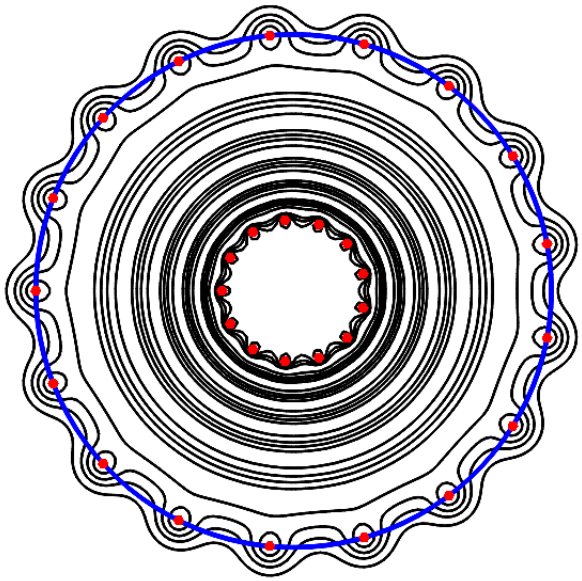
Métodos Numéricos y Modelación Computacional



FREDY VIDES
Centro de Innovación en Cómputo Científico e Industrial
Escuela de Matemática y Ciencias de la Computación
Universidad Nacional Autónoma de Honduras

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

22 de febrero de 2021

Presentación

En este documento presentaremos algunas técnicas de modelación matemática y simulación numérica utilizando las herramientas computacionales GNU Octave, FreeFEM y Calculix. Muchas de las técnicas que presentamos han sido desarrolladas en el Centro de Innovación en Cómputo Científico del Universidad Nacional Autónoma de Honduras (CICC-UNAH).

El material presentado en este documento está orientado a profesionales y estudiantes avanzados de matemática aplicada, 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 de los métodos computacionales considerados en este material, incluyen el cálculo de deformaciones en mecánica estructural, deformaciones en mecánica de fluidos, y simulación de procesos en biología y microbiología.

Índice general

Parte I

GNU Octave

Capítulo 1

Cómputo Matricial Básico con GNU Octave

1.1. Cómputo Matricial Básico

Para desarrollar los proyectos computacionales que involucran programas escritos en el lenguaje de programación de GNU Octave, es recomendable descargar y compilar el código fuente del programa GNU Octave en su versión 6.1.0, enlaces al repositorio del código fuente están disponibles en la dirección `https://www.gnu.org/software/octave/download.html`.

1.1.1. Operaciones y funciones matriciales

Consideremos las siguientes matrices:

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \tag{1.1.1}$$

$$B = \begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{1.1.2}$$

$$C = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{1.1.3}$$

$$D = \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{1.1.4}$$

$$v = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \tag{1.1.5}$$

Los comandos necesarios para ingresar las matrices anteriores en Octave/MATLAB son los siguientes:

```
>> A=[2,-1,0;-1,2,-1;0,-1,2]
A =

   2  -1   0
  -1   2  -1
   0  -1   2

>> B=[0,-1,1;1,0,0;0,1,0];
>> C=[0,0,1;1,0,0;0,1,0];
>> D=[0,-1;1,0;0,1];
>> v=[0;1;0];
```

Operaciones con matrices

Mostraremos a continuación los procedimientos computacionales correspondientes a algunas operaciones algebraicas fundamentales.

- $R_1 = A + B$:

```
>> R1=A+B
R1 =

   2  -2   1
   0   2  -1
   0   0   2
```

- $R_2 = 7A$:

```
>> R2=7*A
R2 =

   14   -7    0
   -7   14   -7
    0   -7   14
```

- $R_3 = B^\top$

```
>> R3=B.'
R3 =

   0   1   0
  -1   0   1
   1   0   0
```

Importante: El símbolo (') en la expresión `R3=B.'` debe ingresarse seleccionando en el teclado la tecla que corresponde al apóstrofo.

- $R_4 = (B + iC)^*$

```
>> R4=(B+i*C)'
R4 =

   0 - 0i   1 - 1i   0 - 0i
  -1 - 0i   0 - 0i   1 - 1i
   1 - 1i   0 - 0i   0 - 0i
```

- $R_5 = BC$

```
>> R5=B*C
R5=B*C
R5 =

  -1   1   0
   0   0   1
   1   0   0
```

- $R_6 = \mathbf{1}_3$

```
>> R6=eye(3)
R6 =

Diagonal Matrix

   1   0   0
   0   1   0
   0   0   1
```

- $R_7 = a_{12}$

```
>> R7=A(1,2)
R7 = -1
```

- $R_8 = [a_{12}\ a_{13}]$

```
>> R8=A(1,2:3)
R8 =

  -1   0
```

- $R_9 = A_2 = [a_{21}\ a_{22}\ a_{33}]$

```
>> R9=A(2,:)
R9 =

  -1   2  -1
```

- $R_{10} = \det(A)$

```
>> R10=det(A)
R10 =  4.0000
```

- $R_{11} = A^{-1}$

```
>> R11=inv(A)
R11 =

   0.75000   0.50000   0.25000
   0.50000   1.00000   0.50000
   0.25000   0.50000   0.75000
```

- $R_{12} = \mathrm{tr}(-2D^*D)$

```
>> R12=trace(-2*D'*D)
R12 = -6
```

- $R_{13} = \langle A^3, v \rangle$

```
>> R13=A(:,3)'*v
R13 = -1
```

- $R_{14} = A \circ B = [a_{ij}b_{ij}]$

```
>> R14=A.*B
R14 =

   0   1   0
  -1   0  -0
   0  -1   0
```

- $R_{15} = A \otimes B = [a_{ij}b_{kl}]$

```
>> R15=kron(A,B)
R15 =

   0  -2   2  -0   1  -1   0  -0   0
   2   0   0  -1  -0  -0   0   0   0
   0   2   0  -0  -1  -0   0   0   0
  -0   1  -1   0  -2   2  -0   1  -1
  -1  -0  -0   2   0   0  -1  -0  -0
  -0  -1  -0   0   2   0  -0  -1  -0
   0  -0   0  -0   1  -1   0  -2   2
   0   0   0  -1  -0  -0   2   0   0
   0   0   0  -0  -1  -0   0   2   0
```

- $R_{16} = AB^{-1}$

```
>> R16=A/B
R16 =

  -0   2  -1
  -1  -1   1
   2   0   1
```

- $R_{17} = B^{-1}A$

```
>> R17=B\A
R17 =

  -1   2  -1
  -0  -1   2
   2  -2   2
```

- Descomposición espectral (en autovalores) $A = P\Lambda P^{-1}$:

```
>> [P,Lambda]=eig(A);
>> A-P*Lambda/P
ans =

   6.6613e-16   2.2204e-16  -1.1102e-16
  -4.4409e-16  -4.4409e-16   0.0000e+00
  -1.8645e-16   4.4409e-16  -4.4409e-16
```

- Descomposición DVS en valores singulares $A = U\Sigma V^*$:

```
>> [U,Sigma,V]=svd(A);
>> A-U*Sigma*V'
ans =

   4.4409e-16   2.2204e-16   6.9797e-17
   2.2204e-16   6.6613e-16   8.8818e-16
  -6.7008e-17  -6.6613e-16  -4.4409e-16
```

- Definir una matriz cero de 4×3:

```
>> Z=zeros(4,3)
Z =

   0   0   0
   0   0   0
   0   0   0
   0   0   0
```

- Definir una matriz de unos de 3×5:

```
>> O=ones(3,5)
O =

   1   1   1   1   1
   1   1   1   1   1
   1   1   1   1   1
```

- Consideremos nuevamente la matriz A anteriormente ingresada:
  - Extraer la diagonal principal (diagonal 0) de A:

```
>> D0=diag(A)
D0 =

   2
   2
   2
```

  - Definir una matriz diagonal $D_A \in \mathbb{C}^{3\times 3}$ con la misma diagonal 0 que A:

```
>> DA=diag(diag(A))
DA =

Diagonal Matrix

   2   0   0
   0   2   0
   0   0   2
```

  - Extraer las diagonales $1, -1, -2$ de A:

```
>> D1=diag(A,1)
D1 =

  -1
  -1

>> D_1=diag(A,-1)
D_1 =

  -1
  -1

>> D_2=diag(A,-2)
D_2 = 0
```

  - Definir una matriz $K \in \mathbb{C}^{4\times 4}$ cuya diagonal 2 es igual a la diagonal -1 de A y todas sus demás entradas son cero:

```
>> K=diag(diag(A,-1),2)
K =
```

```
   0   0  -1   0
   0   0   0  -1
   0   0   0   0
   0   0   0   0
```

Recomendación: Para obtener más información sobre el funcionamiento de cualquier comando estándar de Octave/MATLAB basta escribir:

```
>> help nombre_del_comando
```

en la ventana de comandos.

Descomposiciones y funciones matriciales

Consideremos la representación vectorial del polinomio $p(z) = z^7 - 1$ en términos de sus coeficientes $\mathbf{p} = [p_7\ p_6\ \cdots\ p_1\ p_0] = [1\ 0\ 0\ 0\ 0\ 0\ 0\ -1]$, podemos ingresarlo en el sistema como:

```
>> p=[1,zeros(1,6),-1]
p =

   1   0   0   0   0   0   0  -1
```

Para calcular las raíces de $p(z)$ podemos utlizar la secuencia de comandos:

```
>> r=roots (p)
r =

  -0.90097 + 0.43388i
  -0.90097 - 0.43388i
  -0.22252 + 0.97493i
  -0.22252 - 0.97493i
   1.00000 + 0.00000i
   0.62349 + 0.78183i
   0.62349 - 0.78183i
```

Es posible evaluar p en un elemento cualquiera de su dominio utilizando el comando `polyval`, en particular la expresión:

```
>> polyval(p,-1)
ans = -2
```

permite calcular el valor $p(-1)$. La evaluación de un polinomio puede llevarse a cabo también en un rango de valores, por ejemplo, la expresión:

```
>> t=-1:1/4:1;
>> polyval(p,t)
ans =

 Columns 1 through 8:

  -2.00000  -1.13348  -1.00781  -1.00006  -1.00000  -0.99994  -0.99219  -0.86652
```

```
 Column 9:

   0.00000
```

calcula los valores $p(x_k)$ correspondientes a la partición $1 = t_0 < t_1 < \cdots < t_9 = 1$ del intervalo $[0, 1]$, donde $t_k = -1 + k * h$, $h = 1/4$.

Podemos ahora calcular la matriz compañera $C(p) \in \mathbb{C}^{7\times 7}$ de $p(z)$.

```
>> Cp=fliplr(flipud(compan(p)))
Cp =

   0   1   0   0   0   0   0
   0   0   1   0   0   0   0
   0   0   0   1   0   0   0
   0   0   0   0   1   0   0
   0   0   0   0   0   1   0
   0   0   0   0   0   0   1
   1  -0  -0  -0  -0  -0  -0
```

La teoría de matrices compañeras de polinomios nos asegura que dada una raíz r de $p(z)$, $r \in \sigma(C(p)^\top)$ (r es un autovalor de $C(p)^\top$) con autovector correspondiente $v_r = [1\ r\ r^2\ \cdots\ r^{n-1}]^\top$, es posible verificar esto en particular para r_1, con las siguiente secuencia de comandos:

```
>> Cpt=Cp.';
>> v1=r(1).^(0:6)';
>> norm(Cpt*v1-r(1)*v1)
ans =    4.7018e-15
```

donde `norm(v)` es el comando utilizado para calcular el valor $\|v\|$ para $v \in \mathbb{C}^{n\times m}$. Podemos ahora calcular las matrices de diagonalización P y Q de $C(p)^\top$ y $C(p)$, respectivamente, utlizando matrices de Vandermonde con la siguiente secuencia de comandos:

```
>> P=fliplr(vander (r).');
>> Q=P.';
>> norm(diag(diag(P'*Cpt*P))-P'*Cpt*P)
ans =    1.4378e-14
>> norm(diag(diag(Q*Cp*Q'))-Q*Cp*Q')
ans =    1.4414e-14
```

De forma alternativa podemos calcular las matrices de diagonalización P y Q, los valores propios de $C(p)^\top$ y $C(p)$ utilizando las siguientes secuencias de comandos basadas en el comando `eig(A)` de Octave/MATLAB que calcula la descomposición espectral de $A \in \mathbb{C}^{n\times n}$:

```
>> [Q,l]=eig(Cp);
>> [P,lt]=eig(Cpt);
>> [Q,l]=eig(Cp);
>> norm(diag(diag(P'*Cpt*P))-P'*Cpt*P)
ans =    2.6413e-15
>> norm(diag(diag(Q'*Cp*Q))-Q'*Cp*Q)
ans =    2.6672e-15
```

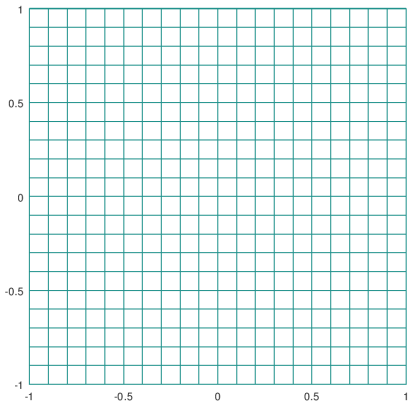
Lemniscatas y Pseudo-espectros

Consideremos nuevamente el polinomio de la sección anterior, una vez ingresado en el sistema con los comandos previos, podemos calcular las Lemniscatas de $p(z)$ para localizar sus raíces computacionalmente utilizando el siguiente procedimiento.

Cálculo de Lemniscata de p(z):

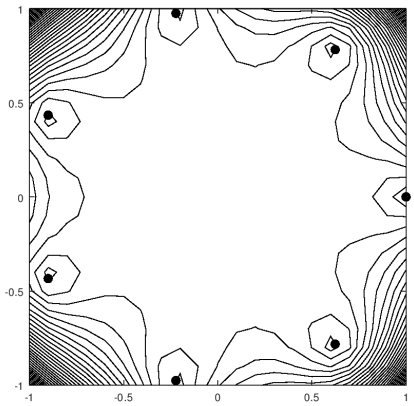
- Creación de malla inicial de la región $[-1,1]^2$ determinada por el Teorema de Gerschgorin aplicado a $C(p)^\top$:

```
>> [x,y]=meshgrid (-1:2/20:1);
>> mesh(x,y,zeros(size(x)));
>> view (2)
>> axis square
```



- Cálculo de Lemniscatas de $p(z)$ en la malla inicial:

```
>> p=[1,zeros(1,6),-1];
>> r=roots (p);
>> contour(x,y,abs(polyval(p,x+i*y)),32,'k')
>> hold on
>> plot(r,'k.','markersize',25)
>> axis square
```

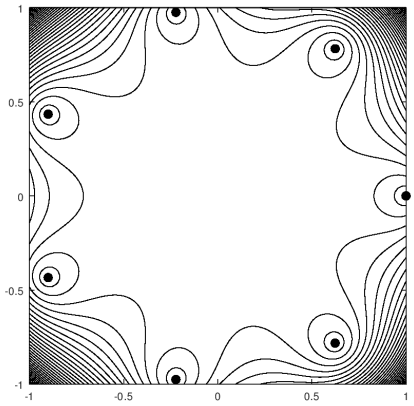


- Refinamiento de malla incial:

```
>> [x,y]=meshgrid (-1:2/100:1);
```

- Cálculo de Lemniscatas de $p(z)$ en la malla refinada:

```
>> close all
>> contour(x,y,abs(polyval(p,x+i*y)),32,'k')
>> hold on
>> plot(r,'k.','markersize',25)
>> axis square
```



El cálculo de Lemniscatas es importante para estudiar la estabilidad de sistemas dinámicos y de control discretos.

Definición: Dado $\varepsilon \geq 0$, se define el ε-Pseudo-espectro $\Lambda_\varepsilon(A)$ de $A \in \mathbb{C}^{n\times n}$ como el conjunto:

$$\Lambda_\varepsilon(A) = \left\{\lambda \in \mathbb{C} | \exists x \in \mathbb{C}^n \backslash \{0\}, E \in \mathbb{C}^{n\times n} : (A+E)x = \lambda x, \|E\| \leq \varepsilon\right\}$$

Podemos además calcular el Pseudo-espectro de $C(p)$ utilizando la siguiente secuencia de comandos basada en el comando `svd(A)` que calcula la descomposición en valores sigulares de A.

Cálculo de Pseudo-espectro de $C(p)$:

- Cálculo de malla de $[-1,1]^2$:

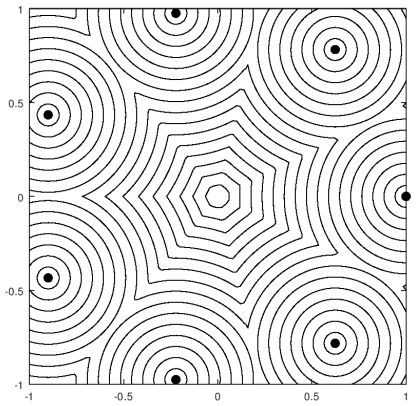
```
>> [x,y]=meshgrid (-1:2/100:1);
```

- Definición de la función `pspectra`:

```
>> pspectra=@(X)min(svd(X));
```

- Cómputo del Pseudo-espectro de $C(p)$:

```
>> Cp=fliplr(flipud(compan(p)));
>> N=size(x,1);
>> for k=1:N,for j=1:N,Z(k,j)=pspectra (Cp-(x(k,j)+i*y(k,j))*eye(7));
> end;end
>> contour(x,y,Z,16,'k')
>> l=eig(Cp);
>> hold on;
>> plot(l,'k.','markersize',25)
>> axis square
```



Al igual que el cómputo de Lemiscatas, el cómputo de Pseudo-espectros es importante para estudiar el compartamiento de sistemas dinámicos y de control discretos, importantes en la simulación numérica de procesos en ingeniería y ciencias.

