción de Orden Bidimensional

Consideremos un modelo Navier de la forma (2.9) para una reducción de orden unidimensional de un material $\mathcal{M}$ libre de divergencia, cuya deformación dinámica estará controlada por la siguiente ecuación.

$$\begin{cases} \mu\nabla^2 u_x + \rho_0\delta\partial_t u_x = \rho_0\partial_t^2 u_x \\ \partial_\eta u(x,t) = u^*(x,t), x \in \partial\mathcal{M} \\ u(x,0) = u_0(x) \\ \partial_t u(x,t) = u_1(x) \end{cases} \tag{3.6}$$

donde ∂_η denota la derivada normal a lo largo de la frontera $\partial\mathcal{M}$, y donde $\mathcal{M} = [0, L_x] \times [0, L_y]$, para $L_x, L_y > 0$ y $\delta \in \mathbf{R}$ determinados por la configuración mecánica del material $\mathcal{M}$.

En esta sección aplicaremos la técnica de control cíclico de estado finito (***CCEF***) desarrollada por F. Vides y presentada en [F. Vides, 2019], esta técnica fue desarrollada para resolver problemas de simulación y control de sistemas basados en datos, entre sus aplicaciones se encuentran simulación computacional de modelos estructurales BIM.

3.2.1. Cálculo de ondas materiales en 2D

Consideremos el sistema dinámico determinado por la ecuación de Navier (2.8) y la restricción (3.6) para un medio continuo 2D. Es posible calcular las ondas materiales correspondientes a flujos de estos sistemas dinámicos utilizando como base los programas FreeFEM y Matlab/Octave que se presentan a continuación.

Programa FreeFEM: `Wave2D.edp`

```
include "ffmatlib.idp"

int N1=50;
int N2=50;

real rho=7e3;
real Ly=1;
real Lx=1;

border aa(t=Ly,0)  { x=0; y=t ;label=1;};        //  borde izquierdo
border bb(t=0,Lx) { x=t; y=0 ;label=2;};        //  borde inferior
border cc(t=0,Ly)  { x=Lx; y=t ;label=3;};       //  borde derecho
border dd(t=Lx,0) { x=t; y=Ly; label=4;};     //  borde superior
mesh Th=buildmesh( bb(N1)+cc(N2)+dd(N1)+aa(N2));
plot(Th, cmm="New mesh",wait=true);

//Time-evolution data
real tmax=2*5, dt=0.01,idt=1/(2*dt), idt2=1/dt^2;
// FE space
fespace Vh(Th, P1);
// forma variacional
func g=0.;
Vh ut,vt,u0=x*(x-4)*y*(y-1)*(exp(-(10)^2*((x-.5)^2 + (y-.5)^2))),u1=u0+dt*g;
//+exp(-9*((x-3.5)^2 + (y-.5)^2))),
//-(.2^2-(x-2)^2-(y-.5)^2)*x*(x-4)*y*(y-1)*(exp(-9*((x-.5)^2 + (y-.5)^2))
//+exp(-9*((x-3.5)^2 + (y-.5)^2))),u1=u0+dt*g;
//u0=sin(pi*x)*sin(pi*y)
real c=10;
real d=4/rho;
macro grad(u) [dx(u), dy(u)]//EOM
problem Wave(ut,vt)=
int2d(Th)(idt2*ut*vt)
-int2d(Th)(2*idt2*u1*vt)
+int2d(Th)(idt2*u0*vt)
```

```
+int2d(Th)(d*idt*ut*vt)
-int2d(Th)(d*idt*u0*vt)
+int2d(Th)(.5*c^2*grad(ut)'*grad(vt))
+int2d(Th)(.5*c^2*grad(u0)'*grad(vt))
;

//Time loop
//real t=0;
int iter=0,
nplot=2;
verbosity=0;
int save=0;

savemesh(Th,"wave2d.msh");
ffSaveVh(Th,Vh,"wave2d_vh.txt");

plot(u0,cmm="Wave t="+0,fill=1, value=0, nbiso=65,dim=3,wait=1);

Vh logu;

for (real t=0;t<=tmax;t+=dt)
{
iter++;
Wave;
if(!(iter%nplot))
{
logu=log(abs(ut)^2);
plot(ut,cmm="Wave t="+t,fill=1, value=0, nbiso=65);
cout <<"t="<<t<<"  u min= "<< ut[].min<<"  u max="<< ut[].max <<endl;
ffSaveData(ut,"wave2d_"+save+".txt");
ffSaveData(logu,"logwave2d_"+save+".txt");
save++;
}
u0=u1;
u1=ut;
}
```

Podemos medir la periodicidad/predictividad de los flujos de este sistema dinámico utilizando el programa siguiente MatLab.

Programa MatLab: `CCEFWave2D.edp`

```
function [p,b,t,xh,W,S,C_per,ftol]=CCEFWave2D(N,samplesize,tol,NRep)

addpath('ffmatlib');
[p,b,t,nv,nbe,nt,labels]=ffreadmesh('wave2d.msh');
```

```
[xh]=ffreaddata('wave2d_vh.txt');

n=N;
w=[];
W=w;

for j = 0:(n-1)
    if (mod(j-1,samplesize-1)==0)
    w_name = sprintf('wave2d_%i.txt', j);
    [w]=ffreaddata(w_name);
    W=[W,w];
    h=waitbar(j/n);
    end
end
close(h);

Nw=size(W,2);
W0=W(:,1:(Nw-1));
W1=W(:,2:Nw);
S=W0\W1;
Ec=@(j,n)sparse(((1:n)==j)');
C_per=spdiags(ones(Nw-1,1)*[1 0 0],-1:1,Nw-1,Nw-1);
Kf=W0-W(:,Nw);
Kf=max(abs(Kf));
ftol=min(Kf);
kf=min(find(abs(Kf-ftol)<=tol));
C_per(:,Nw-1)=Ec(kf,Nw-1);
Vps=eig(full(S));
Vpp=eig(full(C_per));
Bx=[-max([abs(Vpp);abs(Vps)]) max([abs(Vpp);abs(Vps)])];
By=Bx;
Nx=60;
Ny=60;
[X,Y]=meshgrid (Bx(1):diff(Bx)/(Nx-1):Bx(2),By(1):diff(By)/(Ny-1):By(2));
disp('------------------------------------------------------------')
  disp(' Computing behavior Pseudospectra:')
  disp('------------------------------------------------------------')
ps=[1 -fliplr(full(S(:,Nw-1)).')];
pc=[1 -fliplr(full(C_per(:,Nw-1)).')];
Zs=abs(polyval(ps,X+i*Y));
Zc=abs(polyval(pc,X+i*Y));
subplot(121);
contour(X,Y,Zs,0:1/64:1);
hold on;
plot(real(Vps),imag(Vps),'r.','markersize',12);
axis equal;
axis tight;
subplot(122);
```

```
contour(X,Y,Zc,0:2/64:2);
hold on;
plot(real(Vpp),imag(Vpp),'r.','markersize',12);
axis equal;
axis tight;
figure;
plot(abs(Kf-ftol));

tic;
  [Yvcty,Rx,Cx,Sx,py]=LMD_Theorem(W0);
  toc;
  What=Cx+Sx;
  Y0=(1/(What(:,1)'*W(:,1)))*What(:,1)*(What(:,1)'*W(:,1));
  Y1=Y0;
  m=size(W,2)-1;
  disp('-------------------------------------------------------------')
  disp(' Computing behavior forcasting:')
  disp('-------------------------------------------------------------')
  tic;

  w_gen=Rx*EvalProjProd(py,Y1);
  Nrep=max([[m NRep]]);
  figure;
  for k=1:NRep,
    h=waitbar(k/NRep);
    Y1=EvalUnitProd(What,C_per,Y1);
    ffpdeplot(p,b,t,'VhSeq',xh,'XYData',log(abs(w_gen(:,k)).^2), 'Mesh','off',...%'ColorRange'
    'ColorMap',hsv,'Boundary','off','Colorbar','off','CBTitle','w','Title',['Wave Propagation:
    axis off;
    drawnow;
    w_gen=[w_gen,Rx*EvalProjProd(py,Y1)];
  end
  close(h);
  toc;
end

function [W,nw,CW,SW,pw]=LMD_Theorem(data_matrix)
  W=data_matrix;
  [N,m]=size(W);
  [uw,sw,vw]=svd(W,0);
  nw=norm(W);
  CW=W/nw;
  pw=uw;
  Qw=[eye(m);zeros(N-m,m)]-pw*pw(1:m,:)';
  [uwc,swc,vwc]=svd(Qw,0);
  SW=uwc*diag(sqrt(abs(1-(diag(sw).^2)/nw^2)))*vw';
  What=CW+SW;
end
```

```
function Y=EvalProjProd(P,Y)
  m=size(P,2);
  Y1=P'*Y;
  Y=P*Y1;
end

function Y=EvalUnitProd(U,C,Y)
  Y=U'*Y;
  Y=C*Y;
  Y=U*Y;
end
```