1.1.2. Sistemas de Ecuaciones Lineales

En esta sección trabajaremos con algunas herramientas básicas de los programas orientados a matrices Matlab y Octave. Consideremos las siguientes matrices:

$$A = \begin{bmatrix} a & b & & e & b \\ b & a & & & \\ & & c & d & \\ e & & d & c & \\ b & & & & a \end{bmatrix} \tag{1.1.6}$$

$$f = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{bmatrix} \tag{1.1.7}$$

Donde supondremos que $a, b, c, d, e, b_1, ..., b_4 \sim N(0, 1)$. Tenemos que para ingresar las matrices anteriores en Matlab/Octave basta escribir:

```
>> a=randn;
>> b=randn;
>> c=randn;
>> d=randn;
>> e=randn;
>> A=[a b 0 e b;b a 0 0 0;0 0 c d 0;e 0 d c 0;b 0 0 0 a]
A =
-0.90411  -0.45550   0.00000  -1.63458  -0.45550
-0.45550  -0.90411   0.00000   0.00000   0.00000
 0.00000   0.00000   0.66103   0.25664   0.00000
-1.63458   0.00000   0.25664   0.66103   0.00000
-0.45550   0.00000   0.00000   0.00000  -0.90411
>> f=randn(5,1)
f =
  -1.49668
  -0.18249
  -0.44852
  -0.59657
   0.28638
>>
```

Para resolver el sistema de ecuaciones lineales

$$Ax = f$$

donde A y b estan dados por (1.1.6) y (1.1.7) respectivamente, basta con ejecutar la secuencia de comandos.

```
>> x=A\f
x =
   0.533964
  -0.067173
  -0.989978
   0.802238
  -0.585767
```

1.1.3. Normas Vectoriales y Matriciales

Consideremos un vector $x \in \mathbb{R}^7$ y una matriz $A \in \mathbb{R}^{7\times 7}$ generados al azar mediante la siguiente secuencia de comandos:

```
>> x=ceil(10*randn(7,1))
x =

    4
   12
   -3
    1
    4
  -10
  -10
>> A=floor(10*randn(7,7))
A =

  -24    2    8  -14  -12   16  -15
  -23  -12   -4   15   -8   -7   10
   -2    3   -2  -15    1    9   -3
   -6   18   -3    3   27    1   -5
   13    0    5   -4    7   13    8
   13    9   -1    9   14    8    1
  -18   -2  -20  -18   -4    4  -10
```

Las siguientes secuencias de comandos pueden utilizarse para calcular las normas $\|x\|_1$, $\|x\|_2$ y $\|x\|_\infty$, respectivamente:

```
>> norm(x,1)
ans =  44
>> norm(x,2)
ans =  19.647
>> norm(x,inf)
ans =  12
```

Para calcular las normas $\|A\|_1$, $\|A\|_2$ y $\|A\|_\infty$, respectivamente, pueden utilizarse las siguientes secuencias de comandos:

```
>> norm(A,1)
ans =  99
>> norm(A,2)
ans =  52.279
>> norm(A,inf)
ans =  91
```

1.2. Programación Lineal

Considerando el problema modelo del comando `glpk` de GNU Octave que consiste en un problema de optimización de la forma:

$$\min_{x\in\mathcal{S}} f(x) = c^\top x,$$
$$\mathcal{S} = \left\{ y \in \mathbb{R}^3 \;\middle|\; \begin{array}{l} Ay = b \\ y \geq d \end{array} \right\}$$

donde:

$$c = \begin{bmatrix} 10 \\ 6 \\ 4 \end{bmatrix}, A = \begin{bmatrix} 1 & 1 & 1 \\ 10 & 4 & 5 \\ 2 & 2 & 6 \end{bmatrix}, b = \begin{bmatrix} 100 \\ 600 \\ 300 \end{bmatrix}, d = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

La siguiente secuencia de comandos puede utilizarse para resolver el problema.

```
>> c = [10, 6, 4]';
>> A = [ 1, 1, 1;10, 4, 5;2, 2, 6];
>> b = [100, 600, 300]';
>> lb = [0, 0, 0]';
>> ub = [];
>> ctype = "UUU";
>> vartype = "CCC";
>> s = -1;
>> param.msglev = 1;
>> param.itlim = 100;
>>[xmin, fmin, status, extra] = ...
> glpk (c, A, b, lb, ub, ctype, vartype, s, param);
```

La solución aproximada $x \in \mathcal{S}$ que minimiza $f(x)$ puede visualizarse en la ventana de comandos utilizando la secuencia de comandos:

```
>> xmin
xmin =

   33.33333
   66.66667
    0.00000
```

Para resolver el problema derivado:

$$\min_{x \in \mathcal{S} \cap \mathbb{Z}^3} f(x) = c^\top x,$$

es suficiente modificar la variable `vartype` utilizando la siguiente secuencia de comandos:

```
>> vartype = "III";
```

la solución aproximada x del problema puede calcularse y visualizarse utilizando la siguiente secuencia de comandos:

```
[xmin, fmin, status, extra] = ...
> glpk (c, A, b, lb, ub, ctype, vartype, s, param);
Long-step dual simplex will be used
octave:16> xmin
xmin =

   33
   67
    0
```

1.3. Sistemas de Ecuaciones Lineales

1.3.1. Operaciones Elementales

Dado un sistema de ecuaciones lineales con coeficientes reales de la forma:

$$\begin{cases} E_1: \ a_{11}x_1 + \cdots + a_{1n}x_n = b_1 \\ E_2: \ a_{21}x_1 + \cdots + a_{2n}x_n = b_2 \\ \qquad\qquad \cdots \\ E_m: \ a_{m1}x_1 + \cdots + a_{mn}x_n = b_m \end{cases} \tag{1.3.1}$$

Donde cada E_j representa la j-ésima ecuación del sistema (1.11.1). Consideremos las tres operaciones elementales básicas de la forma:

1. $E_j \longleftarrow \alpha E_j$, $\alpha \in \mathbb{R}\backslash\{0\}$ ($\alpha \in \mathbb{R}\backslash\{0\}$ significa que α es un número real distinto de 0)
2. $E_k \longleftarrow E_k + \alpha E_j$, $\alpha \in \mathbb{R}\backslash\{0\}$
3. $E_k \longleftrightarrow E_j$

Considerando ahora la representación matricial de (1.11.1) de la forma:

$$Ax = b \tag{1.3.2}$$

donde $A \in \mathbb{R}^{m\times n}$ (A es una matriz de $m \times n$) y $b \in \mathbb{R}^{m\times 1}$.

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ldots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}, b = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} \tag{1.3.3}$$

La matriz aumentada de (1.3.2) puede escribirse en la forma:

$$\begin{bmatrix} A & | & b \end{bmatrix} \tag{1.3.4}$$

Tenemos que a cada operación elemental de ecuaciones en (1.11.1), corresponde una operación elemental de renglones en (1.3.4), que en el resto del curso representaremos con la misma notación.

Notación. En el resto del curso nos referiremos a las operaciones elementales (de ecuaciones de (1.11.1) o renglones de (1.3.4)) como operaciones elementales de tipo 1,2 o 3, seguún la numeración que hemos considerado en esta sección.

Observación. Notemos que cada operación elemental puede revertirse con una operación elemental del mismo tipo.

Ejemplo: La operación $E_j \longleftarrow \alpha E_j$, para $\alpha \in \mathbb{R}\backslash\{0\}$ produce una nueva ecuación E'_j de la forma:

$$E'_j: \ \alpha a_{j1}x_1 + \cdots + \alpha a_{jn}x_n = \alpha b_j$$

De modo que la operación $E'_j \longleftarrow \frac{1}{\alpha}E'_j$ produce una ecuación E''_j de la forma:

$$\begin{aligned} E''_j: \ & \frac{1}{\alpha}\alpha a_{j1}x_1 + \cdots + \frac{1}{\alpha}\alpha a_{jn}x_n = \frac{1}{\alpha}\alpha b_j \\ : \ & a_{j1}x_1 + \cdots + a_{jn}x_n = b_j \end{aligned}$$

y tenemos entonces que E_j'' y E_j describen la misma ecuación. Dado que E_j es una expresión arbitraria y suficientemente general, toda operación elemental del primer tipo puede anularse o revertirse con una operación elemental del primer tipo.

Ejemplo: Consideremos el siguiente sistema de ecuaciones:

$$\begin{cases} E_1: \ a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \\ E_2: \ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 \\ E_3: \ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \end{cases} \tag{1.3.5}$$

con matriz aumentada correspondiente:

$$A_b = \left[\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{array}\right] \tag{1.3.6}$$

La operación elemental de renglones $R_2 \longleftarrow \alpha R_2$, $\alpha \in \mathbb{R}\backslash\{0\}$ de A_b puede realizarse multiplicando la matriz:

$$E_2(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.3.7}$$

por la izquierda de A_b, en efecto:

$$E_2(\alpha)A_b = \left[\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & b_1 \\ \alpha a_{21} & \alpha a_{22} & \alpha a_{23} & \alpha b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{array}\right]$$

La operación elemental de renglones $R_3 \longleftarrow R_3 + \alpha R_1$, $\alpha \in \mathbb{R}\backslash\{0\}$ de A_b puede realizarse multiplicando la matriz:

$$E_{31}(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \alpha & 0 & 1 \end{bmatrix} \tag{1.3.8}$$

por la izquierda de A_b, en efecto:

$$E_{31}(\alpha)A_b = \left[\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} + \alpha a_{11} & a_{32} + \alpha a_{12} & a_{33} + \alpha a_{13} & b_3 + \alpha b_1 \end{array}\right]$$

La operación elemental de renglones $R_1 \longleftrightarrow R_2$ de A_b puede realizarse multiplicando la matriz:

$$E_{12} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.3.9}$$

por la izquierda de A_b, en efecto:

$$E_{12}A_b = \left[\begin{array}{ccc|c} a_{21} & a_{22} & a_{23} & b_2 \\ a_{11} & a_{12} & a_{13} & b_1 \\ a_{31} & a_{32} & a_{33} & b_3 \end{array}\right]$$

Ejercicio para el lector: Verificar que en efecto, cada operación elemental puede revertirse con una opración elemental del mismo timpo.

Ejercicio para el lector: Considerando la matriz aumentada (1.3.4), verificar que existe una matriz $\mathbf{E}_{jk}(\alpha)$ que multiplicada por la izquierda de (1.3.4), realiza la operación elemental por renglones, correspondiente a una operación elemental de tipo 2.

1.4. Normas Vectoriales y Matriciales

Normas vectoriales

Definición. Una norma vectorial en $\mathbb{R}^n$ es una función $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ con las siguientes propiedades:

1. $\|x\| \geq 0$, $x \in \mathbb{R}^n$
2. $\|x\| = 0$, ssi $x = 0$
3. $\|\alpha x\| = |\alpha|\|x\|$, $\alpha \in \mathbb{R}, x \in \mathbb{R}^n$
4. $\|x + y\| \leq \|x\| + \|y\|$, $x, y \in \mathbb{R}^n$

Notación. En este curso identificaremos $\mathbb{R}^n$ con $\mathbb{R}^{n\times 1}$, es decir, consideraremos los vectores en $\mathbb{R}^n$, como matrices columna de $n \times 1$. En general, un vector arbitrario $x \in \mathbb{R}^n$ se representará como:

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

Una familia de normas vectoriales que estudiaremos con frecuencia en este curso son las normas de la forma.

$$\|x\|_p = \left(\sum_{j=1}^{n} |x_j|^p \right)^{1/p}, x \in \mathbb{R}^n \tag{1.4.1}$$

Un tipo especial de norma vectorial también importante es la norma definida por la expresión.

$$\|x\|_\infty = \max_{1\leq j\leq n} |x_j|, x \in \mathbb{R}^n \tag{1.4.2}$$

Propiedad. Desigualdad Cauchy-Bunyakovsky-Schwarz para sumas) Dados $x, y \in \mathbb{R}^n$:

$$|x \cdot y| = |x^\top y| = \left| \sum_{j=1}^{n} x_j y_j \right| \leq \|x\|_2 \|y\|_2$$

donde $x^\top$ denota la transpuesta de x, y donde $x \cdot y$ denota el producto punto o producto escalar entre x e y.

Definición La distancia d_* inducida por la norma $\|\cdot\|_*$, está definida por la expresión $d_*(x, y) = \|x - y\|_*$, $x, y \in \mathbb{R}^n$.

Definición Una sucesión $\{x_n\}_{n\geq 1} \subset \mathbb{R}^n$ se dice que converge a $x \in \mathbb{R}^n$ con respecto a distancia d_*, si $\lim_{n\to\infty} d_*(x_n, x) = 0$.

Propiedad. Para $x \in \mathbb{R}^n$,

$$\|x\|_\infty \leq \|x\|_2 \leq \sqrt{n}\|x\|_\infty$$

Observación. Dado $x \in \mathbb{R}^n$:

$$\|x\|_2^2 = \sum_{j=1}^{n} |x_j|^2 \leq \sum_{j=1}^{n} \sum_{k=1}^{n} |x_j||x_k| = \left(\sum_{j=1}^{n} |x_j| \right)^2 = \|x\|_1^2$$

$\implies \|x\|_2 \leq \|x\|_1$.

Ejercicio para el lector. Verificar que $d_2(x, y) \leq d_1(x, y)$ para $x, y \in \mathbb{R}^n$.

1.4.1. Normas matriciales

Definición Una norma matricial en $\mathbb{R}^{n\times n}$ (el conjunto de matrices reales de $n \times n$), es una función $\|\cdot\| : \mathbb{R}^{n\times n} \to \mathbb{R}$ con las siguientes propiedades:

1. $\|A\| \geq 0$, $A \in \mathbb{R}^{n\times n}$
2. $\|A\| = 0$, ssi $A = \mathbf{0}$ (A es igual a la matriz $\mathbf{0}$)
3. $\|\alpha A\| = |\alpha|\|A\|$, $\alpha \in \mathbb{R}, A \in \mathbb{R}^{n\times n}$
4. $\|A + B\| \leq \|A\| + \|B\|$, $A, B \in \mathbb{R}^{n\times n}$
5. $\|AB\| \leq \|A\|\|B\|$, $A, B \in \mathbb{R}^{n\times n}$

La distancia d inducida en $\mathbb{R}^{n\times n}$ por la norma $\|\cdot\|$ se define como $d(A, B) = \|A-B\|$, $A, B \in \mathbb{R}^{n\times n}$.

Propiedad. Si $\|\cdot\|$ es una norma vectorial en $\mathbb{R}^n$, entonces

$$\|A\| = \max_{\|x\|=1} \|Ax\| \tag{1.4.3}$$

es una norma matricial.

Notación. Las normas matriciales de la forma (1.4.3) reciben el nombre de normas inducidas por la norma vectorial $\|\cdot\|$.

Observación. Para cualquier $y \in \mathbb{R}^n\backslash\{0\}$ ($y \neq \mathbf{0}$), tenemos que $\|(1/\|y\|)y\| = \|y\|/\|y\| = 1 \Longrightarrow$ para cualquier $A \in \mathbb{R}^{n\times n}$:

$$\frac{1}{\|y\|}\|Ay\| = \left\|A\left(\frac{1}{\|y\|}y\right)\right\| \leq \max_{\|z\|=1} \|Az\| = \|A\|$$

$\Longrightarrow$

$$\|Ay\| \leq \|A\|\|y\|$$

Propiedad. Para cada $A = [a_{jk}] \in \mathbb{R}^{n\times n}$,

$$\|A\|_\infty = \max_{\|x\|_\infty=1} \|Ax\|_\infty = \max_{1\leq j\leq n} \sum_{k=1}^{n} |a_{jk}|$$

Ejercicio para el lector. Dada una matriz $A = [a_{jk}] \in \mathbb{R}^{n\times n}$, sea

$$r(A) = \sum_{j=1}^{n}\sum_{k=1}^{n} |a_{jk}|,$$

(a) Probar o refutar que $\|Ax\|_2 \leq r(A)\|x\|_2$, $x \in \mathbb{R}^n$.

(b) Probar o refutar que $\|A\|_2 \leq r(A)$.

1.5. Factorizaciones Matriciales

1.5.1. Factorización LU

Considerando un sistema de ecuaciones lineales representado en forma matricial:

$$Ax = b \tag{1.5.1}$$

donde $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$ están dados, y $x \in \mathbb{R}^n$ debe determinarse resolviendo el sistema (1.11.1).

Observación. Como se establece en §6.1 y §6.5 del libro de texto [15], al aplicar un algoritmo de eliminación gaussiana para resolver (1.11.1) se necesitan $\mathcal{O}(n^3/3)$ operaciones aritméticas.

Notación. En este curso, $\mathbb{GL}_n$, $\mathbb{D}(n)$, $\mathbb{I}(n)$ y $\mathbb{S}(n)$ denotarán los conjuntos de matrices invertibles, diagonales, triangulares inferiores, y triangulares superiores en $\mathbb{R}^{n\times n}$ (en el sentido estudiado en MM 211 vectores y matrices), respectivamente.

Notación: En este curso, al considerar un sistema de ecuaciones $Ax = b$ se asumirá, a menos que se especifique lo contrario, que A,x son compatibles en el producto, y que Ax y b pertenencen al mismo tipo de vectores (matrices).

Propiedad. Si la eliminación gaussiana se puede realizar en el sistema lineal $Ax = b$ sin intercambios de renglones, entonces la matriz A se puede factorizar como el producto de una matriz triangular $L = [l_{jk}] \in \mathbb{I}(n)$ y una matriz triangular superior $U = [u_{jk}] \in \mathbb{S}(n)$, es decir, $A = LU$, donde $l_{jj} = 1$, para $j = 1, \ldots, n$.

Consderando el siguiente procedimiento para resolver us sistema $Ax = b$ tal que A puede factorizarse como $A = LU$:

1. Solución del sistema $Ax = b$ por factorización LU:

   a) Definir $y = Ux$ (consideración teórica, no se realiza ninguna operación).

   b) Resolver $Ly = b$ (se requieren $\mathcal{O}(n^2)$ operaciones dado que $L \in \mathbb{I}(n)$).

   c) Resolver $Ux = y$ (se requieren $\mathcal{O}(n^2)$ operaciones dado que $U \in \mathbb{S}(n)$).

Observación. Al aplicar la factorización LU el número de operaciones necesario para resolver el sistema $Ax = b$ se reduce a partir de $\mathcal{O}(n^3/3)$ a $\mathcal{O}(2n^2)$.

1.6. Matrices Simétricas Positivas Definidas

En la clase de vectores y matrices se estableción que $A \in \mathbb{R}^{n\times n}$ es simétrica si $A^\top = A$.

Definición. Decimos que una matriz simétrica $A \in \mathbb{R}^{n\times n}$ es positiva semi-definida o **SPSD**, si para cada $x \in \mathbb{R}^n$ se cumple que $x^\top Ax \geq 0$. Si $A \in \mathbb{R}^{n\times n}$ es una matriz simétrica que cumple la restricción $x^\top Ax > 0$ para cada $x \in \mathbb{R}^n\backslash\{0\}$, se dice que A es simétrica positiva definida o **SPD**.

Notación. En este curso también utilizaremos la notación $A > 0$ para denotar que la matriz $A \in \mathbb{R}^{n\times n}$ es SPD, las matrices SPD también serán llamadas definidas positivas (**DP**) o positivas definidas (**PD**) en este curso.

Definición. Una **primera submatriz principal** de una matriz A es una matriz A_k de la forma

$$A_k = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kk} \end{bmatrix}$$

para algunas $1 \leq k \leq n$.

Propiedades. Si $A \in \mathbb{R}^{n\times n}$ es una matriz SPD:

1. $A \in \mathbb{GL}_n$;
2. $a_{ii} > 0$, para cada $i = 1, 2, \cdots, n$;
3. $\text{máx}_{1\leq k,j\leq n} |a_{kj}| \leq \text{máx}_{1\leq i\leq n} |a_{ii}|$;
4. $(a_{ij})^2 < a_{ii}a_{jj}$ para cada $i \neq j$

Propiedad. Dada una matriz $A \in \mathbb{R}^{n\times n}$, las siguientes proposiciones son equivalentes:

1. A es SPD;
2. $\det(A_k) > 0$ para cada una de sus primeras submatrices principales;
3. Se puede realizar eliminación gaussiana sin interacambios de renglones en el sistema $Ax = b$ con todos los elementos pivote positivos.
4. A se puede factorizar como $A = LDL^\top$, donde $L \in \mathbb{I}(n)$, $D \in \mathbb{D}(n)$, con $d_{jj} > 0$ y $l_{jj} = 1$, para cada $1 \leq j \leq n$.
5. A se puede factorizar en la forma $A = LL^\top$, donde $L = [l_{jk}] \in \mathbb{I}(n)$ y $l_{jj} \neq 0$ para cada $1 \leq j \leq n$.

Ejemplo computacional. Consideremos la matriz:

$$U = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

Es claro que la matriz:

$$A = U^\top U = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

es SPD dado que para cada $x \in \mathbb{R}^3\backslash\{0\}$:

$$x^\top Ax = x^\top U^\top Ux = (Ux)^\top (Ux) = (Ux) \cdot (Ux) = \|Ux\|_2^2 > 0$$

dado que $Ux \neq 0$, debido a que $x \neq 0$, y que $U \in \mathbb{GL}_n$ como consecuencia de que $\det(U) = 1 > 0$. Podemos ahora calcular las determinantes de las primeras submatrices principales A_1, A_2, A_3

$$A_1 = \begin{bmatrix} 1 \end{bmatrix}, A_1 = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}, A_3 = A$$

con las siguientes secuencias de comandos:

```
>> A1=A(1,1)
A1 =  1
>> A2=A(1:2,1:2)
A2 =
```

```
   1  -1
  -1   2

>> A3=A(1:3,1:3)
A3 =

   1  -1   0
  -1   2  -1
   0  -1   2
>> det(A1)
ans =  1
>> det(A2)
ans =  1
>> det(A3)
ans =  1
```

Es posible observar que todos los determinantes son positivos, como lo establecen las propiedades previas.

1.7. Factorización de Cholesky

Propiedad. Si $A \in \mathbb{R}^{n\times n}$ es una matriz simétrica, tal que se puede aplicar eliminación gaussiana sin intercambios de renglones al sistema $Ax = b$ correspondiente. Entonces A se puede factorizar como $A = LDL^\top$, donde $L \in \mathbb{I}(n)$, $D \in \mathbb{D}(n)$ con $l_{jj} = 1$, para cada $1 \leq j \leq n$.

Algunas Observaciones:

1. Por propiedades previas de matrices SPD, y por la propiedad anterior, tenemos que para cada $A \in \mathbb{R}^{n\times n}$ SPD, $A = LDL^\top \Longrightarrow$

$$\det(A) = \det(LDL^\top) = \det(L)\det(D)\det(L^\top) = 1 \cdot \det(D) \cdot 1 = \det(D)$$

2. Dado que $L \in \mathbb{GL}_n$ como consecuencia de que $\det(L) = 1 \neq 0$, $A = LDL^\top \Longrightarrow$

$$L^{-1}A(L^{-1})^\top = D$$

de manera que $D = L^{-1}A(L^{-1})^\top$ es claramente SPD, dado que para cada $x \in \mathbb{R}^n\backslash\{0\}$:

$$x^\top Dx = x^\top L^{-1}A(L^{-1})^\top x = ((L^{-1})^\top x)^\top A((L^{-1})^\top x) > 0$$

dado que A es SPD, y dado que $(L^{-1})^\top x \neq 0$ siempre que $x \neq 0$ debido que $L \in \mathbb{GL}_n$.

Por propiedades de matrices SPD $D = L_D L_D^\top$ para $L_D \in \mathbb{I}(n)$, y por las ecuaciones previas, tenemos entonces que,

$$A = LL_D L_D^\top L^\top = LL_D(LL_D)^\top$$

si definimos $L_A = LL_D$ tenemos entonces que.

$$A = L_A L_A^\top$$

Definición. Dada una matriz $A\mathbb{R}^{n\times n}$ SPD, la factorización $LL^\top = A$ con $L \in \mathbb{I}(n)$, recibe el nombre de factorización de **Cholesky** de A.

1.8. Series de Neumann

Consideremos una norma matricial (inducida) $\|\cdot\| : \mathbb{R}^{n\times n} \to \mathbb{R}$ arbitraria.
Dada un matriz $X \in \mathbb{R}^{n\times n}$, sea $S_N(X)$ la matriz definida por la expresión:

$$S_N(X) = \sum_{k=0}^{N} X^k$$

Definición. Dada una matriz $A \in \mathbb{R}^{n\times n}$, se define la serie matricial de Neumann $\mathbb{N}(A)$ de A como la serie matricial dada por la expresión:

$$\mathbb{N}(A) = \sum_{k=0}^{\infty} A^k$$

Si suponemos que $\|A\| < 1$ (donde I denota la identidad compatible en la suma con A), tenemos que:

$$\begin{aligned}(I-A)\left(\sum_{k=0}^{N} A^k\right) &= (I-A)(I+A+A^2+\cdots+A^{N-1}+A^N)\\ &= I - A^{N+1}\end{aligned}$$

$\Longrightarrow$

$$\begin{aligned}\left\|(I-A)\left(\sum_{k=0}^{N} A^k\right) - I\right\| &= \left\|I - A^{N+1} - I\right\|\\ &= \|-A^{N+1}\|\\ &\leq \|A\|\overset{N+1}{\underset{N\to\infty}{\to}}0\end{aligned}$$

$\Longrightarrow S_N(A) \to \mathbb{N}(A) = (I-A)^{-1}$ ($S_N(A)$ converge a $\mathbb{N}(A) = (I-A)^{-1}$ con respecto a la distancia inducida por $\|\cdot\|$). Consideremos ahora una matriz $A \in \mathbb{R}^{n\times n}$ tal que $\|I - A\| < 1$. Por las observaciones previas, tenemos que.

$$S_N(I-A) \to \mathbb{N}(A) = (I-(I-A))^{-1} = A^{-1}$$

Consideremos ahora $A, B \in \mathbb{GL}_n$ tales que $\|I - BA\| < 1$ y $\|I - AB\| < 1$, tenemos entonces que:

$$S_N(I-AB) \to (AB)^{-1} = B^{-1}A^{-1}$$

$\Longrightarrow$

$$BS_N(I-AB) \to BB^{-1}A^{-1} = A^{-1}$$

De forma similar se cumple también que:

$$S_N(I-BA) \to (BA)^{-1} = A^{-1}B^{-1}$$

$\Longrightarrow$

$$S_N(I-BA)B \to A^{-1}B^{-1}B = A^{-1}$$

Las expresiones normadas anteriores nos permiten pensar en B como una aproximación de A^{-1}, dado que en el caso exacto $B = A^{-1}$, se tendría que $\|I - BA\| = \|I - I\| = \|\mathbf{0}\| = 0 < 1$, y de forma similar $\|I - AB\| = \|I - I\| = \|\mathbf{0}\| = 0 < 1$. De manera que si consideramos el sistema de ecuaciones $Ax = b$, $y = BAx = Bb$ sería una primer aproximación de la solución x del sistema, y podemos refinar (mejorar la precisión) de la aproximación $y \approx x$ utilizando el esquema iterativo:

$$S_N(I - BA)y = S_N(I - BA)Bb \to A^{-1}B^{-1}Bb = A^{-1}b = x$$

Ejercicio para el lector: Verificar en detalle que $S_N(I - BA)y \to x$ con respecto a la distancia inducida por la norma vectorial correpondiente (que a su vez induce la norma vectorial en consideración), es decir, demostrar que:

$$\lim_{N\to\infty} \|S_N(I - BA)y - x\| = 0$$

bajo las hipótesis consideradas en los argumentos anteriores.

Ejercicio para el lector: Probar o refutar que para cada $A, B \in \mathbb{R}^{n\times n}$ y cada entero $k \geq 0$:

$$A(I - BA)^k = (I - AB)^k A$$

Ejercicio para el lector: Dadas $A, B \in \mathbb{GL}_n$ y un vector $b \in \mathbb{R}^n$, diseñar un algoritmo iterativo que permita calcular las siguientes expresiones en términos de productos matriz vector, evitando en la medida de lo posible productos matriz-matriz explícitamente calculados:

$$y_N = S_N(I - BA)Bb$$
$$z_N = BS_N(I - AB)b$$

1.9. Prinicipios de Métodos de Gradiente Conjugado: Un Enfoque Iterativo

Consideremos una matriz $A \in \mathbb{R}^{n\times n}$ arbitraria y un vector arbitrario $b \in \mathbb{R}^n$, y consideremos el funcional cuadrático:

$$q_{A,b}(x) = x^\top Ax - 2x^\top b$$

Propiedad. Si $A \in \mathbb{R}^{n\times n}$ es SPD, resolver el sistema $Ax = b$ equivale a resolver el siguiente problema de programación cuadrática.

$$\min_{x\in\mathbb{R}^n} q_{A,b}(x) \tag{1.9.1}$$

La solución de los problemas cuadráticos de la forma (2.2.1) correspondientes a un sistema $Ax = b$ con matriz de coeficientes A SPD, pueden resolverse implementando métodos de gradiente conjugado que producen una secuencia $\{x_k\}_{k\geq 0} \subset \mathbb{R}^n$ tal que $x_k \to x = A^{-1}b$ y donde cada elemento x_k de la sucesión está dado por la expresión:

$$x_k = x_{k-1} + \alpha_{k-1}p_{k-1},\ k \geq 1$$

donde $\alpha_k \in \mathbb{R}$ y $p_k \in \mathbb{R}^n$ se calculan de acuerdo a criterios de optimzación que serán estudiados en detalle más adelante, y $x_0 \in \mathbb{R}^n$ es un elemento que se utiliza para inicializar el esquema iterativo.

1.10. Prinicipios de Despomposiciones ortogonales y Descomposición en valores singulares

1.10.1. Descomposición QR

Iniciamos esta sección recordando que por el teorema de ortogonalizacón de Gram-Schmidt dados $x_1, \ldots, x_m \in \mathbb{R}^n$ linealmente independientes, existen $q_1, \ldots, q_m \in \mathbb{R}^n$ ortonormales tales que:

$$\text{gen}\,\{x_1, \ldots, x_m\} = \text{gen}\,\{q_1, \ldots, q_m\}$$

Además si $X, Q \in \mathbb{R}^{n\times m}$ son las matrices definidas por las expresiones:

$$X = \begin{bmatrix} | & & | \\ x_1 & \cdots & x_m \\ | & & | \end{bmatrix}$$

$$Q = \begin{bmatrix} | & & | \\ q_1 & \cdots & q_m \\ | & & | \end{bmatrix}$$

Se cumple que:

$$\begin{aligned} Q^\top Q &= I_m \\ Q^\top X &= R \\ QR &= X \end{aligned}$$

donde I_m representa la matriz identidad en $\mathbb{R}^{m\times m}$ y $R \in \mathbb{S}(m)$. La descomposición ortogonal $X = QR$ recibe el nombre de descomposición QR de X.

Notación. Una matriz $U \in \mathbb{R}^{n\times m}$ que cumple $U^\top U = I_m$ se denominará **matriz ortogonal** en este curso.

Notación. El conjunto de matrices ortogonales en $\mathbb{R}^{n\times m}$, será denotado por $\mathbb{O}(n, m)$, en el caso de las matrices ortogonales en $\mathbb{R}^{n\times n}$, en algunas ocasiones se escribirá $\mathbb{O}(n)$ en lugar de $\mathbb{O}(n, n)$.

Ejercicio para el lector: Aplicar el teorema de ortogonalización de Gram-Schmidt para verificar que toda matriz $X \in \mathbb{R}^{n\times m}$ con columnas linealmente independientes tiene una factorización QR.

Preguna para el lector: Es posible que toda matriz $X \in \mathbb{R}^{n\times n}$ tenga una factorización QR?

Matrices de Permutación

Un tipo especial de matrices ortogonales en $\mathbb{R}^{n\times n}$, son las matrices de permutación, las cuales se obtienen a partir de la matriz identidad $I \in \mathbb{R}^{n\times n}$ permutando sus columnas (o renglones).

Ejemplos. En $\mathbb{R}^{4\times 4}$ las siguientes matrices son ejemplos de matrices de permutación:

$$Q_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},\ Q_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix},\ Q_3 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix},\ Q_4 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Ejercicio para el lector. Calcular el número de matrices de permutación en $\mathbb{R}^{n\times n}$.

1.10.2. Descomposición en valores singulares

En este sección consideraremos por primera vez uno de los teoremas fundamentales del análisis matricial numérico.

Propiedad (Teorema fundamental de descomposición en valores singulares *SVD*) Si $A \in \mathbb{R}^{m\times n}$, entonces existen matrices ortogonales $U \in \mathbb{R}^{m\times m}$ y $V \in \mathbb{R}^{n\times n}$ tales que:

$$U^\top AV = \Sigma = \text{diag}(\sigma_1, \ldots, \sigma_p) \in \mathbb{R}^{m\times n},\ \ p = \text{mín}\{m, n\}$$

donde $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p \geq 0$.

Donde $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_p) \in \mathbb{R}^{m\times n}$ es una matriz en $\mathbb{R}^{m\times n}$ determinada por la expresión:

$$\Sigma_{jk} = \begin{cases} \sigma_j, & j = k \\ 0, & j \neq k \end{cases}, 1 \leq j \leq p$$

1.10.3. Cómputo de Factorizaciónes Matriciales LU y de Cholesky

Considerando una matriz $A_{h,N} \in \mathbb{R}^{N\times N}$ de la forma:

$$\mathbf{A}_{h,N} = -\frac{1}{h^2}\begin{bmatrix} -2 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & -2 & 1 & 0 \\ 0 & \cdots & 0 & 0 & 1 & -2 & 1 \\ 0 & \cdots & 0 & 0 & 0 & 1 & -2 \end{bmatrix}$$

para $h > 0$. Es posible generar la matriz $A_{h,N}$ utilizando la siguiente secuencia de comandos en Octave:

```
>> A=@(h,N)(-1/(h^2))*spdiags(ones(N,1)*[1 -2 1],-1:1,N,N);
```

En particular, para $h = 1/10$ y $N = 9$, la matriz $A_{1/10,9} \in \mathbb{R}^{9\times 9}$ puede calcularse utilizando la fución A recién definida, utilizando la siguiente secuencia de comandos:

```
>> A9=A(.1,10)
A9 =

Compressed Column Sparse (rows = 10, cols = 10, nnz = 28 [28%])

  (1, 1) ->  200.00
  (2, 1) -> -100.00
  (1, 2) -> -100.00
  (2, 2) ->  200.00
  (3, 2) -> -100.00
  (2, 3) -> -100.00
  (3, 3) ->  200.00
  (4, 3) -> -100.00
  (3, 4) -> -100.00
  (4, 4) ->  200.00
```

```
  (5, 4) -> -100.00
  (4, 5) -> -100.00
  (5, 5) ->  200.00
  (6, 5) -> -100.00
  (5, 6) -> -100.00
  (6, 6) ->  200.00
  (7, 6) -> -100.00
  (6, 7) -> -100.00
  (7, 7) ->  200.00
  (8, 7) -> -100.00
  (7, 8) -> -100.00
  (8, 8) ->  200.00
  (9, 8) -> -100.00
  (8, 9) -> -100.00
  (9, 9) ->  200.00
  (10, 9) -> -100.00
  (9, 10) -> -100.00
  (10, 10) ->  200.00
```

La matriz `A9` está en formato esparcido (los coeficientes iguales a cero no se almacenan en memoria para ahorar espacio en memoria), para visualizar la matriz `A9` en formato denso, puede escribirse (aunque para ahorrar memoria no se recomienda, especialmente para matrices de gran tamaño) la siguiente secuencia de comandos:

```
>> full(A9)
ans =

 Columns 1 through 6:

   200.00000  -100.00000     0.00000     0.00000     0.00000     0.00000
  -100.00000   200.00000  -100.00000     0.00000     0.00000     0.00000
     0.00000  -100.00000   200.00000  -100.00000     0.00000     0.00000
     0.00000     0.00000  -100.00000   200.00000  -100.00000     0.00000
     0.00000     0.00000     0.00000  -100.00000   200.00000  -100.00000
     0.00000     0.00000     0.00000     0.00000  -100.00000   200.00000
     0.00000     0.00000     0.00000     0.00000     0.00000  -100.00000
     0.00000     0.00000     0.00000     0.00000     0.00000     0.00000
     0.00000     0.00000     0.00000     0.00000     0.00000     0.00000
     0.00000     0.00000     0.00000     0.00000     0.00000     0.00000

 Columns 7 through 10:

     0.00000     0.00000     0.00000     0.00000
     0.00000     0.00000     0.00000     0.00000
     0.00000     0.00000     0.00000     0.00000
     0.00000     0.00000     0.00000     0.00000
     0.00000     0.00000     0.00000     0.00000
  -100.00000     0.00000     0.00000     0.00000
   200.00000  -100.00000     0.00000     0.00000
```

```
  -100.00000   200.00000  -100.00000     0.00000
     0.00000  -100.00000   200.00000  -100.00000
     0.00000     0.00000  -100.00000   200.00000
```

Las matrices de la forma $A_{h,N}$ tienen aplicaciones en la solución numérica de ecuaciones diferenciales parciales y ordinarias aplicadas en ciencias e ingeniería.

1.10.4. Ejercicio de Laboratorio 1

Calcular la matriz $A_{\frac{1}{1000},999} \in \mathbb{R}^{999\times 999}$.

Factorización LU

Aplicando el programa `LU.m` definido por el código Octave.

```
function [L,U]=LU(A)
% Descomposicion LU
% A : Matriz
% L : Matrix triangular inferior t.q. A = LL'
% Programador: Fredy Vides
n=size(A,1);
L=eye(n);
U=A(1,:);
for k=1:n,
    U(k,k)=A(k,k)-L(k,1:(k-1))*U(1:(k-1),k);
    for j=(k+1):n,
        U(k,j)=(A(k,j)-L(k,1:(k-1))*U(1:(k-1),j));
    end
    for i=(k+1):n
        L(i,k)=(A(i,k)-L(i,1:(k-1))*U(1:(k-1),k))/U(k,k);
    end
end
```

Calcular:

1. La factorización $LU = A_{\frac{1}{1000},999}$
2. El error normado $\|LU - A_{\frac{1}{1000},999}\|_\infty$ correspondiente.