Podemos también calcular deformación topológica de elementos mecánicos tipo viga utilizando el siguiente programa FreeFEM.

Programa FreeFEM: `WaveBeam2D.edp`

```
include "ffmatlib.idp"

// Parámetros mecánicos
real E = 21.5e4;//32000;
real nu = 0.29;//.17;
// Definición de coeficinetes mecánicos
real mu= E/(2*(1+nu));
real lambda = E*nu/((1+nu)*(1-2*nu));
real d=0;//=4; por defecto

// Coeficiente de fuerza de carga
real f = 1;

// Definición de geometría (ractangular) y mallado
// Sintaxis del comando square:
// mesh Th = square(N_part_x, N_part_y, [Long*x,Alt*y]);

// En esta configuración a la base del rectángulo
// le corresponde la etiqueta 1, los lados siquientes
// se etiquetan contando a partir de la base en el
// sentido anti-horario, por ejemplo, al borde izquierdo
// del rectángulo le corresponde la etiqueta 4.

mesh Th = square(50, 20, [9*x,.2*y]);


// Cómputo de desplazamiento inicial de la viga

fespace Qh(Th,[P1,P1]);
```

```
Qh [uu,vv], [w0,s];

real shift = 0;

//Definición de forma variacional del problema

varf a([uu,vv],[w0,s])=
int2d(Th)(
2*mu*(dx(uu)*dx(w0)+dy(vv)*dy(s)+ ((dx(vv)+dy(uu))*(dx(s)+dy(w0)))/2 )
               + lambda*(dx(uu)+dy(vv))*(dx(w0)+dy(s))
        - shift* (uu*w0 + vv*s)
             )
  + on(4,uu=0,vv=0);

varf b([uu,vv],[w0,s])=
int2d(Th)(uu*w0 + vv*s);

// Cómputo de matrices estructurales

matrix A= a(Qh,Qh,solver=UMFPACK);
matrix B= b(Qh,Qh,solver=CG,eps=1e-20);

// Selección del número de respuestas mecánicas

int nev=1;

// Cómputo de respuestas mecánicas

real[int] ev(nev);
Qh[int] [eV,eW](nev);


int k=EigenValue(A,B,sym=true,sigma=nu,
value=ev,vector=eV,tol=1e-10,maxit=0,ncv=0);

k=min(k,nev);

mesh th0;

real coef=1;

[uu,vv]=[eV[nev-1],eW[nev-1]];
th0 = movemesh(Th, [x+coef*uu, y+coef*vv]);
plot(th0, wait=true);

//coef=1;
```

```
// Parametros temporales
real dt=.01;
real idt2=1/dt^2;

// Definición de funciones de cálculo y de prueba
fespace Vh(Th, P2);
Vh ut, vt,u0,v0,u1,v1,w1,w2;
Vh w;
func g=0.;
u0=uu;//0;//x*(x-1)*y*(y-1)*exp(-9*((x-.5)^2 + (y-.5)^2));
u1=u0+dt*g;
v0=vv;//(4-x)*.1;
v1=v0;

// Macros
real sqrt2=sqrt(2.);
macro epsilon(u1,u2) [dx(u1),dy(u2),(dy(u1)+dx(u2))/sqrt2] //
// The sqrt2 is because we want: epsilon(u1,u2)'* epsilon(v1,v2) = epsilon(u): epsilon(v)
macro div(u,v) ( dx(u)+dy(v) ) //




// Planteamiento y solución de problema de deformación estructural
problem Wave([ut, vt], [w1, w2])
   = int2d(Th)(idt2*(ut*w1+vt*w2))
     -int2d(Th)(2*idt2*(u1*w1+v1*w2))
     +int2d(Th)(idt2*(u0*w1+v0*w2))
      +int2d(Th)(d*(ut*w1+vt*w2)/(2*dt))
      -int2d(Th)(d*(u0*w1+v0*w2)/(2*dt))
      +int2d(Th)(.5*(lambda*div(u0,v0)*div(w1,w2)))
      +int2d(Th)(.5*2.*mu * ( epsilon(u0,v0)' * epsilon(w1,w2)))
      +int2d(Th)(.5*(lambda * div(ut, vt) * div(w1, w2)))
      +int2d(Th)(.5*2.*mu * ( epsilon(ut,vt)' * epsilon(w1,w2)))
   - int2d(Th)(f*w2)
   + on(4,ut=0,vt=0);


//plot([u, v], wait=1, ps="lamevect.eps", coef=coef);

// Cómputo de malla de deformación

real tmax=5;
int iter=0;
int nplot=2;

savemesh(Th,"wavebeam_2d.msh");
ffSaveVh(Th,Vh,"wavebeam_vh_2d.txt");
```

```
int iter2=0;

for (real t=0;t<=tmax;t+=dt)
{
Wave;
if(!(iter%nplot))
{
mesh th1 = movemesh(Th, [x+ut*coef, y+vt*coef]);
plot(th1,wait=0);
savemesh(th1,"wavebeam_2d_def_"+iter2+".msh");
ffSaveData3(w,ut,vt,"wavebeam_data_2d_"+iter2+".txt");
iter2++;
}
u0=u1;
u1=ut;
v0=v1;
v1=vt;
w=sqrt(ut*ut+vt*vt);
iter++;
}
cout<<"eigenvalue= "<<ev<<"\n\n";
```

Es posible medir la periodicidad/predictividad de la muestra discreta de flujos del sistema dinámico correspondiente utilizando el siguiente programa MatLab.

Programa FreeFEM: `CCEFWaveBeam2D.m`

```
function [W,xx,yy]=CCEFWaveBeam2D(N,sample,tol,NRep)

% 2D beam dynamical deformation model
% [W,xx,yy]=CCEFWaveBeam2D(250,2,5e-3,500);
%
% Author: F. Vdies <fredy.vides@unah.edu.hn>
% Created: 2019-08-03
%
% Copyright (C) 2019
%
% This program is free software: you can redistribute it and/or modify it
% under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% This program is distributed in the hope that it will be useful, but
% WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
```

```
%
% You should have received a copy of the GNU General Public License
% along with this program.  If not, see
% <https://www.gnu.org/licenses/>.
%

addpath('ffmatlib');

[p,b,t]=ffreadmesh('wavebeam_2d.msh');
[vh]=ffreaddata('wavebeam_vh_2d.txt');


W=[];

[xmesh,~,ymesh,~]=prepare_mesh(p,t);
x=linspace(0,9,100);
y=linspace(0,.2,20);
[xx,yy]=meshgrid(x,y);
Z=zeros(size(xx));

disp('======================================================');
disp('            Computing History Data:')
disp('======================================================');
tic,
for k=0:N
h=waitbar(k/N);
[w,Ex,Ey]=ffreaddata(['wavebeam_data_2d_',num2str(k),'.txt']);

[~,pdeData1]=convert_pde_data(p,t,vh,Ex');
[~,pdeData2]=convert_pde_data(p,t,vh,Ey');

Ux=fftri2grid(xx,yy,xmesh,ymesh,pdeData1{1});
Uy=fftri2grid(xx,yy,xmesh,ymesh,pdeData2{1});


W=[W,[Ux(:);Uy(:)]];

end
toc;

close(h);

[Nwx,Nw]=size(W);
W0=W(:,1:(Nw-1));
W1=W(:,2:Nw);
S1=W0\W(:,Nw-1);
Ec=@(j,n)sparse(((1:n)==j)');
C_per=spdiags(ones(Nw-1,1)*[1 0 0],-1:1,Nw-1,Nw-1);
```

```
S=C_per;
Kf=W0-W(:,Nw);
Kf=max(abs(Kf));
%Kf=diag(Kf'*Kf);
ftol=min(Kf);
kf=min(find(abs(Kf-ftol)<=tol));
disp(['Index = (',num2str(Nw-1),',',num2str(kf),')']);
C_per(:,Nw-1)=Ec(kf,Nw-1);
S(:,Nw-1)=S1;
Vps=eig(full(S));
Vpp=eig(full(C_per));
Bx=[-max([abs(Vpp);abs(Vps)]) max([abs(Vpp);abs(Vps)])];
By=Bx;
Nx=60;
Ny=60;
[X,Y]=meshgrid (Bx(1):diff(Bx)/(Nx-1):Bx(2),By(1):diff(By)/(Ny-1):By(2));

disp('------------------------------------------------------------')
  disp(' Computing behavior Pseudospectra:')
  disp('------------------------------------------------------------')

ps=[1 -fliplr(full(S(:,Nw-1)).')];
pc=[1 -fliplr(full(C_per(:,Nw-1)).')];

Zs=abs(polyval(ps,X+sqrt(-1)*Y));
Zc=abs(polyval(pc,X+sqrt(-1)*Y));
subplot(121);
contour(X,Y,Zs,0:1/64:1);
hold on;
plot(real(Vps),imag(Vps),'r.','markersize',12);
axis equal;
axis tight;
subplot(122);
contour(X,Y,Zc,0:2/64:2);
hold on;
plot(real(Vpp),imag(Vpp),'r.','markersize',12);
axis equal;
axis tight;
figure;
plot(abs(Kf-ftol));

tic;
  [Yvcty,Rx,Cx,Sx,py]=LMD_Theorem(W0);
  toc;
  What=Cx+Sx;
  Y0=(1/(What(:,1)'*W(:,1)))*What(:,1)*(What(:,1)'*W(:,1));
  Y1=Y0;
  m=size(W,2)-1;
```

```
  disp('------------------------------------------------------------')
  disp(' Computing behavior forcasting:')
  disp('------------------------------------------------------------')
  tic;

  w_gen=Rx*EvalProjProd(py,Y1);
  Nrep=max([[m NRep]]);
  [nx,mx]=size(xx);
  Z=zeros(nx,mx);
  figure;
  hold on;
  for k=1:NRep,
    h=waitbar(k/NRep);
    Y1=EvalUnitProd(What,C_per,Y1);
    Uxx=reshape(w_gen(1:(Nwx/2),k)/4,nx,mx);
    Uyy=reshape(w_gen((1+Nwx/2):Nwx,k)/4,nx,mx);
    surf(xx+Uxx,Z+.05*(k-1),yy+Uyy,sqrt(Uxx.^2+Uyy.^2));
    shading interp;
    axis([-.1,10.1,-.1,.05*(NRep-1),-1.2,1.2]);
    %axis equal;
    %axis off;
    %colormap(ocean);
    %camlight headlight;
    %lighting gouraud;
    view(3);
    pause(.1);
    w_gen=[w_gen,Rx*EvalProjProd(py,Y1)];
  end
  hold off;
  close(h);
  toc;


function [W,nw,CW,SW,pw]=LMD_Theorem(data_matrix)
  W=data_matrix;
  [N,m]=size(W);
  [uw,sw,vw]=svd(W,0);
  nw=norm(W);
  CW=W/nw;
  pw=uw;
  Qw=[eye(m);zeros(N-m,m)]-pw*pw(1:m,:)';
  [uwc,swc,vwc]=svd(Qw,0);
  SW=uwc*diag(sqrt(abs(1-(diag(sw).^2)/nw^2)))*vw';
  What=CW+SW;


function Y=EvalProjProd(P,Y)
  m=size(P,2);
```

```
  Y1=P'*Y;
  Y=P*Y1;


function Y=EvalUnitProd(U,C,Y)
  Y=U'*Y;
  Y=C*Y;
  Y=U*Y;
```

3.2.2. Cálculo de ondas materiales en 3D

Consideremos el sistema dinámico determinado por la ecuación de Navier (2.8) para un medio continuo 3D. Es posible calcular las ondas materiales correspondientes a flujos de estos sistemas dinámicos utilizando como base el programa FreeFEM que se presenta a continuación.