Solución. Para calcular la factorización LU de A en Octave es posible aplicar el programa `LU.m` utilizando la siguiente secuencia de comandos:

1. Generar las matriz de prueba $A_{h,N}$ para el algoritmo:

```
>> A=@(h,N)(-1/(h^2))*spdiags(ones(N,1)*[1 -2 1],-1:1,N,N);
>> AhN=A(1/1000,999);
```

2. Calcular la factorización LU de la matriz $A_{h,N}$ correspondiente, estimando el tiempo de ejecución del programa con los comandos `tic-toc` de Octave:

```
>> tic,[L,U]=LU(AhN);toc
Elapsed time is 23.2845 seconds.
```

3. Estimar el error absoluto de aproximación de la factorización LU aproximada de $A_{h,N}$ producida por el programa, con respecto a la norma $\|\cdot\|_\infty$, utilizando la siguiente secuencia de comandos:

```
>> norm(AhN-L*U,inf)
ans =  0.00000000011642
```

4. También es posible estimar el error relativo de aproximación de la factorización LU aproximada de $A_{h,N}$ producida por el programa, con respecto a la norma $\|\cdot\|_\infty$, utilizando la siguiente secuencia de comandos:

```
>> norm(AhN-L*U,inf)/norm(AhN,inf)
ans =    2.9104e-17
```

Factorización de Cholesky

Aplicando el programa `Chole.m` definido por el código Octave.

```
function L=Chole(A)
% L = Chole (A)
% Descomposicion elemental de Cholesky
% A : Matriz SPD
% L : Matrix triangular inferior t.q. A = LL'
% Programador: Fredy Vides
n=size(A,1);
L(1,1)=sqrt(A(1,1));
L(2:n,1)=A(2:n,1)/L(1,1);
for k=2:n,
    L(k,k)=sqrt(A(k,k)-L(k,1:(k-1))*L(k,1:(k-1))');
    for i=(k+1):n
        L(i,k)=(A(i,k)-L(i,1:(k-1))*L(k,1:(k-1))')/L(k,k);
    end
end
```

Calcular:

1. La factorización $LL^\top = A_{\frac{1}{1000},999}$
2. El error normado $\|LL^\top - A_{\frac{1}{1000},999}\|_\infty$ correspondiente.

1.10.5. Ejercicio de Laboratorio 2

Modificar los programas `LU.m` y `Chole.m` definidos anteriormente para realizar un cómputo más eficiente (en el sentido del número de operaciones realizadas, uso de la memoria y tiempo de cómputo) en el caso especial de matrices tridiagonales como las matrices $A_{h,N}$ definidas anteriormente.

Resolver nuevamente le Ejercicio de Laboratorio 1 con las nuevas versiones de `LU.m` y `Chole.m`, justificando de forma teórica o computacional que las nuevas versiones de los programas propuestas por usted, son en efecto más eficientes.

1.11. Métodos Iterativos de Solución de Sistemas de Ecuaciones

1.11.1. Principios generales

Considerando un sistema de ecuaciones lineales representado en forma matricial:

$$Ax = b \tag{1.11.1}$$

donde $A = [a_{jk}] \in \mathbb{R}^{n\times n}$, $b = [b_j] \in \mathbb{R}^n$ están dados, y $x = [x_k] \in \mathbb{R}^n$ debe determinarse resolviendo el sistema (1.11.1).

Definición 1.11.1. Dada $A \in \mathbb{GL}_n$, la representación $A = M - N$ para $M, N \in \mathbb{R}^{n\times n}$, se denomia una **separación** aditiva o **separación** de A.

Definición 1.11.2. Dada una matriz $X \in \mathbb{R}^{n\times n}$, se denota $\lambda(X)$ el conjunto de números determinados por la expresión:

$$\lambda(X) = \{\lambda \in \mathbb{C} : \det(X - \lambda I) = 0\}. \tag{1.11.2}$$

El conjunto $\lambda(X)$ recibe el nombre de **espectro de** X en este curso.

Definición 1.11.3. Dada una matriz $X \in \mathbb{R}^{n\times n}$, se denota por $\rho(X)$ el número definido por la expresión:

$$\rho(X) = \text{máx}\{|\lambda| : \lambda \in \lambda(X)\}. \tag{1.11.3}$$

El númer $\rho(X)$ se denomina **radio espectral** de X.

Propiedad 1.11.4. *Dada $A \in \mathbb{R}^{n\times n}$:*

$$\rho(A) = \inf_{\|\cdot\|} \|A\|$$

donde el ínfimo es calculado considerando el conjunto de todas las normas matriciales (inducidas).

Dada $A \in \mathbb{R}^{n\times n}$ y una separación $A = M - N$ arbitraria. Consideremos la familia general de métodos iterativos determinados por la relación de recurrencia:

$$Mx_{k+1} = Nx_k + b \tag{1.11.4}$$

Cuando $M \in \mathbb{GL}_n$, la convergencia de una sucesión determinada por (1.11.4) está determinada por $\lambda(M^{-1}N)$, el valor de $\rho(M^{-1}N)$ es crítico para la convergencia de la sucesión definida por (1.11.4).

Teorema 1.11.5. *Suponiendo que $b \in \mathbb{R}^n$ y $A = M - N \in \mathbb{GL}_n$. Si $M \in \mathbb{GL}_n$ y $\rho(M^{-1}N) < 1$, entonces la sucesión $\{x_k\}_{k\geq 0} \subset \mathbb{R}^n$ definida por* (1.11.4) *cumple: $x_k \to x = A^{-1}b$, para cualquier vector inicial $x_0 \in \mathbb{R}^n$.*

Demostración. Sea $e_k = x_k - x$ el error de la k-ésima iteración. Dado que $Ax = (M - N)x = b$, $\Longrightarrow$ $Mx = Nx + b$, $\Longrightarrow$

$$\begin{aligned} Me_{k+1} &= M(x_{k+1} - x) \\ &= Mx_{k+1} - Mx \\ &= Nx_k + b - (Nx + b) \\ &= N(x_k - x) = Ne_k \end{aligned}$$

$\Longrightarrow$

$$e_{k+1} = M^{-1}Ne_k = \left(M^{-1}N\right)^2 e_{k-1} = \cdots = \left(M^{-1}N\right)^{k+1} e_0 \tag{1.11.5}$$

Dado que $\rho(M^{-1}N) < 1$, por Propiedad 1.11.4 existe una norma matricial $\|\cdot\|$ tal que:

$$\|M^{-1}N\| < 1 \tag{1.11.6}$$

Por (1.11.5) y (1.11.6):

$$\|e_{k+1}\| = \| \left(M^{-1}N\right)^{k+1} e_0\| \leq \|M^{-1}N\|^{k+1}\|e_0\| \xrightarrow[k\to\infty]{} 0$$

$\Longrightarrow$

$$x_k - A^{-1}b = x_k - x = e_k \to 0.$$

Por tanto, $x_k \to A^{-1}b$. □

1.11.2. Métodos iterativos elementales

Dada una matriz $A = [a_{jk}] \in \mathbb{R}^{n\times n}$ consideremos las siguientes matrices definidas con base en los coeficientes de A:

$$D_A = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & a_{nn} \end{bmatrix}$$

$$L_A = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ -a_{21} & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ -a_{n1} & \cdots & -a_{nn-1} & 0 \end{bmatrix}$$

$$U_A = \begin{bmatrix} 0 & -a_{12} & \cdots & -a_{1n} \\ 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & -a_{n-1n} \\ 0 & \cdots & 0 & 0 \end{bmatrix}$$

Observación 1.11.6. Es posible observar que:

$$A = D_A - L_A - U_A$$

Dada $A \in \mathbb{GL}_n$, consideraremos algunos métodos iterativos elementales, y las condiciones en las que pueden aplicarse para resolver us sistema de ecuaciones de la forma (1.11.1).

Método de Richardson

Es el método determinado por la ecuación de recurrencia:

$$x_{k+1} = (I - A)x_k + b$$

Ejercicio para el lector 1.11.1. Determinar condiciones suficientes para que el método de Richardson produzca una sucesión $x_k \to A^{-1}b$.

Ejercicio para el lector 1.11.2. Escribir un programa Octave que implemente el método de Richardson.

Método de Jacobi

Es el método determinado por la ecuación de recurrencia:

$$D_A x_{k+1} = (L_A + U_A)x_k + b$$

Ejercicio para el lector 1.11.3. Determinar condiciones suficientes para que el método de Jacobi produzca una sucesión $x_k \to A^{-1}b$.

Ejercicio para el lector 1.11.4. Escribir un programa Octave que implemente el método de Jacobi.

Método de Gauss-Seidel

Es el método determinado por la ecuación de recurrencia:

$$(D_A - L_A)x_{k+1} = U_A x_k + b$$

Ejercicio para el lector 1.11.5. Determinar condiciones suficientes para que el método de Gauss-Seidel produzca una sucesión $x_k \to A^{-1}b$.

Ejercicio para el lector 1.11.6. Escribir un programa Octave que implemente el método de Gauss-Seidel.

Método SOR: Sobrerelajaciones sucesivas

Es el método determinado por la ecuación de recurrencia:

$$(D_A - \omega L_A)x_{k+1} = \omega(U_A x_k + b) + (1-\omega)Dx_k$$

Ejercicio para el lector 1.11.7. Determinar condiciones suficientes para que el método SOR produzca una sucesión $x_k \to A^{-1}b$.

Ejercicio para el lector 1.11.8. Escribir un programa Octave que implemente el método de SOR.

1.12. Prinicipios de Métodos de Gradiente Conjugado: Métodos del descenso más empinado

Consideremos una matriz $A \in \mathbb{R}^{n\times n}$ arbitraria y un vector arbitrario $b \in \mathbb{R}^n$, y consideremos el funcional cuadrático:

$$\phi_{A,b}(x) = \frac{1}{2}x^\top Ax - x^\top b$$

Propiedad. Si $A \in \mathbb{R}^{n\times n}$ es SPD, resolver el sistema $Ax = b$ equivale a resolver el siguiente problema de programación cuadrática.

$$\min_{x\in\mathbb{R}^n} \phi_{A,b}(x) \tag{1.12.1}$$

La solución de los problemas cuadráticos de la forma (2.2.1) correspondientes a un sistema $Ax = b$ con matriz de coeficientes A SPD, pueden resolverse implementando métodos de gradiente conjugado que producen una secuencia $\{x_k\}_{k\geq 0} \subset \mathbb{R}^n$ tal que $x_k \to x = A^{-1}b$ y donde cada elemento x_k de la sucesión está dado por la expresión:

$$x_k = x_{k-1} + \alpha_{k-1}p_{k-1},\ k \geq 1$$

donde $\alpha_k \in \mathbb{R}$ y $p_k \in \mathbb{R}^n$ se calculan de acuerdo a criterios de optimzación que serán estudiados en detalle más adelante, y $x_0 \in \mathbb{R}^n$ es un elemento que se utiliza para inicializar el esquema iterativo.

Ejercicio para el lector 1.12.1. Demostrar que:

$$-\nabla\phi_{A,b}(x) = b - Ax$$

Observación 1.12.1. En un punto específico x_c, $\phi_{A,b}(x_c)$ decrece más rápidamente en la dirección del gradiente:

$$-\nabla\phi_{A,b}(x_c) = b - Ax_c$$

Sea r_c el resíduo $r_c = b - Ax_c$ de x_c. Si $r_c \neq 0$, entonces existe $\alpha > 0$ tal que $\phi_{A,b}(x_c + \alpha r_c) < \phi_{A,b}(x_c)$. En el método del descenso más empinado (con línea de búsqueda exacta) definimos:

$$\alpha = \frac{r_c^\top r_c}{r_c^\top A r_c}$$

con el fin de minimizar la expresión:

$$\phi_{A,b}(x_c + \alpha r_c) = \phi_{A,b}(x_c) - \alpha r_c^\top r_c + \frac{1}{2}\alpha^2 r_c^\top A r_c$$

Ejercicio para el lector 1.12.2. Verificar que $\alpha = \frac{r_c^\top r_c}{r_c^\top A r_c}$ minimiza la fórmula $\phi_{A,b}(x_c + \alpha r_c)$ definida por la expresión:

$$\phi_{A,b}(x_c + \alpha r_c) = \phi_{A,b}(x_c) - \alpha r_c^\top r_c + \frac{1}{2}\alpha^2 r_c^\top A r_c$$

Las considereciones anteriores permiten derivar el siguiente algoritmo prototípico para el método del descenso más empinado:

x_0: vector de inicialización

$r_0 \leftarrow b - Ax_0$

$k \leftarrow 0$

mientras $r_k \neq 0$

$k \leftarrow k + 1$

$\alpha_k \leftarrow (r_k^\top r_k)/(r_k^\top A r_k)$

$x_k \leftarrow x_{k-1} + \alpha_k r_{k-1}$

$r_k \leftarrow b - Ax_k$

fin

Ejercicio para el lector 1.12.3. Investigar sobre métodos computacionales de implementación de gradientes conjugados que permitan mejorar las características de convergencia de algoritmos de descenso más empinado como el anterior.

Ejercicio para el lector 1.12.4. Escribir un programa Octave que implemente el algoritmo elemental de descenso más empinado descrito anteriormente.

Ejercicio para el lector 1.12.5. Escribir un programa Octave que implemente el algoritmo de gradiente conjugado encontrado a través de su investigación.

1.13. Método Iterativos de Cálculo de Eigenvalores y Eigenvectores

1.14. Preliminares

Notación 1.14.1. Utilizando la notación usual de álgebra, geometría y trigonometría elementales, en lo secesivo se escribirá $\mathbb{C}$ para denotar el conjunto de números complejos.

Definición 1.14.2. Dada una matriz $A \in \mathbb{C}^{n\times n}$ (A es una matriz de $n\times n$ con coeficientes complejos) y dado un eigenvalor $a \in \lambda(A)$, se denomina eigenvector de A correspondiente a a, al vector $x \in \mathbb{C}^n$ (un vector con coeficientes complejos) que cumple la condición.

$$Ax = ax \tag{1.14.1}$$

Cada par de la forma (a, x) determinado por la ecuación (1.14.1), se denomina un **eigenpar** de A.

Definición 1.14.3. Dada una matriz $X = [x_{jk}] \in \mathbb{C}^{m\times n}$, se denota por $X^* \in \mathbb{C}^{n\times m}$ la matriz transpuesta conjugada de X, es decir, $X^* = \overline{X}^\top = [\overline{x}_{kj}] \in \mathbb{C}^{n\times m}$.

Ejemplo 1. Considerando la matriz:

$$A = \begin{bmatrix} 1+2i & 3 & -i \\ 3i & 0 & -2 \\ -2+i & -i & -1 \\ -1 & 1-3i & 5i \end{bmatrix}$$

al calcular A^* se obtiene la siguiente matriz.

$$A^* = \begin{bmatrix} 1-2i & -3i & -2-i & -1 \\ 3 & 0 & i & 1+3i \\ i & -2 & -1 & -5i \end{bmatrix}$$

Es posible verificar esta operación utilizando la siguientes secuencias de comandos en Octave.

```
>> A=[1+2*i,3,-i;3*i,0,-2;-2+i,-i,-1;-1,1-3*i,5*i]
A =

   1 + 2i   3 + 0i  -0 - 1i
   0 + 3i   0 + 0i  -2 + 0i
  -2 + 1i  -0 - 1i  -1 + 0i
  -1 + 0i   1 - 3i   0 + 5i

>> A'
ans =

   1 - 2i   0 - 3i  -2 - 1i  -1 - 0i
   3 - 0i   0 - 0i  -0 + 1i   1 + 3i
  -0 + 1i  -2 - 0i  -1 - 0i   0 - 5i
```

Notación 1.14.4. En lo sucesivo, dado un número complejo $z = z_1 + iz_2 \in \mathbb{C}$, la expresión $|z|$ se utilizará para representar el módulo de z, definido por la expresión $|z| = (\overline{z}z)^{1/2} = (z_1^2 + z_2^2)^{1/2}$

Definición 1.14.5. Dados $x, y \in \mathbb{C}^n$, el producto escalar $x \cdot y$ se define como el número complejo determinado por la siguiente expresión.

$$x \cdot y = y^* x = \sum_{j=1}^{n} x_j \overline{y}_j$$

Definición 1.14.6. Dados $x \in \mathbb{C}^n$, el producto escalar $\cdot$ induce la norma $\| \cdot \|_2 : \mathbb{C}^n \to \mathbb{R}$ definida por la expresión.

$$\|x\|_2 = (x \cdot x)^{1/2} = (x^* x)^{1/2} = \left(\sum_{j=1}^{n} |x_j|^2 \right)^{1/2}$$

Definición 1.14.7. La norma $\| \cdot \|_2$ en $\mathbb{C}^n$ induce la distancia $d_2 : \mathbb{C}^n \times \mathbb{C}^n \to \mathbb{R}$ definida por la expresión.

$$d_2(x, y) = \|x - y\|_2$$

Notación 1.14.8. Dada una sucesión $\{x_k\}_{k \geq 0} \subset \mathbb{C}^n$, escribimos $x_k \xrightarrow[k \to \infty]{} x$ o $x_k \longrightarrow x$ (donde $x \in \mathbb{C}^n$), si $\lim_{k \to \infty} d_2(x_n, x) = 0$.

Propiedad 1.14.9. *(Versión compleja de la desigualdad de Cauchy-Schwarz) Dados $x, y \in \mathbb{C}^n$, se cumple la siguiente desigualdad.*

$$|x \cdot y| \leq \|x\|_2 \|y\|_2$$

Definición 1.14.10. Se dice que una función $\phi : \mathbb{C}^n \to \mathbb{C}$ es una funcional lineal si para cualesquiera $x, y \in \mathbb{C}^n$ y cualquier $\alpha \in \mathbb{C}$, se cumple la siguiente condición.

$$\phi(x + \alpha y) = \phi(x) + \alpha \phi(y)$$

Observación 1.14.11. Es importante observar que la definición 1.14.10 no contradice la definición de funcional lineal estudiada en un curso de vectores y matrices.

Ejercicio para el lector 1.14.1. Demostrar en detalle que la definición 1.14.10 es equivalente a la definición usual de funcional lineal, en el sentido del curso de vectores y matrices.

Propiedad 1.14.12. *Para cada funcional lineal $\phi : \mathbb{C}^n \to \mathbb{C}$, existe un único $v_\phi \in \mathbb{C}^n$ tal que:*

$$\phi(x) = x \cdot v_\phi$$

Definición 1.14.13. Dado $f : \mathbb{C}^n \to \mathbb{C}$ decimos que f es continua en un punto $x \in \mathbb{C}^n$, si para cada $\{x_n\}_{n \geq 0} \subset \mathbb{C}^n$ tal que $x_n \xrightarrow[n \to \infty]{} x$, se cumple que $f(x_n) \xrightarrow[n \to \infty]{} f(x)$. Se dice que $f : \mathbb{C}^n \to \mathbb{C}$ es continua en $\mathbb{C}^n$ si f es continua en cada punto $x \in \mathbb{C}^n$.

Propiedad 1.14.14. *Toda funcional lineal $\phi : \mathbb{C}^n \to \mathbb{C}$ es continua en $\mathbb{C}^n$.*

Ejercicio para el lector 1.14.2. Demostrar en detalle la propiedad 1.14.14.

1.14.1. Método de Potencia

Iniciamos esta sección considerando las siguientes suposiciones.