Programa FreeFEM: `WaveBeam3D.edp`

```
load "msh3"
load "tetgen"


// Parameters
int N1=20;
int N2=5;
int N3=40;

real Lx=.35;
real Ly=.45;
real P=.15;

// 2D mesh
border C01(t=0, Lx){x=t; y=0; label=1;}
border C02(t=0, P){ x=Lx; y=t; label=2;}
border C03(t=Lx, Lx/2+P/2){ x=t; y=P; label=3;}
border C04(t=P, Ly-P){ x=Lx/2+P/2; y=t; label=4;}
border C05(t=Lx/2+P/2, Lx){ x=t; y=Ly-P; label=5;}
border C06(t=Ly-P, Ly){ x=Lx; y=t; label=6;}
border C07(t=Lx, 0){ x=t; y=Ly; label=7;}
border C08(t=Ly, Ly-P){ x=0; y=t; label=8;}
border C09(t=0, Lx/2-P/2){ x=t; y=Ly-P; label=9;}
border C10(t=Ly-P, P){ x=Lx/2-P/2; y=t; label=10;}
border C11(t=Lx/2-P/2, 0){ x=t; y=P; label=11;}
border C12(t=P, 0){ x=0; y=t; label=12;}
border C13(t=0,2*pi){x=Lx/2+.25*P*cos(t);y=Ly/2+.25*P*sin(t);label=13;}

mesh Th2 = buildmesh(C01(N1) + C02(N2) + C03(N3) + C04(N1)+C05(N3)+C06(N2)+C07(N1)+C08(N2)+C09

int[int] rup=[0,2],  //  upper face  2d region 0 -> 3d label 2
```

```
   rdown=[0,1],      //  lower face  2d region 0 -> 3d label 1
   rmid=[1,3,  //  vert face.  2d label 1 -> 3d label 1
        2,4,  //  vert face.  2d label 2 -> 3d label 1
        3,5,  //  vert face.  2d label 3 -> 3d label 1
        4,6,  // ...
        5,7,
        6,8,
        7,9,
        8,10,
        9,11,
        10,12,
        11,13,
        12,14,
        13,15],  //  vert face.  2d label 4 -> 3d label 1
   rtet=[0,0];  //  2d region  0-> 3d region  0
real zmin=0,zmax=4.25;

mesh3 Th=buildlayers(Th2,20,
   zbound=[zmin,zmax],
   region=rtet, //  region number
   labelmid=rmid,  // 4 vert.  faces labels number
   labelup = rup,
   labeldown = rdown);

plot(Th2,wait=1);

Th = movemesh(Th, [x, cos(pi/2)*y+sin(pi/2)*z,-sin(pi/2)*y+cos(pi/2)*z+2]);

plot(Th,wait=1);

// Configuración mecánica de la estructura/material

real E = 32e9;
real sigma = 0.17;
real rho = 2400;
real mu = E/(2*(1+sigma));
real lambda = E*sigma/((1+sigma)*(1-2*sigma));
real gravity = -9.81;
real d=10; //amortiguamiento = 4 (por defecto)

// Cómputo de deformación incial

real shift = 0;

fespace Vh(Th, [P1, P1, P1]);
Vh [ux, uy, uz];
Vh [vx, vy, vz];
Vh [uu,vv,ww];
```

```
fespace Vhs(Th,P1);
Vhs uabs;


//Macros
real sqrt2 = sqrt(2.);
macro epsilon(ux, uy, uz) [dx(ux), dy(uy), dz(uz),
(dz(uy)+dy(uz))/sqrt2,
(dz(ux)+dx(uz))/sqrt2,
(dy(ux)+dx(uy))/sqrt2] //
macro div(ux, uy, uz) (dx(ux) + dy(uy) + dz(uz)) //

//Planteamiento del Problema Variacional de Deflexión Estática

varf A ([ux, uy, uz], [vx, vy, vz])
= int3d(Th)(
  lambda * div(vx, vy, vz) * div(ux, uy, uz)
+ 2. * mu * (
  epsilon(vx, vy, vz)' * epsilon(ux, uy, uz)
              - shift* (ux*vx + uy*vy+ uz*vz)
)
)
        - int3d(Th)(gravity*vz)
+ on(1, ux=0, uy=0, uz=0)
;

// Definición de matriz estructural y vector de cargas

matrix K = A(Vh, Vh, solver=sparsesolver);

varf m([ux,uy,uz],[vx,vy,vz])=
int3d(Th)(ux*vx + uy*vy+uz*vz);

matrix M= m(Vh,Vh,solver=CG,eps=1e-20);

int nev=1;

// Cómputo de respuestas mecánicas

real[int] ev(nev);
Vh[int] [eVx,eVy,eVz](nev);


int k=EigenValue(K,M,sym=true,sigma=sigma,
value=ev,vector=eVx,tol=1e-10,maxit=0,ncv=0);

k=min(k,nev);
```

```
// Visualización y almacenamiento de resultados

mesh3 th0;


real coef=5e-2;

  [uu,vv,ww]=[eVx[nev-1],eVy[nev-1],eVz[nev-1]];
  th0 = movemesh(Th, [x+coef*uu, y+coef*vv,z+coef*ww]);
  plot(th0,Th,value=true,fill=true, wait=true);



// Definición de elementos de análisis dinámico

// Parametros temporales
real dt=.01;
real idt2=1/dt^2;


//fespace Vh(Th, [P1,P1,P1]);
Vh [ut,vt,wt],[u0,v0,w0],[u1,v1,w1],[q1,q2,q3];
func g=0.;
[u0,v0,w0]=[uu,vv,ww];
[u1,v1,w1]=[u0,v0,w0]+dt*[g,0,0];



cout << "lambda,mu,gravity ="<<lambda<< " " << mu << " " << gravity << endl;

// Planteamiento de problema de deflexión dinámica

problem Wave([ut,vt,wt], [q1,q2,q3], solver=sparsesolver)
   = int3d(Th)(idt2*rho*(ut*q1+vt*q2+wt*q3))
     -int3d(Th)(2*rho*idt2*(u1*q1+v1*q2+w1*q3))
     +int3d(Th)(idt2*rho*(u0*q1+v0*q2+w0*q3))
      +int3d(Th)(d*rho^2*(ut*q1+vt*q2+wt*q3)/(2*dt))
      -int3d(Th)(d*rho^2*(u0*q1+v0*q2+w0*q3)/(2*dt))
      +int3d(Th)(.5*(lambda*div(u0,v0,w0)*div(q1,q2,q3)))
      +int3d(Th)(.5*2.*mu * ( epsilon(u0,v0,w0)' * epsilon(q1,q2,q3)))
      +int3d(Th)(.5*(lambda * div(ut, vt,wt) * div(q1, q2,q3)))
      +int3d(Th)(.5*2.*mu * ( epsilon(ut,vt,wt)' * epsilon(q1,q2,q3)))
   - int3d(Th)(gravity*q3)
  + on(1,ut=0,vt=0,wt=0)
  ;
```

```
//int[int] ref2=[1,0,2,0];

real tmax=5;
int iter=0;
int nplot=2;


int iter2=0;

for (real t=0;t<=tmax;t+=dt)
{
Wave;
if(!(iter%nplot))
{
mesh3 th1 = movemesh(Th, [x+ut*coef, y+vt*coef,z+wt*coef]);
plot(Th,th1,wait=0);
savemesh(th1,"wavebeam3d_def_"+iter2+".mesh");
uabs=sqrt(ut*ut+vt*vt+wt*wt);
iter2++;
}
[u0,v0,w0]=[u1,v1,w1];
[u1,v1,w1]=[ut,vt,wt];
iter++;
}
```