Suposición 1.14.15. Se considera una familia general de matrices tal que cada matriz $A \in \mathbb{C}^{n \times n}$ en la familia tiene un eigenpar dominante computable con el método de potencia, en el sentido determinado por las siguientes suposiciones:

1. Existe $a \in \lambda(A)$ tal que $|\alpha| < |a|$, para cada $\alpha \in \lambda(A)$.

2. Existe un conjunto linealmente independiente de eigenvectores de A.

Suposición 1.14.16. Sin perder generalidad se considera una representación del conjunto $\lambda(A) = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ tal que $\lambda_1 = a$ y:

$$|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$$

Se considera una conjunto $\{x_1, \ldots, x_n\} \subset \mathbb{C}^n$ de eigenvectores linealmente independiente de A, tales que:

$$Ax_j = \lambda_j x_j$$

Por suposición 1.14.15, suposición 1.14.16 y propiedades elementales de vectores y matrices se cumple que $\{x_1, \ldots, x_n\}$ es una base de $\mathbb{C}^n$. Sea $x_0 \in \mathbb{C}^n$ un elemento cualquiera en $\mathbb{C}^n$ para el cual existen escalares $c_1(x_0), \ldots, c_n(x_0) \in \mathbb{C}$ tales que:

$$x_0 = c_1(x_0)x_1 + \cdots + c_n(x_0)x_n \tag{1.14.2}$$

y $c_1(x_0) \neq 0$.

Es posible formar una sucesión $\{x_k\}_{k\geq 0}$ definida por las expresiones:

$$\begin{aligned} x_1 &= Ax_0 \\ x_2 &= Ax_1 \\ &\cdots \\ x_k &= Ax_{k-1} \end{aligned}$$

Se cumplirá entonces que:

$$x_k = A^k x_0 \tag{1.14.3}$$

Por (1.14.3) se cumple entonces lo siguiente.

$$\begin{aligned} x_k &= A^k x_0 \\ &= c_1(x_0)A^k x_1 + c_2(x_0)A^k x_2 + \cdots + c_n(x_0)A^k x_n \\ &= c_1(x_0)\lambda_1^k x_1 + c_2(x_0)\lambda_2^k x_2 + \cdots + c_n(x_0)\lambda_n^k x_n \\ &= \lambda_1^k \left(c_1(x_0)x_1 + c_2(x_0)\left(\frac{\lambda_2}{\lambda_1}\right)^k x_2 + \cdots + c_n(x_0)\left(\frac{\lambda_n}{\lambda_1}\right)^k x_n \right) \end{aligned} \tag{1.14.4}$$

Dado un funcional lineal $\phi : \mathbb{C}^n \to \mathbb{C}$ que cumple la condición:

$$\phi(x_1) \neq 0 \tag{1.14.5}$$

Si se consideran la sustitución:

$$y_k = c_2(x_0)\left(\frac{\lambda_2}{\lambda_1}\right)^k x_2 + \cdots + c_n(x_0)\left(\frac{\lambda_n}{\lambda_1}\right)^k x_n \tag{1.14.6}$$

Dado que $|\lambda_j| \leq |\lambda_1|$ para cada $2 \leq j \leq n$, $|\lambda_j/\lambda_1|^k \xrightarrow[k\to\infty]{} 0 \Longrightarrow (\lambda_j/\lambda_1)^k \to 0$, para cada $2 \leq j \leq n$. $\Longrightarrow$

$$y_k \xrightarrow[k\to\infty]{} 0 \tag{1.14.7}$$

Por definición 1.14.10 es posible obtener la siguiente expresión.

$$\phi(x_k) = \phi\left(\lambda_1^k \left(c_1(x_0)x_1 + y_k\right)\right) = \lambda_1^k \left(c_1(x_0)\phi(x_1) + \phi(y_k)\right) \tag{1.14.8}$$

Por propiedad 1.14.14 $\phi(y_k) \xrightarrow[k\to\infty]{} \phi(0) = 0 \Longrightarrow c_1(x_0)\phi(x_1) + \phi(y_{k+1}) \xrightarrow[k\to\infty]{} c_1(x_0)\phi(x_1) \Longrightarrow$

$$\frac{\phi(x_{k+1})}{\phi(x_k)} = \frac{\lambda_1^{k+1}\left(c_1(x_0)\phi(x_1) + \phi(y_{k+1})\right)}{\lambda_1^k\left(c_1(x_0)\phi(x_1) + \phi(y_k)\right)} = \lambda_1 \frac{c_1(x_0)\phi(x_1) + \phi(y_{k+1})}{c_1(x_0)\phi(x_1) + \phi(y_k)} \longrightarrow \lambda_1 \tag{1.14.9}$$

Las consideraciones anteriores permiten derivar el siguiente algoritmo prototípico para el método de potencia:

Algoritmo 1. **Método de potencia**

entradas A, x, N

salidas k, x, r

Paso 0: Definir una regla de cómputo de un funcional lineal ϕ que cumpla (1.14.5)

para $k = 1, \ldots, N$ **hacer**

$y \leftarrow Ax$

$r \leftarrow \phi(y)/\phi(x)$

$x \leftarrow y/\|y\|_2$

fin

Ejercicio para el lector 1.14.3. Escribir un programa Octave que implemente al algoritmo prototípico del método potencia para calcular un eigenpar dominante (a, x) de una matriz A que cumple con las condiciones de convergencia del método de potencia, y que además permite detener el cómputo de elementos de la sucesión $\{x_k\}_{k\geq 0}$ determinada por el método iterativo, una vez que se alcanza un error absoluto $\|Ax_k - a_k x_k\|_2 \leq \varepsilon$, donde $\varepsilon > 0$ es un valor de tolerancia determinado por el usuario del programa.

Ejercicio para el lector 1.14.4. Escribir un programa Octave que genere una matriz $A \in \mathbb{R}^{3000\times 3000}$ tal que el método de potencia aplicado a A aproxima un eigenpar dominante (λ_1, x_1) de A para una toleracia $\leq 1 \times 10^{-10}$ (de preferencia en un número de iteraciones $N \ll 3000$, es decir, el valor de N es considerablemente menor que 3000).

1.14.2. Método de Potencia Inversa

Iniciamos esta sección revisando la siguiente propiedad.

Propiedad 1.14.17. *Si $a \in \lambda(A)$ y si $A \in \mathbb{GL}_n$, entonces $a^{-1} \in \lambda(A^{-1})$.*

Ejercicio para el lector 1.14.5. Aplicando propiedades estudiadas en el curso de vectores y matrices, demostrar en detalle la propiedad 1.14.17.

La propiedad 1.14.17 propone una vía para calcular el valor propio más pequeño de una matriz $A \in \mathbb{C}^{n\times n}$.

Suposición 1.14.18. Suponiendo que $A \in \mathbb{C}^{n\times n}$ es una matriz, tal que, existe una representación del conjunto $\lambda(A) = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ tal que:

$$|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_{n-1}| > |\lambda_n| > 0$$

Observación 1.14.19. Es claro que si una matriz $A \in \mathbb{C}^{n\times n}$ cumple la suposición 1.14.18, entonces $A \in \mathbb{GL}_n$, dado que $0 \notin \lambda(A)$.

Ejercicio para el lector 1.14.6. Verificar en detalle la observación 1.14.19.

Por la propiedad 1.14.17, para cada matriz $A \in \mathbb{C}^{n\times n}$ se cumple que $\lambda_j^{-1} \in \lambda(A^{-1})$ para cada $\lambda_j \in \lambda(A)$, además:

$$|\lambda_n^{-1}| > |\lambda_{n-1}^{-1}| \geq \cdots \geq |\lambda_2^{-1}| \geq |\lambda_1^{-1}| > 0 \tag{1.14.10}$$

Como consecuencia de (1.14.10) es posible aplicar el método de potencia a A^{-1} para calcular el eigenpar (λ_n^{-1}, x_n) de A^{-1}.

Observación 1.14.20. Es importante observar que no es una práctica computacional eficiente calcular la secuencia $\{x_k\}_{k\geq 0}$ correspondiente al método de la potencia aplicado a A^{-1}, utilizando las relaciones de recurrencia:

$$x_{k+1} = A^{-1}x_k, \;\; k \geq 0$$

en lugar de este cómputo, es más eficiente calcular cada x_{k+1} resolviendo los siguientes sistemas de ecuaciones lineales.

$$Ax_{k+1} = x_k, \;\; k \geq 0$$

El método determinado por la observación 1.14.20 se denomina **método de potencia inversa**.

Ejercicio para el lector 1.14.7. Escribir un programa Octave que modificando al algoritmo prototípico del método potencia con base en la observación 1.14.20, para calcular un eigenpar (λ_n, x_n) de una matriz A que cumple con las condiciones de convergencia del método de potencia y con la suposición 1.14.18, y que además permite detener el cómputo de elementos de la sucesión $\{x_k\}_{k\geq 0}$ determinada por el método iterativo, una vez que se alcanza un error absoluto $\|A^{-1}x_k - a_k^{-1}x_k\|_2 \leq \varepsilon$, donde $\varepsilon > 0$ es un valor de tolerancia determinado por el usuario del programa.

Ejercicio para el lector 1.14.8. Escribir un programa Octave que genere una matriz $A \in \mathbb{R}^{3000\times 3000}$ tal que el método de potencia aplicado a A^{-1} converge a un eigenpar dominante (λ_n^{-1}, x_n) de A (de preferencia en un número de iteraciones $N \ll 3000$, es decir, el valor de N es considerablemente menor que 3000).

1.14.3. Resumen de esquemas iterativos elementales de cómputo de eigenpares

Método de potencia

- Ecuación de recurrencia: $x_{k+1} = Ax_k$.
- Resultado: eigenpar (a, x) correspondiente al eiganvalor a más grande en magnitud.

Método de potencia inversa

- Ecuación de recurrencia: $Ax_{k+1} = x_k$.
- Resultado: eigenpar (a, x) correspondiente al eiganvalor a más pequeño en magnitud.

Método de potencia con traslación

- Ecuación de recurrencia: $x_{k+1} = (A - \mu I)x_k$.
- Resultado: eigenpar (a, x) correspondiente al eiganvalor a más lejano en distancia al valor de μ.

Método de potencia inversa con traslación

- Ecuación de recurrencia: $(A - \mu I)x_{k+1} = x_k$.
- Resultado: eigenpar (a, x) correspondiente al eiganvalor a más cercano en distancia al valor de μ.

1.15. Teoremas de Schur y Gershgorin

Definición 1.15.1. Dos matrices $A, B \in \mathbb{C}^{n\times n}$ se dicen **similares**, si existe $P \in \mathbb{GL}_n$ tal que $B = PAP^{-1}$.

Propiedad 1.15.2. *Si las matrices $A, B \in \mathbb{C}^{n\times n}$, entonces $\lambda(A) = \lambda(B)$, es decir, A y B tienen los mismos eigenvalores.*

Ejercicio para el lector 1.15.1. Verificar en detalle la propiedad 1.15.2.

Definición 1.15.3. Una matriz $U \in \mathbb{C}^{n\times n}$ se denomina **unitaria** si $U^*U = UU^* = I$. El conjunto de matrices unitarias en $\mathbb{C}^{n\times n}$ será denotado en este curso por $\mathbb{U}(n)$

Ejemplo 2. La matriz

$$U = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & i \\ i & 1 \end{bmatrix}$$

es unitaria dado que se cumple que $U^*U = UU^* = I$.

Observación 1.15.4. Para cada matriz $U \in \mathbb{U}(n)$, se cumple que $U^{-1} = U^*$. Además, es claro que $\mathbb{U}(n) \subset \mathbb{GL}_n$, es decir, toda matriz unitaria es invertible.

Propiedad 1.15.5. *(Teorema de Schur) Toda matriz $A \in \mathbb{C}^{n\times n}$ es unitariamente similar a una matriz $T_A \in \mathbb{S}(n)$, es decir, existe $U \in \mathbb{U}(n)$ tal que $T_A = U^*AU \in \mathbb{S}(n)$.*

Definición 1.15.6. Una matriz $A \in \mathbb{C}^{n\times n}$ se dice Hermitiana si $A^* = A$.

Ejemplo 3. La matriz $H \in \mathbb{C}^{3\times 3}$ definida por la expresión,

$$H = \begin{bmatrix} 2 & 3i & -2+i \\ -3i & -1 & 3 \\ -2-i & 3 & 1 \end{bmatrix}$$

es Hermitiana dado que se cumple la siguiente condición.

$$H^* = \begin{bmatrix} 2 & 3i & -2+i \\ -3i & -1 & 3 \\ -2-i & 3 & 1 \end{bmatrix} = H$$

Propiedad 1.15.7. *Toda matriz Hermitiana $A \in \mathbb{C}^{n\times n}$ es similar a una matriz $D_A \in \mathbb{D}(n)$.*

Ejercicio para el lector 1.15.2. Demostrar en detalle la propiedad 1.15.7.

1.15.1. Localización de eigenvalores

Uno de los teoremas fundamentales de localización de eigenvalores es el teorema de Gershgorin, cuyo formulación se presenta a continuación.

Propiedad 1.15.8. *(Teorema de Gershgorin) Dada $A \in \mathbb{C}^{n\times n}$ se cumple la siguiente condición:*

$$\lambda(A) \subset \bigcup_{j=1}^{n} D_j$$

donde cada **disco** $D_j \subset \mathbb{C}$ *está definido por las siguientes expresiones.*

$$D_j = \left\{ z \in \mathbb{C} : |z - a_{jj}| \leq \sum_{k=1, k\neq j}^{n} |a_{jk}| \right\}$$

Ejemplo 4. Considerando la matriz $A \in \mathbb{C}^{4\times 4}$ determinada por la expresión.

$$A = \begin{bmatrix} 1+2i & 3 & -i & -2i \\ 3i & 0 & -2 & 3i \\ -2+i & -i & -1 & i \\ -1 & 1-3i & 5i & 1 \end{bmatrix}$$

Es posible verificar la propiedad 1.15.8, bosquejando gráficamente la unión de discos determinados por la propiedad 1.15.8 y graficando en el plano el conjunto $\lambda(A)$ de eigenvalores de A, esto puede hacerse utilizando las siguientes secuencias de comandos en Octave.

```
>> A=[1+2*i,3,-i,-2*i;3*i,0,-2,3*i;-2+i,-i,-1,i;-1,1-3*i,5*i,1];
>> D=diag(A);
>> R=sum(abs(A-diag(D)),2);
>> t=0:1/60:1;
>> C=exp(2*pi*i*t);
>> plot(eig(A),'b.','markersize',20);
>> hold on;
>> for k=1:size(A,1),plot(D(k)+R(k)*C,'r','markersize',12);end
>> hold off;
>> axis equal
>> axis tight
>> grid on
>> legend ('\lambda(A)','\partial D_1','\partial D_2',...
> '\partial D_3','\partial D_4')
```

La salida gráfica producida por la secuencia de comandos previamente implementados en Octave se muestra en la fig. 1.1.

Ejercicio para el lector 1.15.3. Considerando la matriz $A(a, b, c, d) \in \mathbb{C}^{n\times n}$:

$$A = \begin{bmatrix} b & c & 0 & 0 & 0 & d \\ a & b & c & 0 & 0 & 0 \\ 0 & a & b & c & 0 & 0 \\ 0 & 0 & a & b & c & 0 \\ 0 & 0 & 0 & a & b & c \\ d & 0 & 0 & 0 & a & b \end{bmatrix}$$

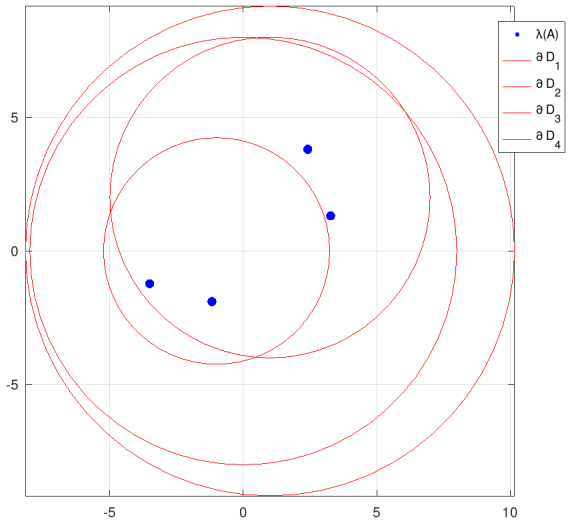


Figura 1.1: Ejemplo gráfico de cómputo de la región de localización de $\lambda(A)$ determinada por la propiedad 1.15.8. Los círculos rojos representan las fronteras ∂D_j de los discos D_j definidos como parte de la propiedad 1.15.8, y los puntos azules representan el conjunto $\lambda(A) \subset \mathbb{C}$ de los eigenvalores de A. A partir de este bosquejo, es posible visualizar que el espectro $\lambda(A)$ de la matriz A queda contenido en la unión $D_1 \cup D_2 \cup D_3 \cup D_4$ de los discos determinados por la propiedad 1.15.8.

aplicar las propiedad 1.15.5 y propiedad 1.15.8 para estimar subconjuntos propios $S(a,b,c,d)$ del conjunto $\mathbb{C}$ de números complejos, tales que $\lambda(A(a,b,c,d)) \subset S(a,b,c,d)$ para las siguientes restricciones de los números a, b, c, d.

1. $a, b, c, d \in \mathbb{C}$.
2. $a, b, c, d \in \mathbb{R}$.
3. $a, b, c, d \in \mathbb{R}$; $a = c$.
4. $a, c \in \mathbb{C}$; $b, d \in \mathbb{R}$; $a = \overline{c}$.

1.16. Factorizaciones Ortogonales en $\mathbb{C}^{n\times n}$

1.17. Prinicipios de Despomposiciones ortogonales y Descomposición en valores singulares

1.17.1. Descomposición QR

Iniciamos esta sección considerando que la siguiente versión del teorema de ortogonalizacón de Gram-Schmidt que permite etablecer lo siguiente.

Propiedad 1.17.1. *(Teorema de Gram-Schmidt) dados $x_1, \ldots, x_m \in \mathbb{R}^n$ linealmente independientes, existen $q_1, \ldots, q_m \in \mathbb{R}^n$ ortonormales tales que:*

$$\text{gen}\,\{x_1, \ldots, x_m\} = \text{gen}\,\{q_1, \ldots, q_m\}$$

Por la propiedad 1.17.1 se cumple que si $X, Q \in \mathbb{R}^{n\times m}$ son las matrices definidas por las expresiones:

$$X = \begin{bmatrix} | & & | \\ x_1 & \cdots & x_m \\ | & & | \end{bmatrix}$$

$$Q = \begin{bmatrix} | & & | \\ q_1 & \cdots & q_m \\ | & & | \end{bmatrix}$$

Se cumple que:

$$Q^\top Q = I_m$$
$$Q^\top X = R$$
$$QR = X$$

donde I_m representa la matriz identidad en $\mathbb{R}^{m\times m}$ y $R \in \mathbb{S}(m)$. La descomposición ortogonal $X = QR$ recibe el nombre de descomposición QR de X.

Notación 1.17.2. Una matriz $U \in \mathbb{R}^{n\times m}$ que cumple $U^\top U = I_m$ se denominará **matriz ortogonal** en este curso.

Notación 1.17.3. El conjunto de matrices ortogonales en $\mathbb{R}^{n\times m}$, será denotado por $\mathbb{O}(n, m)$, en el caso de las matrices ortogonales en $\mathbb{R}^{n\times n}$, en algunas ocasiones se escribirá $\mathbb{O}(n)$ en lugar de $\mathbb{O}(n, n)$.

Ejercicio para el lector 1.17.1. Aplicar el teorema de ortogonalización de Gram-Schmidt para verificar que toda matriz $X \in \mathbb{R}^{n\times m}$ con columnas linealmente independientes tiene una factorización QR.

Ejercicio para el lector 1.17.2. Es posible que toda matriz $X \in \mathbb{R}^{n\times n}$ tenga una factorización QR?

Matrices de Permutación

Un tipo especial de matrices ortogonales en $\mathbb{R}^{n\times n}$, son las matrices de permutación, las cuales se obtienen a partir de la matriz identidad $I \in \mathbb{R}^{n\times n}$ permutando sus columnas (o renglones).

Ejemplos. En $\mathbb{R}^{4\times 4}$ las siguientes matrices son ejemplos de matrices de permutación:

$$Q_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \; Q_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \; Q_3 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \; Q_4 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Ejercicio para el lector 1.17.3. Calcular el número de matrices de permutación en $\mathbb{R}^{n\times n}$.

1.17.2. Descomposición en valores singulares

En este sección consideraremos por primera vez uno de los teoremas fundamentales del análisis matricial numérico.

Propiedad 1.17.4. *(Teorema fundamental de descomposición en valores singulares* SVD*) Si $A \in \mathbb{R}^{m\times n}$, entonces existen matrices ortogonales $U \in \mathbb{R}^{m\times m}$ y $V \in \mathbb{R}^{n\times n}$ tales que:*

$$U^\top AV = \Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_p) \in \mathbb{R}^{m\times n}, \;\; p = \min\{m, n\}$$

donde $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p \geq 0$. *Donde* $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_p) \in \mathbb{R}^{m\times n}$ *es una matriz en* $\mathbb{R}^{m\times n}$ *determinada por la expresión:*

$$\Sigma_{jk} = \begin{cases} \sigma_j, & j = k \\ 0, & j \neq k \end{cases}, 1 \leq j \leq p$$

1.18. Matrices Elementales de Householder

En esta sección se presenta la idea original de A. S. Householder documentada en [12], para el cálculo de factorización ortogonal por matrices elementales de Householder.

Definición 1.18.1. Dados $u, v \in \mathbb{C}^n$ y dado $\sigma \in \mathbb{C}$, de denomina matriz elemental, la matriz $E(u, v; \sigma)$ de la forma:

$$E(u, v; \sigma) = I - \sigma uv^*$$

Definición 1.18.2. Una matriz $A \in \mathbb{C}^{n\times n}$ se denomina una **involución** o **simetría** si $A^2 = AA = I$.

La observación de Householder

A. S. Householder realizó el siguiente análisis.

Observación 1.18.3. (Observación de Householder) Dados $u, v \in \mathbb{C}^n$ tales que:

$$\begin{cases} \|u\|_2 = \|v\|_2 \\ u^*v = v^*u \end{cases} \tag{1.18.1}$$

se cumple que existe $w \in \mathbb{C}^n$ tal que la matriz $H(w) = E(w, w; 2) \in \mathbb{C}^{n\times n}$ cumple las siguientes restricciones.

$$\begin{cases} H(w)^* = H(w) = H(w)^{-1} \\ H(w)u = v \end{cases} \tag{1.18.2}$$

Householder también observó que cuando $u - v \neq 0$, el vector $w \in \mathbb{C}^n$ puede calcularse utilizando la siguiente fórmula.

$$w = \frac{1}{\|u - v\|_2}(u - v) \tag{1.18.3}$$

Definición 1.18.4. La matriz $H(w) = E(w, w; 2)$ de denomina matriz elemental de Householder correspondiente de dos vectores $u, v \in \mathbb{C}^n$ que cumplen la condición (1.18.1).

Ejercicio para el lector 1.18.1. Dados dos vectores $u, v \in \mathbb{C}^n$ que cumplen (1.18.1), demostrar o refutar que siempre existe una simetría $H \in \mathbb{C}^{n\times n}$ tal que $Hu = v$.

Observación 1.18.5. Dadas dos matrices $A, B \in \mathbb{U}(n)$ es posible verificar que $AB \in \mathbb{U}(n)$ y que $BA \in \mathbb{U}(n)$.

Ejercicio para el lector 1.18.2. Verificar la observación 1.18.5.

Ejercicio para el lector 1.18.3. Aplicando matrices elementales de Householder y la 1.18.5, demostrar que para cada matriz $X \in \mathbb{R}^{n\times m}$ tal que las columnas de X son linealmente independientes, existe $H \in \mathbb{U}(n)$ tal que la matriz $Y = [y_{jk}] = HX$ cumple la restricción: $y_{jk} = 0$, cuando $k < j \leq n$, para $k = 1, \ldots, m$.

Ejercicio para el lector 1.18.4. Desarrollar un programa Octave que calcule para cada matriz $X \in \mathbb{R}^{n\times m}$ cuyas columnas son linealmente independientes, la factorización $Y = HX$ descrita en el ejercicio para el lector 1.18.3.

Ejemplo 5. (Aplicación eficience de matrices elementales de Householder) Es posible aplicar el programa `House.m`, disponible en el repositorio de archivos de la clase, para calcular las componentes de un matriz elemental $H(w)$ correspondiente a dos vectores $u, v \in \mathbb{C}^n$ que cumplen (1.18.1). Una posible implementación del programa `House.m` para resolver los 1.18.3 y 1.18.4 se presenta en el programa Octave `HouseFactor.m` cuyo código se muestra a conitnuación.

```
function [H,W,R]=HouseFactor(X)
R=X;
[m,n]=size(R);
H=eye(m);
for k=1:min([n m-1])
  T=triu(R);
  p=norm(R(k:m,k))/norm(T(k:m,k));
  [b,w]=House(R(k:m,k),p*T(k:m,k));
    W(:,k)=sqrt(b)*[zeros(k-1,1);w];
  H=H-W(:,k)*(W(:,k)'*H);
  R(k:m,:)=R(k:m,:)-b*w*(w'*R(k:m,:));
end
```

Es posible aplicar `HouseFactor.m` para calcular una factorización de Householder de una matriz $X \in \mathbb{R}^{5\times 3}$ generada al azar, utilizando la siguiente secuencia de comandos.

```
X=ceil(10*randn(5,3))
X =

   11    4   -4
   -1   -9   10
    6   10  -16
   -8   -1    8
   -3   18    6

>> [H,W,R]=HouseFactor(X);
```

Ahora es posible verificar que las condiciones de la factorización de Householder se cumplen (aproximadamente debido a los efectos de la aritmética finita).

```
>> R
R =

   15.19868    4.40828  -15.26448
   -0.00000  -22.41801    4.40315
   -0.00000    0.00000  -14.81917
   -0.00000    0.00000    0.00000
    0.00000    0.00000    0.00000

>> norm(H*X-R)
```

```
ans =    9.7025e-15
>> Q=H(1:3,:);
>> norm(Q'*Q*X-X)
ans =    2.3387e-14
>> Q'*Q*X
ans =

   11.00000    4.00000   -4.00000
   -1.00000   -9.00000   10.00000
    6.00000   10.00000  -16.00000
   -8.00000   -1.00000    8.00000
   -3.00000   18.00000    6.00000
```

Observación 1.18.6. Es importante observar que si bien el programa `HouseFactor.m` desarrollado por el autor de estas lecturas, resuelve los ejercicio para el lector 1.18.3 y ejercicio para el lector 1.18.4, el autor ha escrito este programa para ser didácticamente simple y fácil de leer, no obstante, la eficiencia del programa puede mejorarse.

Ejercicio para el lector 1.18.5. Desarrollar una versión más eficiente del programa `HouseFactor.m` (**presentando evidencia documental o un argumento teórico riguroso que demuestre la mejora en eficiencia**).

Observación 1.18.7. El ejemplo 5 permite, en particular, observar una relación entre la aplicación del algoritmo de Gram-Schmidt para calcular una descomposición QR de una matriz $X \in \mathbb{R}^{m\times n}$ con columnas linealmente independientes.

Ejercicio para el lector 1.18.6. Aplicar `HouseFactor.m` o una versión más eficiente desarrollada por usted, para desarrollar un programa Octave que permita calcular la descomposición QR de una matriz $X \in \mathbb{R}^{m\times n}$ con columnas linealmente independientes.

1.19. Mini Proyecto de Aplicación: Ajuste por Mínimos Cuadrados

Considerando el problema determinado por el ajuste de una colección de datos $\mathcal{D} = \{(x_j, y_j)\}_{j=1}^{N} \subset \mathbb{R}^2$ a través de una curva determinada por una expresión $y = f(x)$ donde $f(x) = a_0 + a_1x + a_2x^2 + \cdots a_nx^n$.

Considerando además la matriz $X \in \mathbb{R}^{N\times(n+1)}$ definida por la expresión.

$$X = \begin{bmatrix} x_1^n & x_1^{n-1} & \cdots & x_1^2 & x_1 & 1 \\ x_2^n & x_2^{n-1} & \cdots & x_2^2 & x_2 & 1 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ x_N^n & x_N^{n-1} & \cdots & x_N^2 & x_N & 1 \end{bmatrix} \tag{1.19.1}$$

Para este proyecto se hace la siguiente suposición.

Suposición 1.19.1. Se supone que $N \geq n+1$, y que las columnas de la matriz $X \in \mathbb{R}^{N\times(n+1)}$ definida por (2.6.1) son linealmente independientes.

Notación 1.19.2. Sean $a \in \mathbb{R}^{n+1}$ y $y \in \mathbb{R}^n$ los vectores definidos por las expresiones.

$$a = \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_1 \\ a_0 \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N-1} \\ y_N \end{bmatrix}, \tag{1.19.2}$$

Observación 1.19.3. Es posible demostrar que el problema:

$$a = \operatorname{argmin}_{b \in \mathbb{R}^{n+1}} \frac{1}{2} \|Xb - y\|_2^2 \tag{1.19.3}$$

es equivalente a resolver el siguiente sistema.

$$X^\top X a = X^\top y \tag{1.19.4}$$

Ejercicio para el lector 1.19.1. Desarrollar un programa Octave que resuelva el problema (2.6.3), para las condiciones adecuadas de $X \in \mathbb{R}^{N \times (n+1)}$.

Ejercicio para el lector 1.19.2. Dado un número entero $n > 0$ (determinado por el usuario). Desarrollar un programa Octave que genere conjuntos de datos $\mathcal{D} = \{(x_j, y_j)\}_{j=1}^N \subset \mathbb{R}^2$ tales que la suposición 2.6.1 se cumple. Aplicar el programa desarrollado como parte del ejercicio para el lector 2.6.2 para calcular el polinomio que mejor ajusta los datos en $\mathcal{D}$ en el sentido de los mínimos cuadrados.

1.20. Proyectores

Definición 1.20.1. Una matriz $P \in \mathbb{C}^{n \times n}$ se denomina un **proyector** si $P^2 = P = P^*$.

Definición 1.20.2. Dada una matriz $X \in \mathbb{C}^{n \times m}$, se denomina el rango de X como el número entero determinado por el máximo número de columnas linealmente independientes de X, se denota por $\operatorname{rk}(X)$ el rango de X.

Definición 1.20.3. Dada una matriz $A \in \mathbb{C}^{n \times m}$, se define el **espacio imagen** de A como el conjunto $\{y = Ax : x \in \mathbb{C}^m\} \subseteq \mathbb{C}^n$.

Dada una matriz $F \in \mathrm{C}^{n \times m}$ cuyas columnas son linealmente idependientes, y dado $x \in \mathbb{C}^m$. El vector Fx es un elemento del espacio generado por la columnas de F. El vector Fx es una proyección ortogonal de un vector $y \in \mathbb{C}^n$ si se cumple la siguiente restricción.

$$F^*(y - Fx) = 0 \tag{1.20.1}$$

Por (1.20.1) se tiene que Fx es la proyección de un vector $z \in \mathbb{C}^n$ en el espacio imagen de F siempre que se cumple la siguiente restricción.

$$x = (F^*F)^{-1}F^*z \tag{1.20.2}$$

Observación 1.20.4. Es importante observar que, en efecto, $F^*F \in \mathbb{GL}_m$ como consecuencia de las propiedades de la descomposición QR de F, dado que las columnas de F son linealmente independientes.

Ejercicio para el lector 1.20.1. Verificar la observación 1.20.4.

Con base en la observación 1.20.4, si se define la matriz

$$P_F = F(F^*F)^{-1}F^* \tag{1.20.3}$$

se pueden realizar las siguientes observaciones.

Observación 1.20.5. Es posible observar que la matriz P_F definida por (1.20.3) cumple las siguientes restricciones.

$$\begin{aligned} P_F^2 &= P_F P_F \\ &= F(F^*F)^{-1}F^*F(F^*F)^{-1}F^* \\ &= F(F^*F)^{-1}F^* = P_F \end{aligned} \tag{1.20.4}$$

$$\begin{aligned} P_F^* &= (F(F^*F)^{-1}F^*)^* \\ &= F((F^*F)^{-1})^*F^* \\ &= F((F^*F)^*)^{-1}F^* \\ &= F(F^*F)^{-1}F^* = P_F \end{aligned} \tag{1.20.5}$$

Además, por (1.20.4) y (1.20.5) es posible observar que la matriz $I - P_F$ cumple las siguientes restricciones.

$$\begin{aligned} (I - P_F)^2 &= (I - P_F)(I - P_F) \\ &= I - P_F - P_F + P_F^2 \\ &= I - 2P_F + P_F = I - P_F \end{aligned} \tag{1.20.6}$$

$$(I - P_F)^* = I^* - P_F^* = I - P_F \tag{1.20.7}$$

$$(I - P_F)P_F = P_F - P_F^2 = P_F - P_F = \mathbf{0} \tag{1.20.8}$$

$$P_F(I - P_F) = P_F - P_F^2 = P_F - P_F = \mathbf{0} \tag{1.20.9}$$

Observación 1.20.6. Por la observación 1.20.5, si P_F es la matriz definida por la ecuación (1.20.3) para una matriz $F \in \mathbb{C}^{m\times n}$ cuyas columnas son linealmente independientes, entonces P_F y $I - P_F$ son proyectores, y además $P_F(I - P_F) = (I - P_F)P_F = \mathbf{0}$.

Definición 1.20.7. Dada una matriz $A \in \mathbb{C}^{n\times m}$ cuyas columnas son linealmente independientes, se denotará por P_A el proyector definido por la expresión $P_A = A(A^*A)^{-1}A^*$.

Definición 1.20.8. Dada una matriz $A = [a_{jk}] \in \mathbb{C}^{n\times n}$ se denomina traza de A el número $\mathrm{tr}(A)$ definido por la siguiente expresión.

$$\mathrm{tr}(A) = a_{11} + a_{22} + \cdots + a_{nn}$$

Ejercicio para el lector 1.20.2. Dado un proyector $P \in \mathbb{C}^{n\times n}$. Demostrar que P cumple con las siguientes condiciones.

- $\lambda(P) \subset \{0, 1\}$
- $\mathrm{tr}(P) = \mathrm{rk}(P)$
- $I - 2P \in \mathbb{U}(n)$
- $\lambda(I - 2P) \subset \{-1, 1\}$

1.21. Pseudoinversas de Moore-Penrose

Dada una matriz arbitraria A tal que $\mathrm{rk}(A) = r > 0$, si se considera una representación

$$A = FR^*$$

donde tanto F como R tienen r columnas linealmente independientes. Es válido definir las matrices.

$$F_A = P_F = F(F^*F)^{-1}F^*$$
$$F_{A^*} = P_R = R(R^*R)^{-1}R^*$$

Ahora es posible considerar la matriz definida por la siguiente expresión.

$$A^+ = R(R^*R)^{-1}(F^*F)^{-1}F^* \tag{1.21.1}$$

Observación 1.21.1. Por cómputo directo es posible verificar lo siguiente.

$$\begin{aligned} AA^+ &= FR^*R(R^*R)^{-1}(F^*F)^{-1}F^* \\ &= F(F^*F)^{-1}F^* = P_F = F_A \end{aligned}$$

También es posible verificar lo siguiente.

$$\begin{aligned} A^+A &= R(R^*R)^{-1}(F^*F)^{-1}F^*FR^* \\ &= R(R^*R)^{-1}R^* = P_R = F_{A^*} \end{aligned}$$

Además, A y A^+ cumplen las siguientes condiciones.

$$AA^+A = A, \quad A^+AA^+ = A^+ \tag{1.21.2}$$

Definición 1.21.2. Dada una matriz A tal que $\mathrm{rk}(A) = r > 0$, se denomina **pseudoniversa** (de Moore-Penrose) o **inversa generalizada** de A, la matriz A^+ definida por la expresión (1.21.1).

Observación 1.21.3. La pseudo inversa A^+ de una matriz A tal que $\mathrm{rk}(A) = r > 0$ cumple las restricciones (1.21.2).

1.21.1. Pseudoinversas y problemas de mínimos cuadrados

Dada una matriz $A \in \mathbb{C}^{m\times n}$ tal que $\mathrm{rk}(A) = r > 0$, es posible demostrar que A^+ es única. Dado un vector $y \in \mathbb{C}^m$, formalmente A^+ resuelve el siguiente problema de mínimos cuadrados.

$$x = \mathrm{argmin}_{z\in\mathbb{C}^n} \|y - Az\|_2^2$$

Donde tal como se establece en el planteamiento del problema, la solución (minimizador) x es el vector $x \in \mathbb{C}^n$ que permite obtener el mínimo valor posible para la expresión $\|y - Ax\|_2^2$. Dado que el vector Ax es la proyección ortogonal de y sobre el espacio imagen de A. Por tanto la solución requerida es

$$x = A^+ y,$$

y además se cumplen las siguientes condiciones.

$$Ax = AA^+y = P_F y$$
$$A^*Ax = A^*AA^+y = A^*P_F y = RF^*F(F^*F)^{-1}F^*y = RF^*y = (FR^*)^*y = A^*y$$

Ejercicio para el lector 1.21.1. Dada una matriz $A \in \mathbb{C}^{n\times m}$ tal que $\mathrm{rk}(A) = r > 0$, y dado $y \in \mathbb{C}^n$. Demostrar que existe un proyector $Q \in \mathbb{C}^{n\times n}$ que cumple la siguiente restricción.

$$\|Qy\|_2^2 = \min_{x\in\mathbb{C}^m} \|Ax - y\|_2^2$$

Es posible utilizar el comando `pinv` de Octave para calcular la pseudoinversa (de Moore-Penrose) de una matriz.

Ejemplo 6. Es posible generar una matriz al azar en $\mathbb{C}^{7\times 7}$ de rango 3 utilizando la siguiente secuencia de comandos.

```
>> A=ceil(randn(7,3));
>> A=A*ceil(randn(7,3))'
A =

   0   0   1  -1   0   0   1
   0   0   1  -1   0   0   1
   1   1   2   0   0  -1  -2
  -1   2   2   1   1  -2  -2
  -1   2   3   0   1  -2  -1
  -3   3   4   0   2  -3   0
  -3   3   1   3   2  -3  -3
```

Es posible calcular el rango $\mathrm{rk}(A)$ utilizando la siguiente secuencia de comandos.

```
>> rank(A)
ans =  3
```

Para calcular A^+ es posible utilizar el comando `pinv` como se muestra en la siguiente secuencia de comandos.

```
>> Ap=pinv(A);
```

Podemos verificar **aproximadamente** algunas de las propiedades fundamentales de A^+ utilizando octave, la razón por la que en general la verificación es aproximada se debe, como se ha discutido anteriormente en el curso, a los efectos de la aritmética finita.