3.3. Cálculo de Deformación Dinámica en Mecánica de Fluidos

Consideremos un modelo dinámico genérico de Navier-Stokes de la forma.

$$\frac{\partial u}{\partial t} + u \cdot \nabla u - \nu \Delta u + \nabla p = 0, \nabla \cdot u = 0 \tag{3.7}$$

Podemos integrar estas ecuaciones en el tiempo utilizando el esquema `convect` de **FreeFEM** utilizándolo para discretizar el operador $\frac{\partial u}{\partial t} + u \cdot \nabla u$, produciendo un esquema de aproximación de la forma

$$\begin{array}{rl} \frac{1}{\tau}(u^{n+1} - u^n \circ X^n) - \nu \Delta u^{n+1} + \nabla p^{n+1} & = 0, \\ \nabla \cdot u^{n+1} & = 0 \end{array} \tag{3.8}$$

donde el término $u^n \circ X^n(x) \approx u^n(x - u^n(x)\tau)$ será aproximado con el esquema `convect`.

Cálculo de historial de evolución de un Fluido de Navier-Stokes con Obstáculo Cilíndrico

En este ejemplo resolveremos el problema modelo de flujo de Navier-Stokes con obstáculo cilíndrico utilizando el algoritmo de Uzawa precondicionado con el método de Cahouet-Chabart. Para esto utilizaremos el programa `NS2D.edp` cuyo código **FreeFEM** se describle a continuación.

```
// Parameters
verbosity = 0;
real D = 0.1;
```

```
real H = 0.41;
real cx0 = 0.2;
real cy0 = 0.2; //center of cylinder
real xa = 0.15;
real ya = 0.2;
real xe = 0.25;
real ye = 0.2;
int nn = 15;

//TODO
real Um = 1.5; //max velocity (Rey 100)
real nu = 1e-3;

func U1 = 4.*Um*y*(H-y)/(H*H); //Boundary condition
func U2 = 0.;
real T=2;
real dt = D/nn/Um; //CFL = 1
real epspq = 1e-10;
real eps = 1e-6;

// Variables
func Ub = Um*2./3.;
real alpha = 1/dt;
real Rey = Ub*D/nu;
real t = 0.;

// Mesh
border fr1(t=0, 2.2){x=t; y=0; label=1;}
border fr2(t=0, H){x=2.2; y=t; label=2;}
border fr3(t=2.2, 0){x=t; y=H; label=1;}
border fr4(t=H, 0){x=0; y=t; label=1;}
border fr5(t=2*pi, 0){x=cx0+D*sin(t)/2; y=cy0+D*cos(t)/2; label=3;}
mesh Th = buildmesh(fr1(5*nn) + fr2(nn) + fr3(5*nn) + fr4(nn) + fr5(-nn*3));

// Fespace
fespace Mh(Th, [P1]);
Mh p;

fespace Xh(Th, [P2]);
Xh u1, u2;

fespace Wh(Th, [P1dc]);
Wh w; //vorticity

// Macro
macro grad(u) [dx(u), dy(u)] //
macro div(u1, u2) (dx(u1) + dy(u2)) //
```

```
// Problem
varf von1 ([u1, u2, p], [v1, v2, q])
    = on(3, u1=0, u2=0)
    + on(1, u1=U1, u2=U2)
    ;

//remark : the value 100 in next varf is manualy fitted, because free outlet.
varf vA (p, q) =
    int2d(Th)(
          grad(p)' * grad(q)
    )
    + int1d(Th, 2)(
          100*p*q
    )
    ;

varf vM (p, q)
    = int2d(Th, qft=qf2pT)(
          p*q
    )
    + on(2, p=0)
    ;

varf vu ([u1], [v1])
    = int2d(Th)(
          alpha*(u1*v1)
        + nu*(grad(u1)' * grad(v1))
    )
    + on(1, 3, u1=0)
    ;

varf vu1 ([p], [v1]) = int2d(Th)(p*dx(v1));
varf vu2 ([p], [v1]) = int2d(Th)(p*dy(v1));

varf vonu1 ([u1], [v1]) = on(1, u1=U1) + on(3, u1=0);
varf vonu2 ([u1], [v1]) = on(1, u1=U2) + on(3, u1=0);

matrix pAM = vM(Mh, Mh, solver=UMFPACK);
matrix pAA = vA(Mh, Mh, solver=UMFPACK);
matrix AU = vu(Xh, Xh, solver=UMFPACK);
matrix B1 = vu1(Mh, Xh);
matrix B2 = vu2(Mh, Xh);

real[int] brhs1 = vonu1(0, Xh);
real[int] brhs2 = vonu2(0, Xh);

varf vrhs1(uu, vv) = int2d(Th)(convect([u1, u2], -dt, u1)*vv*alpha) + vonu1;
varf vrhs2(v2, v1) = int2d(Th)(convect([u1, u2], -dt, u2)*v1*alpha) + vonu2;
```

```
// Uzawa function
func real[int] JUzawa (real[int] & pp){
    real[int] b1 = brhs1; b1 += B1*pp;
    real[int] b2 = brhs2; b2 += B2*pp;
    u1[] = AU^-1 * b1;
    u2[] = AU^-1 * b2;
    pp = B1'*u1[];
    pp += B2'*u2[];
    pp = -pp;
    return pp;
}

// Preconditioner function
func real[int] Precon (real[int] & p){
    real[int] pa = pAA^-1*p;
    real[int] pm = pAM^-1*p;
    real[int] pp = alpha*pa + nu*pm;
    return pp;
}

// Initialization
p = 0;

// Time loop
int ndt = T/dt;
for(int i = 0; i < ndt; ++i){
    // Update
    brhs1 = vrhs1(0, Xh);
    brhs2 = vrhs2(0, Xh);

    // Solve
    int res = LinearCG(JUzawa, p[], precon=Precon, nbiter=100, verbosity=10, veps=eps);
    assert(res==1);
    eps = -abs(eps);

    // Vorticity
    w = -dy(u1) + dx(u2);
    plot(w, fill=true, wait=0, nbiso=40);

    // Update
    dt = min(dt, T-t);
    t += dt;
    if(dt < 1e-10*T) break;
}

// Plot
plot(w, fill=true, nbiso=40);
```

```
// Display
cout << "u1 max = " << u1[].linfty
    << ", u2 max = " << u2[].linfty
    << ", p max = " << p[].max << endl;
```

- Ejecutamos `NS2D.edp` con **FreeFEM**.
- Esto produce salidas gráficas como las mostradas en la figura

Figura 3.1: Dos perfiles de flujo de Navier-Stokes alrededor de un obstáculo cilíndrico.