Para calcular el valor $\|AA^+A - A\|_2$, es posible escribir la siguiente secuencia de comandos:

```
>> norm(A*Ap*A-A)
ans =    3.8665e-15
```

El proyector P_F determinado por la observación 1.21.1 puede calcularse aproximadamente con Octave utilizando la siguiente secuencia de comandos basada en la observación 1.21.1.

```
>> Pf=A*Ap;
```

Es posible verificar aproximadamente algunas de las propiedades fundamentales del proyector P_F utilizando la siguiente secuencia de comandos.

```
>> norm(Pf'-Pf)
ans =    7.7442e-16
```

Esta secuencia de comandos aproxima el valor $\|P_F^* - P_F\|_2$.

```
>> norm(Pf^2-Pf)
ans =    6.5079e-16
```

Esta secuencia de comandos aproxima el valor $\|P_F^2 - P_F\|_2$.

```
>> trace (Pf)
ans =  3
>> rank(Pf)
ans =  3
```

Estas secuencias de comandos calculan los valores $\mathrm{tr}(P_F)$ y $\mathrm{rk}(P_F)$, respectivamente. Es posible observar que tal como lo predice la teoría en el ejercicio para el lector 1.20.2,

$$\mathrm{rk}(P_F) = 3 = \mathrm{tr}(P_F).$$

Esta es una de las muchas formas de **invariantes topológicos** que aparecen en álgebra lineal numérica.

Ejercicio para el lector 1.21.2. Dada una matriz $A \in \mathbb{C}^{n\times m}$ tal que $\mathrm{rk}(A) = r > 0$, y dado un vector $y \in \mathbb{C}^n$. Desarrollar un programa Octave que calcula (aproximadamente) el proyector $Q \in \mathbb{C}^{n\times n}$ que cumple la siguiente restricción.

$$\|Qy\|_2^2 = \min_{x\in\mathbb{C}^m} \|Ax - y\|_2^2$$

Ejercicio para el lector 1.21.3. Desarrollar un programa Octave que genera un par de matrices de prueba $A \in \mathbb{C}^{n\times m}, y \in \mathbb{C}^n$, que cumplen las condiciones requeridas por el ejercicio para el lector 1.21.2.

1.22. Subespacios de Krylov

Definición 1.22.1. Dado un polinomio $p(z) = a_n z^n + a_{n-1} z^{n-1} + \cdots + a_1 z + a_0$ con coeficientes complejos, y dada una matriz $A \in \mathbb{C}^{m\times m}$, escribiremos $p(A)$ para denotar la matriz

$$p(A) = a_n A^n + a_{n-1} A^{n-1} + \cdots + a_1 A + a_0 I$$

la matriz $p(A)$ se denomina un polinomio en A. Se denotará por $\mathcal{P}_n$ el conjunto de todos los polinomios con coeficiones complejos de grado $\leq n$.

Definición 1.22.2. Dado un vector $v \in \mathbb{C}^n$ y una matriz $A \in \mathbb{C}^{n\times n}$, se denomina subespacio de Krylov (correspondiente a A, v) de grado m el subespacio $\mathcal{K}_m \subset \mathbb{C}^n$ determinado por la siguiente expresión.

$$\mathcal{K}_m(A, v) = \operatorname{gen}\{v, Av, \ldots, A^{m-1}v\} \tag{1.22.1}$$

Se denota por $\mathbf{K}(A, b)$ la matriz en $\mathbb{C}^{n\times m}$ determinada por la siguiente expresión.

$$\mathbf{K}_m(A, v) = \begin{bmatrix} v & Av & \cdots & A^{m-1}v \end{bmatrix} \tag{1.22.2}$$

Observación 1.22.3. Dado un vector $v \in \mathbb{C}^n$ y una matriz $A \in \mathbb{C}^{n\times n}$, es posible obvervar lo siguiente.

$$\begin{aligned} \mathcal{K}_m(A, v) &= \{p(A)v \ : \ p \in \mathcal{P}_{m-1}\} \\ &= \{\mathbf{K}_m(A, v)a \ : \ a \in \mathbb{C}^m\} \end{aligned}$$

Ejercicio para el lector 1.22.1. Verificar la observación 1.22.3.

Sea J_n la matriz en $\mathcal{C}^n$ determinada por la siguiente expresión.

$$J_n = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & 0 \end{bmatrix} \tag{1.22.3}$$

Dado un vector $v \in \mathbb{C}^n$, una matriz $A \in \mathbb{C}^{n\times n}$ y dado $m \leq n$, es posible obvervar lo siguiente.

$$A\mathbf{K}_m(A, b) = \mathbf{K}_m(A, b)J_m + (A^m v)\hat{e}^*_{m,m} \tag{1.22.4}$$

donde $\hat{e}_{j,m}$ denota la j-ésima columna de la matriz identidad I_m de $m \times m$.

GMRES: Métodos de resíduo mínimo generalizado

Dado un sistemas de ecuaciones lineales

$$Ax = b, \tag{1.22.5}$$

para $A \in \mathbb{C}^{n\times n}$ y $b \in \mathbb{C}^n$. Los métodos iterativos de (subespacios de) Krylov producen sucesiones de la forma $x_k = x_0 + q_k$, donde x_0 es una sposición inicial de la solución de (1.22.5), y donde cada q_k es extraído del subespacio $\mathcal{K}_k(A, r_0)$, para $r_0 = b - Ax_0$. Por la observación 1.22.3, para cada q_k

existe $q \in \mathcal{P}_{k-1}$ tal que $q_k = q(A)r_0$. Si se define el polinomio $p(z) = 1 - zq(z)$, entonces $p \in \mathcal{P}_k$ y es posible medir la convergencia de los métodos utilizando el residuo

$$\begin{aligned} r_k &= b - Ax_k \\ &= b - Ax_0 + Ax_0 - Ax_k \\ &= r_0 - A(x_k - x_0) \\ &= r_0 - Aq_k \\ &= r_0 - Aq(A)r_0 \\ &= (I - Aq(A))r_0 = p(A)r_0 \end{aligned} \tag{1.22.6}$$

Los diversos métodos iterativos de Krylov difieren en la forma en que calculan los polinomios residuales p correspondientes a cada residuo r_k de la forma (1.22.6). El objetivo de los métodos GMRES es resolver el problema:

$$p_k = \operatorname{argmin}_{p \in \mathcal{P}_k, p(0)=1} \|p(A)r_0\|_2 \tag{1.22.7}$$

para cada iteración, encontrando a la vez, un balance entre la factibilidad y computabilidad de las soluciones. Es decir para cada iteración se busca el polinomio $p_k \in \mathcal{P}_k$ tal que $p_k(0) = 1$ y $\|p_k(A)r_0\|_2 \leq \|p(A)r_0\|_2$ para cualquier $p \in \mathcal{P}_k$ tal que $p(0) = 1$. Además, para que sea efectivo, un método iterativo GMRES debe alcanzar valores suficientemente pequeños para $\|r_k\|_2$, para $k \ll n$.

El núcleo del preceso iterativo GMRES es el **proceso de Arnoldi**, un mecanismo que permite calcular construir una base ortonormal $\{u_1, \ldots, u_k\}$ para cada $\mathcal{K}_k(A, r_0)$ siempre que $\operatorname{rk}(\mathbf{K}_k(A, b)) = k > 0$, donde cada u_k puede calcularse aplicando una variación del teorema de ortogonalización de Gram-Schmidt, utilizando las siguientes ecuaciones de recurrencia.

$$\begin{aligned} u_1 &= \frac{1}{\|r_0\|_2} r_0, \\ v_{k+1} &= Au_k - \sum_{j=1}^{k} ((Au_k) \cdot u_j)u_j = Au_k - \sum_{j=1}^{k} (u_j^*(Au_k))u_j \\ u_k &= \frac{1}{\|u_k\|_2} u_k \end{aligned} \tag{1.22.8}$$

Este preceso de ortonormalización puede volverse altamente costoso computacionalemente a medida que se incrementa k.

Para estudiar el proceso de Arnoldi, es conveniente organizar el proceso de ortogonalización en forma matricial. Sea $H_k = [h_{jk}] \in \mathbb{C}^{k \times k}$ la matriz de Hessenberg, cuyos coeficientes cumplen las condiciones,

$$h_{jk} = \begin{cases} u_j^*(Au_k), & j > k+1 \\ 0, & j \leq k+1 \end{cases} \tag{1.22.9}$$

y sea $\tilde{H}_k$ la matriz en $\mathbb{C}^{(k+1) \times k}$ de la forma

$$\tilde{H}_k = \begin{bmatrix} H_k \\ (u_{k+1}^*(Au_k))\hat{e}_{k,k}^* \end{bmatrix} \tag{1.22.10}$$

Si se definen las matrices $U_m = \begin{bmatrix} u_1 & \cdots & u_m \end{bmatrix} \in \mathbb{C}^{n \times k}$. Por (1.22.9) y (1.22.10) se cumplirá lo siguiente.

$$\begin{aligned} AU_k &= U_k H_k + (u_{k+1}^*(Au_k))u_{k+1}\hat{e}_{k,k}^* \\ &= U_{k+1}\tilde{H}_k \end{aligned} \tag{1.22.11}$$

Por ortonormalidad de los vectores $u_1, \ldots, u_{k+1}$, premultiplicando (1.22.11) por U_k^* se obtinene la siguiente expresión.

$$H_k = U_k^* A U_k \tag{1.22.12}$$

Observación 1.22.4. La matriz de Hessenberg H_k es la **restricción/compresión** de la matriz A al subespacio de Krylov $\mathcal{K}_k(A, r_0)$ de grado k.

La versión del algoritmo básico de Arnoldi propuesta por Saad en [13] se presenta a continuación.

Algoritmo 2. **Método de Arnoldi**

Seleccionar $u_1 \in \mathbb{C}^n$ tal que $\|u_1\|_2 = 1$

para $k = 1, \ldots, m$ **hacer**

Calcular $h_{jk} = u_j^*(Au_k)$, **para** $j = 1, \ldots, k$

Calcular $w_k = Au_k - \sum_{j=1}^{k} h_{jk} u_j$

$h_{k+1,k} \leftarrow \|w_k\|_2$

si $h_{k+1,k} = 0$ **entonces** *Detener*

$u_{k+1} \leftarrow w_k / h_{k+1,k}$

fin

El algoritmo 2 permita calcular para un vector arbitrario $v \in \mathbb{C}^n$ y una matriz $A \in \mathbb{C}^{n\times n}$, las matrices $U_k, U_{k+1}, H_k, \tilde{H}_k$ que cumplen las condiciones (1.22.11).

Podemos aplicar el proceso de Arnoldi para calcular cada polinomio p_k que resuelve el problema de optimización (1.22.7). Sea $\{\nu_k\}_{j=1}^{k}$ el conjunto de raíces del polinomio $p_k \in \mathcal{P}_k$ a ser determinado. Consideremos en particular le raíz l-ésima de p_k, se cumplirá entonces que p_k puede representarse en la forma:

$$p_k(z) = \left(1 - \frac{z}{\nu_l}\right) q(z)$$

para algún $q \in \mathcal{P}_{k-1}$. Se cumple entonces que $q_k(A) \in \mathcal{K}_k(A, v)$, de manera que existe $y \in \mathbb{C}^k$ tal que $q(A)r_0 = U_k y$. La optimalidad de mínimos cuadrados de (1.22.7) implica que r_k debe ser ortogonal a cualquier vector en $A\mathcal{K}_k(A, r_0) = \operatorname{gen}\{Ar_0, A^2 r_0, \ldots, A^k r_0\}$, por tanto:

$$0 = (AU_k)^* r_k = U_k^* A^* \left(I - \frac{1}{\nu_l} A\right) U_k y.$$

Sustituyendo la identidad $AU_k = U_{k+1}\tilde{H}_k$ en la expresión previa, se obtiene la siguiente ecuación.

$$\nu_l \tilde{H}_k U_k^* U_k y = \tilde{H}_k^* U_{k+1}^* U_{k+1} \tilde{H}_k y,$$

por tanto, como consecuencia de (1.22.11), ν_l es solución del siguiente problema generalizado de eigenvalores.

$$\tilde{H}_k^* \tilde{H}_k y = \nu_l H_k^* y \tag{1.22.13}$$

Ejemplo 7. Es posible verificar aproximadamente las propiedades de los subespacios de Krylov previamente estudiadas. Consideremos en particular una matriz $A \in \mathbb{C}^{100\times 100}$ y un vector $v \in \mathbb{C}^{100}$ generados al azar utilizando la siguiente secuencia de comandos de Octave.

```
>> n=100;
>> A=randn(n)+i*randn(n);
>> v=randn(n,1)+i*randn(n,1);
```

Es posible calcular las matrices $U_k, U_{k+1}, H_k, \tilde{H}_k$ correspondientes a $\mathcal{K}_k(A, v)$ y $A\mathcal{K}_k(A, v)$, y que cumplen las condiciones (1.22.11), utilizando el comando `krylov` de Octave. Para este ejemplo consideraremos $k = 10$.

```
>> k=10;
>> tic,[Vk, hk, Nk] = krylov (A, v, k+1);toc
Elapsed time is 0.010746 seconds.
>> Hk0=hk(1:k,1:k);
>> Hk1=hk(:,1:k);
>> Uk=Vk(:,1:k);
```

Ahora es posible verificar que las condiciones (1.22.11) se cumplen aproximadamente (debido a los efectos de la aritmética finita, y a los errores de redondeo y truncamiento).

```
>> norm(A*Uk-Vk*Hk1)
ans =    1.7738e-14
```

Es posible calcular aproximadamente el polinomio p_k determinado por el problema (1.22.7), aplicando el comando `eig` de Octave para resolver el problema (1.22.13), utilizando la siguiente secuencia de comandos de Octave.

```
>> tic,l=1./eig(Hk1'*Hk1,Hk0');toc
Elapsed time is 0.0277238 seconds.
>> tic,pk=fliplr(poly(l));toc
Elapsed time is 0.00821304 seconds.
```

Para verificar (aproximadamente) la identidad (1.22.7) es posible aplicar la siguiente secuencia de comandos.

```
>> ptest=[pk(1:k)+1e-9*randn(10*n,k) ones(10*n,1)];
```

Para generar los coeficientes (en formato Octave) de 1000 elementos en $\mathcal{P}_k^1 = \{p \in \mathcal{P}_k : p(0) = 1\}$ cuyos coeficientes se obtienen perturbando los coeficientes (libres) de p_k. Ahora es posible visualizar de forma aproximada la identidad (1.22.7) utilizando la siguiente secuencia de comandos.

```
>> tic,for j=1:(10*n),test(j)=norm(polyvalm(ptest(j,:),A)*v);end;toc
Elapsed time is 16.5601 seconds.
>> L=1:(10*n);
>> plot(L,norm(polyvalm(pk,A)*v)*ones(1,10*n),'r.-',...
> 'markersize',12,L,test,'b.','markersize',12);
>> grid on
>> axis tight
```

La salida gráfica producida por la secuencia de comandos anterior se muestra en la fig. 1.2.

Ejercicio para el lector 1.22.2. Desarrollar un programa Octave llamado Arnoldi.m (**puede utilizar el comando** `krylov` o desarrollar su programa sin utilizar el comando `krylov`), que para cualquier matriz $A \in \mathbb{C}^{n\times n}$, cualquier vector $v \in \mathbb{C}^n$, y cualquier entero positivo $k \leq n$, calcule una base ortonormal de $\{u_1, \ldots, u_k\}$ de $\mathcal{K}_k(A, v)$, y la matriz de Hessenberg H_k correspondiente.

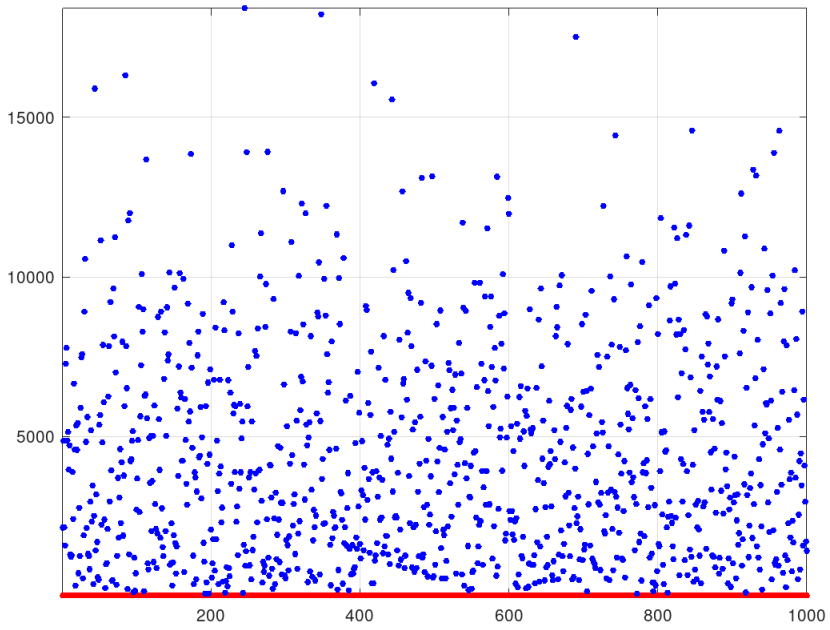


Figura 1.2: En esta gráfica se ilustra la verificación aproximada de la condición (1.22.7) que debe ser cumplida por $p_k \in \mathcal{P}_k^1$.

Ejercicio para el lector 1.22.3. Dada una matriz $A \in \mathbb{C}^{n\times n}$, un vector $v \in \mathbb{C}^n$, un entero positivo $k \leq n$, y el polinomio p_k determinado por (1.22.7) para el subespacio $\mathcal{K}_k(A, b)$. Demostrar que existe un proyector $Q \in \mathbb{C}^{n\times n}$ que cumple la siguiente restricción.

$$\|Qv\|_2 = \|p_k(A)v\|_2$$

Ejercicio para el lector 1.22.4. Desarrollar un programa Octave llamado `MinPoly.m` que para una matriz $A \in \mathbb{C}^{n\times n}$ y un vector $v \in \mathbb{C}^n$, permita calcular el polinomio p_k determinado por (1.22.7) para el subespacio $\mathcal{K}_k(A, b)$, resolviendo directamente el problema de mínimos cuadrados (1.22.7) en lugar del problema generalizado de eigenvalores (1.22.13). Este nuevo programa puede también utilizar el comando `krylov`, en caso de ser necesario.