Cálculo de historial de evolución de una Calle de Vórtices de von Kármán con Obstáculo Cilíndrico

Consiremos el modelo diferencial determinado por la ecuación:

En este ejemplo resolveremos el problema modelo de flujo de una calle de vórtices de von Kármán con Obstáculo Cilíndrico. Para esto utilizaremos el programa `CFSA-Karman-Vortex-Street.edp` cuyo código **FreeFEM** se describle a continuación. Además aplicaremos el método CCEF desarrollado por F. Vides en [Vides, F., 2019], para calcular la predicción de comportamiento del flujo, para esto utilizaremos el programa Octave llamado `CCEFCKarman2D.m`. El procedimiento computacional a seguir es el siguiente.

- Escribir un programa **FreeFEM** que calcule estimaciones de elemento finito de las calles de vórtices de von Kármán, el cual llamaremos `CFSA-Karman-Vortex-Street.edp` y cuyo código tendrá la forma.

```
/* von Karman vortex street simulation
 * Developed as part of CFSA method
 * Developed by F. Vides
 * Presented in:
 * "On Cyclic Finite-State Approximation of Data-Driven Systems." IEEE Xplore. 2019.
 *
 * Author: Fredy Vides <fredy.vides@unah.edu.hn>
 * Created: 2019-02-06
 * Based on karman-vortex-street.edp by: Chloros2 <chloros2@gmx.de>
```

```
 *
 * Copyright (C) 2018 Chloros2 <chloros2@gmx.de>
 *
 * Copyright (C) 2019
 *
 * This program is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see
 * <https://www.gnu.org/licenses/>.
 *
 */

include "ffmatlib.idp"

real scale = 0.5;
real width = 2*4.0*scale;
real height = 2*0.8*scale;
real R = 0.04*scale;

real xcg=(.2+.45)/2-.02;
real ycg=.25*2*.8;

int C1=1,C2=2,C3=3,C4=4,C5=5,C6=6,C7=7,S=8,S1=9;
border floor(t=0,width){ x=t; y=0; label=S1;};
border ceiling(t=width,0){ x=t; y=height; label=C1;};
border right(t=0,height){ x=width; y=t; label=C3;};
border left(t=height,0){ x=0; y=t; label=C2;};
border cir(t=2*pi,0){ x=0.1*width+R*cos(t); y=0.5*height+R*sin(t); label=C4;};
/*border Splus(t=.2, .45){x=t-.02; y=.25*(2*.8-.3*(4*(.2-t))^2+0.17735*sqrt(4*(t-.2)) - 0.
   - 0.212836*(4*(t-.2))^2 + 0.17363*(4*(t-.2))^3 - 0.06254*(4*(t-.2))^4); label=S;}
  border Sminus(t=.45, .2){x=t-.02; y=.25*(2*.8-.3*(4*(.2-t))^2-(0.17735*sqrt(4*(t-.2)) -
     -0.212836*(4*(t-.2))^2 + 0.17363*(4*(t-.2))^3 - 0.06254*(4*(t-.2))^4)); label=S;}*/

border Splus(t=.2, .45){x=xcg+cos(pi/4)*(t-.02-xcg)+sin(pi/4)*(-ycg+.25*(2*.8-0*.3*(4*(.2-
   - 0.212836*(4*(t-.2))^2 + 0.17363*(4*(t-.2))^3 - 0.06254*(4*(t-.2))^4)); y=ycg-sin(pi/4
   - 0.212836*(4*(t-.2))^2 + 0.17363*(4*(t-.2))^3 - 0.06254*(4*(t-.2))^4)); label=S;}
border Sminus(t=.45, .2){x=xcg+cos(pi/4)*(t-.02-xcg)+sin(pi/4)*(-ycg+.25*(2*.8-0*.3*(4*(.2
   -0.212836*(4*(t-.2))^2 + 0.17363*(4*(t-.2))^3 - 0.06254*(4*(t-.2))^4))); y=ycg-sin(pi/4
   -0.212836*(4*(t-.2))^2 + 0.17363*(4*(t-.2))^3 - 0.06254*(4*(t-.2))^4))); label=S;}
```

```
int n=7;

mesh Th=buildmesh(floor(5*(width/height)*n)+right(5*n)+
ceiling(5*(width/height)*n)+left(5*n)+Splus(10*n)+Sminus(10*n));

fespace Xh(Th, P2);
fespace Mh(Th, P1);
Xh u2, v2, u1, v1, up1, up2;
Mh p, q;
Xh v, u, w;
Xh uold = 0;
Xh vcty;

plot(Th);

int nRuns=2*700;
bool reuseMatrix=false;
real dt=7.0*scale*scale;


real vinf = 0.0018/scale;  // m/s
real rho = 1000.0;         // density kg/m^3
real nu = 0.9e-6;          // viscocity m^2/s
real cp = 4.2*1000.0;      // heat capacity J/kg*K
real lambda = 7*0.6;       // heat conductance W/mK
real mu = nu*rho;
real a = lambda/(rho*cp);

cout << "Reynolds Number: " << 2*R*vinf/nu << endl;
cout << "Prandtl Number: " << nu/a << endl;

problem  NS([u1,u2,p],[v1,v2,q],solver=UMFPACK,init=reuseMatrix) =
    int2d(Th)( u1*v1 + u2*v2                        //Velocity field Discretization
             + dt*nu*(dx(u1)*dx(v1) + dy(u1)*dy(v1) //Viscocity term Discretization
             +     dx(u2)*dx(v2) + dy(u2)*dy(v2))
             + p*q*1.e-6                            //Stabilization
             - dt*(p*(dx(v1) + dy(v2))/rho          //Thermal pressure
             + q*(dx(u1) + dy(u2))) )               //Mass conservation
    - int2d(Th)(convect([up1,up2],-dt,up1)*v1
             + convect([up1,up2],-dt,up2)*v2)
  + on(C2,u1=vinf,u2=0)
  + on(C1,S1,u2=0)
  + on(S,u1=0,u2=0)
;

real Tsurf = 1;
problem convectdiffusion(u,v) =
```

```
    int2d(Th)(u*v + a*dt*(dx(u)*dx(v) + dy(u)*dy(v)))
    - int2d(Th)(convect([up1,up2],-dt,uold)*v)
  + on(C2,u=0)
  + on(S,u=Tsurf);

savemesh(Th,"karman_vortex.msh");
ffSaveVh(Th,Mh,"karman_vortex_mh.txt");
ffSaveVh(Th,Xh,"karman_vortex_xh.txt");

int k,kvcty=1;
real t=0.0;

int samplesize=6;

for (int i=0 ; i < nRuns ; i++) {
   up1 = u1;
   up2 = u2;
   NS;
   reuseMatrix = true;
   convectdiffusion;
   uold = u;   /* temperature plot */
   t+=dt;
   /* vorticity */
   vcty = dx(u2)-dy(u1);
   if((i-1)%(samplesize-1)==0) {
   plot(vcty, wait = false, value=0, fill=true, ShowAxes=false,
   cmm="RunNumber: "+i+"/"+nRuns+"  Time: "+t+"sec"+"  vInf: "+vinf+"m/s");
   ffSaveData(vcty,"karman_vortex_vorticity_"+kvcty+".txt");
   kvcty++;
   }
}
```

- Ejecutamos `CFSA-Karman-Vortex-Street.edp` para generar patrones de estimación de calles de vórtices de von Kármán.
- Esto produce salidas gráficas como las mostradas en la figura 3.2. Además genera archivos de malla `*.mesh` y de datos `*.txt` que permiten almacenar los patrones de calle de vórtices de von Kármán.
- Podemos visualizar la malla de la región de cómputo en Octave utilizando la siguiente secuencia de comandos.

```
>> figure
>> ffpdeplot(p,b,t,'Mesh','on','Boundary','on','Title','Mallado');
>> axis equal
>> axis tight
```

- Esto produce una salida gráfica como la mostrada en la figura 3.3.

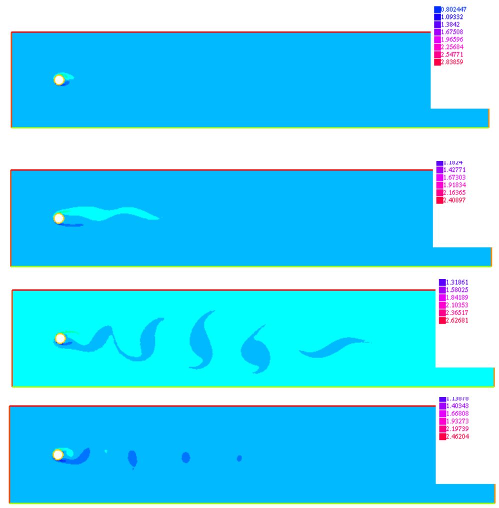

Figura 3.2: Cuatro perfiles de flujo de calle de vórtices de von Kármán alrededor de un obstáculo cilíndrico.

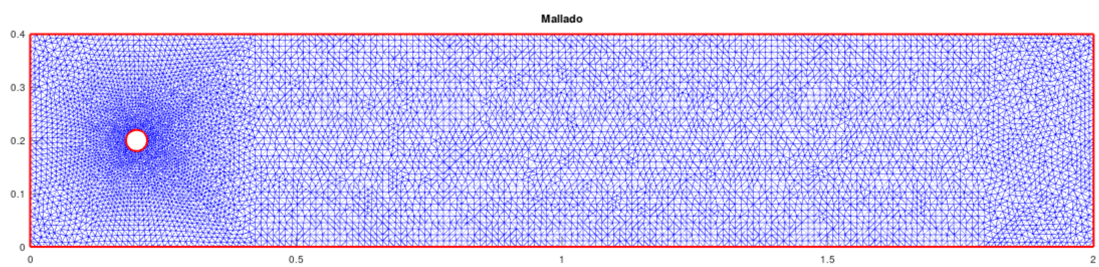

Figura 3.3: Malla de la región de cómputo.

- Escribir un programa **Octave** que calcule predicciones y controles cíclicos de estado finito de las calles de vórtices de von Kármán, el cual llamaremos `CCEFKarmanVortexStreet.m` y cuyo código tendrá la forma.

```
function [p,b,t,xh,W,S,C_per,ftol]=CCEFKarmanVortexStreet(N,samplesize,tol,NRep)
% * von Karman vortex street simulation
% * Developed as part of CFSA method
% * Developed by F. Vides
% * Presented in:
% * "On Cyclic Finite-State Approximation of Data-Driven Systems." IEEE Xplore. 2019.
% *
% * Author: Fredy Vides <fredy.vides@unah.edu.hn>
% * Created: 2019-02-06
%
% * Copyright (C) 2019
% *
% Example:
% [p,b,t,xh,W,S,C_per,ftol]=CCEFKarmanVortexStreet(280,2,1e-10,350);
```

```
% *
% * This program is free software: you can redistribute it and/or modify it
% * under the terms of the GNU General Public License as published by
% * the Free Software Foundation, either version 3 of the License, or
% * (at your option) any later version.
% *
% * This program is distributed in the hope that it will be useful, but
% * WITHOUT ANY WARRANTY; without even the implied warranty of
% * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% * GNU General Public License for more details.
% *
% * You should have received a copy of the GNU General Public License
% * along with this program.  If not, see
% * <https://www.gnu.org/licenses/>.
% *
% */

addpath('ffmatlib');
[p,b,t,nv,nbe,nt,labels]=ffreadmesh('karman_vortex.msh');

[xh]=ffreaddata('karman_vortex_xh.txt');
[mh]=ffreaddata('karman_vortex_mh.txt');

n=N;
w=[];
W=w;

for j = 1:n
    if (mod(j-1,samplesize-1)==0)
    w_name = sprintf('karman_vortex_vorticity_%i.txt', j);
    [w]=ffreaddata(w_name);
    W=[W,w];
    h=waitbar(j/n);
    end
end
close(h);

Nw=size(W,2);
W0=W(:,1:(Nw-1));
W1=W(:,2:Nw);
S=W0\W1;
Ec=@(j,n)sparse(((1:n)==j)');
C_per=spdiags(ones(Nw-1,1)*[1 0 0],-1:1,Nw-1,Nw-1);
Kf=W0-W(:,Nw);
%Kf=max(abs(Kf));
Kf=diag(Kf'*Kf);
ftol=min(Kf);
kf=min(find(abs(Kf-ftol)<=tol));
```

```
C_per(:,Nw-1)=Ec(kf,Nw-1);
Vps=eig(full(S));
Vpp=eig(full(C_per));
Bx=[-max([abs(Vpp);abs(Vps)]) max([abs(Vpp);abs(Vps)])];
By=Bx;
Nx=60;
Ny=60;
[X,Y]=meshgrid (Bx(1):diff(Bx)/(Nx-1):Bx(2),By(1):diff(By)/(Ny-1):By(2));
Zs=zeros(Ny,Nx);
Zc=Zs;
E=eye(Nw-1);
pspectra=@(Z)min(svd(Z));
tic;
disp('-------------------------------------------------------------')
  disp(' Computing behavior Pseudospectra:')
  disp('-------------------------------------------------------------')
%  tic;
%for k=1:Ny,
%   for l=1:Nx,
%      Zs(k,l)=pspectra(S-(X(k,l)+i*Y(k,l))*E);
%      Zc(k,l)=pspectra(C_per-(X(k,l)+i*Y(k,l))*E);
%   end
%   h=waitbar(k/Ny);
%end
ps=[1 -fliplr(full(S(:,Nw-1)).')];
pc=[1 -fliplr(full(C_per(:,Nw-1)).')];
Zs=abs(polyval(ps,X+i*Y));
Zc=abs(polyval(pc,X+i*Y));
toc;
subplot(121);
contour(X,Y,Zs,64);
hold on;
plot(real(Vps),imag(Vps),'r.','markersize',12);
axis equal;
axis tight;
subplot(122);
contour(X,Y,Zc,64);
hold on;
plot(real(Vpp),imag(Vpp),'r.','markersize',12);
axis equal;
axis tight;
figure;
plot(abs(Kf-ftol));

tic;
  [Yvcty,Rx,Cx,Sx,py]=LMD_Theorem(W0);
  toc;
  What=Cx+Sx;
```

```
  Y0=(1/(What(:,1)'*W(:,1)))*What(:,1)*(What(:,1)'*W(:,1));
  Y1=Y0;
  m=size(W,2)-1;
  disp('-------------------------------------------------------------')
  disp(' Computing behavior forcasting:')
  disp('-------------------------------------------------------------')
  tic;

  w_gen=Rx*EvalProjProd(py,Y1);
  Nrep=max([[m NRep]]);
  figure;
  for k=1:NRep,
    h=waitbar(k/NRep);
    Y1=EvalUnitProd(What,C_per,Y1);
    ffpdeplot(p,b,t,'VhSeq',xh,'XYData',w_gen(:,k), 'Mesh','off','ColorRange',[-0.1,0.1],.
    'ColorMap',winter,'Boundary','off','CBTitle','w','Title',['Vorticity: \Omega_{' num2st
    drawnow;
    pause(.1);
    w_gen=[w_gen,Rx*EvalProjProd(py,Y1)];
  end
  close(h);
  toc;
end

function [W,nw,CW,SW,pw]=LMD_Theorem(data_matrix)
  W=data_matrix;
  [N,m]=size(W);
  [uw,sw,vw]=svd(W,0);
  nw=norm(W);
  CW=W/nw;
  pw=uw;
  Qw=[eye(m);zeros(N-m,m)]-pw*pw(1:m,:)';
  [uwc,swc,vwc]=svd(Qw,0);
  SW=uwc*diag(sqrt(abs(1-(diag(sw).^2)/nw^2)))*vw';
  What=CW+SW;
end

function Y=EvalProjProd(P,Y)
  m=size(P,2);
  Y1=P'*Y;
  Y=P*Y1;
end

function Y=EvalUnitProd(U,C,Y)
  Y=U'*Y;
  Y=C*Y;
  Y=U*Y;
end
```

- Ejecutamos el programa `CCEFKarmanVortexStreet` utilizando el comando.

```
>> [p,b,t,xh,W,S,C_per,ftol]=CCEFKarmanVortexStreet(78,2,1e-11,150);
--------------------------------------------------------------
 Computing behavior Pseudospectra:
--------------------------------------------------------------
Elapsed time is 4.52009 seconds.
Elapsed time is 0.57788 seconds.
--------------------------------------------------------------
 Computing behavior forcasting:
--------------------------------------------------------------
Elapsed time is 225.528 seconds.
```

- Esto produce salidas gráficas como las mostradas en la figura 3.4. Además los diagramas espectrales de predicción se muestran en la figura 3.5.

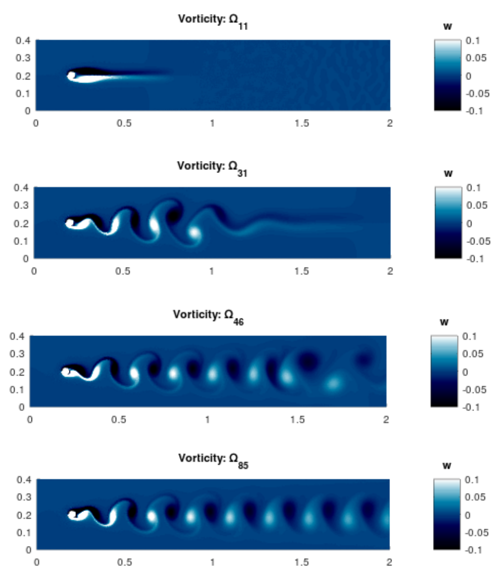

Figura 3.4: Cuatro perfiles de flujo de calle de vórtices de von Kármán alrededor de un obstáculo cilíndrico.

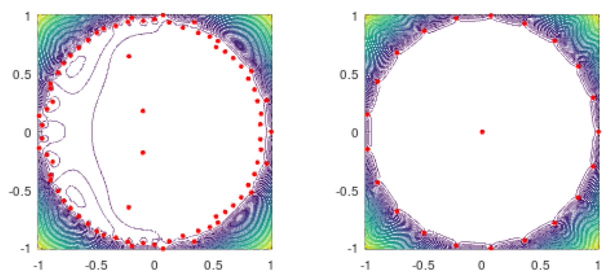

Figura 3.5: Diagramas espectrales de predicción del comportamiento del modelo: Predicción DMD (izquierda) y predicción CCEF (derecha).

Bibliografía

[1] Oliver Olivella, X., de Saracíbar Bosch, C. A. (2002), Mecánica de medios continuos para ingenieros. EDICIONS UPC.

[2] Hecht, F. FreeFEM Documentation. Release 4.2.1.

[3] Zill, Dennis G, Cullen, Michael R. Ecuaciones Diferenciales. 3 ed. México; McGraw - Hill Interamericana Editores, 2010.

[4] Penney, Edwards (2001). Ecuaciones Diferenciales. 2 ed. MEXICO, PRETICE HALL : 2001007

[5] Kattan, P (2008). MATLAB Guide to Finite Elements. Springer-Verlag Berlin Heidelberg 2008.

[6] F. Vides (2018). Introducción al Cómputo Científico e Industrial con GNU Octave. (Lecturas de Clase UNAH)

[7] F. Vides (2019). On Cyclic Finite-State Approximation of Data-Driven Systems. IEEEXplore 2019. Preprint disponible de forma gratuita en https://arxiv.org/pdf/1907.06568.pdf.