Ejercicio para el lector 1.22.5. Desarrollar un programa Octave llamado `TestMinPoly.m` que para dos enteros positivos k, n tales que $k \leq n$ (que pueden ingresarse como argumentos de la función `TestMinPoly.m` por el usuario) genere una matriz $A \in \mathbb{C}^{n\times n}$, un vector $v \in \mathbb{C}^n$, y calcule el polinomio p_k determinado por (1.22.7) para el subespacio $\mathcal{K}_k(A, v)$, aplicando el programa `MinPoly.m`, produciendo una salida gráfica similar a la de la fig. 1.2 que permite verificar aproximadamente que el polinomio solución p_k cumple (1.22.7).

Capítulo 2

Optimización Numérica

2.1. Optimización sin Restricciones

Definición 2.1.1. Dado $y \in \mathbb{R}^n$ y dado (un número real) $r > 0$, se denota por $B_r(y)$ la bola (abierta) de radio $r > 0$ centrada en y definida por la expresión:

$$B_r(y) = \{x \in \mathbb{R}^n : \|x - y\|_2 < r\}$$

Cuando se busca minimizar una función objetivo, es posible estar interesado en encontrar un mínimo o bien local, o bien global.

Definición 2.1.2. Un punto $x^* \in \mathbb{R}^n$ se denomina un **minimizador global** para $f \in \mathbb{C}^2(\mathbb{R}^n, \mathbb{R})$ si:

$$f(x^*) \leq f(x) \quad \forall x \in \mathbb{R}^n,$$

mientras que x^* es un **minimizador local** para f si existe un radio $0 < r < \infty$ tal que:

$$f(x^*) \leq f(x) \quad \forall x \in B_r(x^*) \subset \mathbb{R}^n.$$

Dada $f \in C^2(\mathbb{R}^n, \mathbb{R})$, denotamos por $\nabla f(x)$ el gradiente de f definido por la expresión:

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(x) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x) \end{bmatrix}$$

Dada una función $F \in C^2(\mathbb{R}^n, \mathbb{R}^n)$ escribimos $\mathbb{J}F(x)$ para denotar el Jacobiano de F definido por la expresión:

$$\mathbb{J}F(x) = \begin{bmatrix} (\nabla F_1(x))^\top \\ \vdots \\ (\nabla F_n(x))^\top \end{bmatrix} \in C(\mathbb{R}^n, \mathbb{R}^{n \times n})$$

La matriz Hessiana $\mathbb{H}f(x)$ puede definirse en términos de los operadores anteriores, utilizando la siguiente secuencia de operaciones:

$$\mathbb{H}f(x) = \mathbb{J}\nabla f(x)$$

Observación 2.1.3. Cuando $f \in C^2(\mathbb{R}^n, \mathbb{R})$, $\mathbb{H}f(x)$ es una matriz simétrica para cada $x \in \mathbb{R}^n$.

Definición 2.1.4. Un punto $x^* \in \mathbb{R}^n$ se denomina **estacionario** o **crítico** para $f \in \mathbb{C}^2(\mathbb{R}^n, \mathbb{R})$ si $\nabla f(x^*) = 0$, o un punto **regular** si $\nabla f(x^*) \neq 0$.

Definición 2.1.5. La función $f : \mathbb{R}^n \to \mathbb{R}$ se denomina convexa si $\forall x, y \in \mathbb{R}^n$ y $\forall \alpha \in [0, 1]$,

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y);$$

y se denomina Lipschitz continua si existe una constante $L > 0$ tal que

$$|f(x) - f(y)| \leq L\|x - y\|_2, \ \forall x, y \in \mathbb{R}^n.$$

Propiedad 2.1.6. *(Condiciones de optimalidad.) Sea $x^* \in \mathbb{R}^n$. Si x^* es un minimizador para una función $f : \mathbb{R}^n \to \mathbb{R}$) (bien local o global) y si existe $r > 0$ tal que $f \in C^1(B_r(x^*))$, entonces $\nabla f(x) = 0$. Además, si $f \in C^2(B_r(x^*))$, $\mathbb{H}f(x^*)$ es SPSD. Viceversa, sea $r > 0$ tal que $f \in C^2(B_r(x^*))$. Si $\nabla f(x^*) = 0$ y $\mathbb{H}f(x)$ es SPSD para cada $x \in B_r(x^*)$, entonces x^* es un minimizador local para f. Finalemente, si $f \in C^1(\mathbb{R}^n, \mathbb{R})$, convexa en $\mathbb{R}^n$ y $\nabla f(x^*) = 0$, entonces x^* es un minimizador global para f.*

2.2. Aplicación: Prinicipios de Métodos de Gradiente Conjugado

Consideremos una matriz $A \in \mathbb{R}^{n\times n}$ arbitraria y un vector arbitrario $b \in \mathbb{R}^n$, y consideremos el funcional cuadrático:

$$\phi_{A,b}(x) = \frac{1}{2}x^\top Ax - x^\top b$$

Propiedad 2.2.1. *Si $A \in \mathbb{R}^{n\times n}$ es SPD, resolver el sistema $Ax = b$ equivale a resolver el siguiente problema de programación cuadrática.*

$$\min_{x\in\mathbb{R}^n} \phi_{A,b}(x) \tag{2.2.1}$$

La solución de los problemas cuadráticos de la forma (2.2.1) correspondientes a un sistema $Ax = b$ con matriz de coeficientes A SPD, pueden resolverse implementando métodos de gradiente conjugado que producen una secuencia $\{x_k\}_{k\geq 0} \subset \mathbb{R}^n$ tal que $x_k \to x = A^{-1}b$ y donde cada elemento x_k de la sucesión está dado por la expresión:

$$x_k = x_{k-1} + \alpha_{k-1}p_{k-1}, \ k \geq 1$$

donde $\alpha_k \in \mathbb{R}$ y $p_k \in \mathbb{R}^n$ se calculan de acuerdo a criterios de optimzación que serán estudiados en detalle más adelante, y $x_0 \in \mathbb{R}^n$ es un elemento que se utiliza para inicializar el esquema iterativo.

Ejercicio para el lector 2.2.1. Demostrar que:

$$-\nabla\phi_{A,b}(x) = b - Ax$$

Ejercicio para el lector 2.2.2. Demostrar en detalle la Propiedad 2.2.1.

Observación 2.2.2. En un punto específico x_c, $\phi_{A,b}(x_c)$ decrece más rápidamente en la dirección del gradiente:

$$-\nabla\phi_{A,b}(x_c) = b - Ax_c$$

Sea r_c el resíduo $r_c = b - Ax_c$ de x_c. Si $r_c \neq 0$, entonces existe $\alpha > 0$ tal que $\phi_{A,b}(x_c + \alpha r_c) < \phi_{A,b}(x_c)$. En el método del descenso más empinado (con línea de búsqueda exacta) definimos:

$$\alpha = \frac{r_c^\top r_c}{r_c^\top Ar_c}$$

con el fin de minimizar la expresión:

$$\phi_{A,b}(x_c + \alpha r_c) = \phi_{A,b}(x_c) - \alpha r_c^\top r_c + \frac{1}{2}\alpha^2 r_c^\top Ar_c$$

Ejercicio para el lector 2.2.3. Verificar que $\alpha = \frac{r_c^\top r_c}{r_c^\top A r_c}$ minimiza la fórmula $\phi_{A,b}(x_c + \alpha r_c)$ definida por la expresión:

$$\phi_{A,b}(x_c + \alpha r_c) = \phi_{A,b}(x_c) - \alpha r_c^\top r_c + \frac{1}{2}\alpha^2 r_c^\top A r_c$$

Las considereciones anteriores permiten derivar el siguiente algoritmo prototípico para el método del descenso más empinado:

x_0: vector de inicialización

$r_0 \leftarrow b - Ax_0$

$k \leftarrow 0$

mientras $r_k \neq 0$

$k \leftarrow k + 1$

$\alpha_k \leftarrow (r_k^\top r_k)/(r_k^\top A r_k)$

$x_k \leftarrow x_{k-1} + \alpha_k r_{k-1}$

$r_k \leftarrow b - Ax_k$

fin

Ejercicio para el lector 2.2.4. Investigar sobre métodos computacionales de implementación de gradientes conjugados que permitan mejorar las características de convergencia de algoritmos de descenso más empinado como el anterior.

Ejercicio para el lector 2.2.5. Escribir un programa Octave que implemente tanto el algoritmo de descenso más empinado.

Ejercicio para el lector 2.2.6. Escribir un programa Octave que implemente tanto el algoritmo de gradiente conjugado resultante de su investigación.

2.3. Métodos de Newton

Considerando una función $f \in C^2(\mathbb{R}^n, \mathbb{R})$ cuyas primeras y segundas derivadas paraciales podemos calcular. Si se considera ahora el problema determinado por el siguientes sistema de ecuaciones no lineales.

$$\nabla f(x) = 0 \tag{2.3.1}$$

Es posible aplicar una expansión truncada de Taylor en varias variables para calcular una aproximación de primer orden de $\nabla f(x)$ determinada por la siguiente expresión.

$$\nabla f(x + h) \approx \nabla f(x) + (\mathbb{J}\nabla f(x))h \tag{2.3.2}$$

Aplicando (2.3.1) es posible aproximar el vector h en (2.3.2) a través de la siguiente expresión.

$$h \approx -(\mathbb{H}f(x))^{-1}\nabla f(x) \tag{2.3.3}$$

Donde se ha aplicado la siguiente identidad considerada en lecturas previas.

$$\mathbb{H}f(x) = \mathbb{J}\nabla f(x) \tag{2.3.4}$$

Definición 2.3.1. La expresión (2.3.3) permite derivar lo que se denomina en este curso un **método de Newton**

Un método de Newton puede aplicarse para resolver el problema determinado por el cómputo de un punto **estacionario** o **crítico** de f, el cual como se estudio en lecturas previas, puede estar relacionado con problemas de la forma.

$$x = \text{argmin}_{y \in \mathcal{S}} f(y) \tag{2.3.5}$$

donde $\mathcal{S} \subset \mathbb{R}^n$. La notación utilizada en (2.3.5) permite hacer énfasis en que el problema que se busca resolver es el correspondiente al cálculo de un minimizador de global o local de la función objetivo f.

La ideas previamente consideradas para los métodos de Newton pueden aplicarse para derivar un algoritmo prototípico como el que se muestra a continuación.

Algoritmo 3. **Método de Newton**

entradas x_0, N, ε

salidas k, x_k

para $k = 0, \ldots, N$ **hacer**

resolver $\mathbb{H}f(x_k)h_k = -\nabla f(x_k)$

$x_{k+1} \leftarrow x_k + h_k$

si $\|x_{k+1} - x_k\|_2 \leq \varepsilon$ **interrumpir**

fin

2.4. Métodos de Descenso o Búsqueda Lineal

Suposición 2.4.1. Por simplicidad, en esta sección se considerará que $f \in C^2(\mathbb{R}^n, \mathbb{R})$ está acotada inferiormente.

Un método de descenso o búsqueda lineal es un método iterativo que produce una sucesión $\{x_k\}_{k \geq 0}$ donde cada x_{k+1} depende de:

- x_k
- un vector d_k que depende de $\nabla f(x_k)$
- un parámetro adecuado $\alpha_k \in \mathbb{R}$.

Dado $x_0 \in \mathbb{R}^n$ la forma genérica del método puede abreviarse en el siguiente algoritmo prototípico.

Algoritmo 4. **Método Genérico de Búsqueda Lineal**

entradas x_0, N, ε

salidas k, x_k

para $k = 0, \ldots, N$ **hacer**

encontrar dirección $d_k \in \mathbb{R}^n$

calcular $\alpha_k \in \mathbb{R}$

$x_{k+1} \leftarrow x_k + \alpha_k d_k$

si $\|x_{k+1} - x_k\|_2 \leq \varepsilon$ **interrumpir**

fin

Observación 2.4.2. Es importante establecer que el vector d_k debe ser una *dirección de descenso*, en el sentido de que se cumplen las siguientes condiciones:

$$\begin{array}{ll} d_k^\top \nabla f(x_k) < 0, & \text{si } \nabla f(x_k) \neq 0, \\ d_k = 0, & \text{si } \nabla f(x_k) = 0. \end{array} \tag{2.4.1}$$

2.4.1. Direcciones de descenso de uso más frecuente

Direcciones de Newton

$$d_k = -(\mathbb{H}f(x_k))^{-1} \nabla f(x) \tag{2.4.2}$$

Direcciones de Casi-Newton

$$d_k = -H_k^{-1} \nabla f(x) \tag{2.4.3}$$

donde $H_k \approx \mathbb{H}f(x_k)$.

Direcciones Gradiente

$$d_k = -\nabla f(x) \tag{2.4.4}$$

donde $H_k \approx \mathbb{H}f(x_k)$.

Direcciones Gradiente Conjugadas

$$\begin{array}{l} d_0 = -\nabla f(x) \\ d_{k+1} = -\nabla f(x_{k+1}) - \beta_k d_k, \ \ k \geq 0. \end{array} \tag{2.4.5}$$

donde consideraremos que $\beta_k = -\|\nabla f(x_k)\|_2^2 / \|\nabla f(x_{k-1})\|_2^2$.

2.4.2. Estrategias de selección de la longitud de paso α_k

Dada una dirección de búsqueda d_k, en términos de óptimos α_k debería cumplir de forma ideal la siguiente condición.

$$\alpha_k = \text{argmin}_{\alpha \in \mathbb{R}} f(x_k + \alpha d_k) \tag{2.4.6}$$

Calculando una expansión truncada de Taylor de segundo orden alrededor de x_k se obtiene la siguiente expresión.

$$f(x_k + \alpha_k d_k) \approx f(x_k) + \alpha d_k^\top \nabla f(x_k) + \frac{\alpha^2}{2} d_k^\top \mathbb{H} f(x_k) d_k \tag{2.4.7}$$

Aplicando (2.4.7) en el caso particular de una función cuadrática de la forma

$$f(x) = \frac{1}{2} x^\top A x - x^\top b + c \tag{2.4.8}$$

con $A \in \mathbb{R}^{n \times n}$ SPD, $b \in \mathbb{R}^n$, y $c \in \mathbb{R}$, La expansión (2.4.7) es exacta. En este caso $\mathbb{H} f(x_k) = A$, $\nabla f(x_k) = Ax_k - b = -r_k$, para cada $k \geq 0$. Además, en el caso particular de f determinada por (2.4.8), es posible calcular el valor óptimo de alpha aplicando las técnicas aplicadas en la solución de ejercicios de las lecturas previas, obteniendo la siguiente expresión.

$$\alpha_k = \frac{d_k^\top r_k}{d_k^\top A x_k} \tag{2.4.9}$$

Condiciones de Wolfe

En general un criterio apropiada para el cómputo de $\alpha_k > 0$ está basado en lo que se conoce como las **condiciones de Wolfe**, determinadas por las siguientes expresiones.

$$\begin{aligned} f(x_k + \alpha_k d_k) &\leq f(x_k) + \sigma \alpha_k d_k^\top \nabla f(x_k) \\ d_k^\top \nabla f(x_k + \alpha_k d_k) &\geq \delta d_k^\top \nabla f(x_k) \end{aligned} \tag{2.4.10}$$

donde σ, δ son constantes predeterminadas para el problema, que cumplen $0 < \sigma < \delta < 1$.

2.4.3. Método de Descenso con direcciones de casi Newton

Las aproximaciones $H_k \approx \mathbb{H} f(x_k)$ deben ser calculadas de manera que cumplan las siguientes restricciones.

- $H_{k+1}(x_{k+1} - x_k) = \nabla f(x_{k+1}) - \nabla f(x_k)$
- H_k es SPD
- H_k cumple además la siguiente condición.

$$\lim_{k \to \infty} \frac{\|(H_k - \mathbb{H} f(x_k)) d_k\|_2}{\|d_k\|} = 0$$

Es posible calcular las matrices H_k utilizando la estrategia de Broyden, Fletcher, Goldfarb y Shanno (BFGS) basada en la siguiente relación de recurrencia.

$$H_{k+1} = H_k + \frac{1}{y_k^\top s_k} y_k y_k^\top - \frac{1}{s_k^\top H_k s_k} H_k s_k s_k^\top H_k^\top \tag{2.4.11}$$

donde $s_k = x_{k+1} - x_k$ y $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$.

Un algoritmo prototípico de búsqueda lineal con direcciones de casi Newton puede ahora ser derivado.

Algoritmo 5. **Método de Búsqueda Lineal con Direcciones de Casi Newton**

entradas $x_0, H_0 \approx \mathbb{H}f(x_0), N, \varepsilon$

salidas k, x_k

para $k = 0, \ldots, N$ **hacer**

resolver $H_k d_k = -\nabla f(x_k)$

calcular $\alpha_k \in \mathbb{R}$ que cumple (2.4.10)

$x_{k+1} \leftarrow x_k + \alpha_k d_k$

$s_k \leftarrow x_{k+1} - x_k$

$y_k \leftarrow \nabla f(x_{k+1}) - \nabla f(x_k)$

calcular H_k aplicando (2.4.11)

si $\|s_k\|_2 \leq \varepsilon$ **interrumpir**

fin

2.5. Mini Proyecto de Aplicación: Ajuste por Mínimos Cuadrados

Considerando el problema determinado por el ajuste de una colección de datos $\mathcal{D} = \{(x_j, y_j)\}_{j=1}^N \subset \mathbb{R}^2$ a través de una curva determinada por una expresión $y = f(x)$ donde $f(x) = a_0 + a_1 x + a_2 x^2 + \cdots a_n x^n$.

Considerando además la matriz $X \in \mathbb{R}^{N\times(n+1)}$ definida por la expresión.

$$X = \begin{bmatrix} x_1^n & x_1^{n-1} & \cdots & x_1^2 & x_1 & 1 \\ x_2^n & x_2^{n-1} & \cdots & x_2^2 & x_2 & 1 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ x_N^n & x_N^{n-1} & \cdots & x_N^2 & x_N & 1 \end{bmatrix} \tag{2.5.1}$$

Para este proyecto se hace la siguiente suposición.

Suposición 2.5.1. Se supone que $N \geq n+1$, y que las columnas de la matriz $X \in \mathbb{R}^{N\times(n+1)}$ definida por (2.6.1) son linealmente independientes.

Notación 2.5.2. Sean $a \in \mathbb{R}^{n+1}$ y $y \in \mathbb{R}^n$ los vectores definidos por las expresiones.

$$a = \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_1 \\ a_0 \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N-1} \\ y_N \end{bmatrix}, \tag{2.5.2}$$

Ejercicio para el lector 2.5.1. Demostrar que el problema:

$$a = \text{argmin}_{b\in\mathbb{R}^{n+1}} \frac{1}{2}\|Xb - y\|_2^2 \tag{2.5.3}$$

es equivalente a un programa cuadrático de la forma (2.3.5), para f determinado por una expresión de la forma (2.4.8).

Ejercicio para el lector 2.5.2. Desarrollar un programa Octave que resuelva el problema (2.6.3) aplicando un método de búsqueda lineal con direcciones de Newton o casi Newton.

Ejercicio para el lector 2.5.3. Dado un número entero $n > 0$ (determinado por el usuario). Desarrollar un programa Octave que genere conjuntos de datos $\mathcal{D} = \{(x_j, y_j)\}_{j=1}^N \subset \mathbb{R}^2$ tales que la suposición 2.6.1 se cumple. Aplicar el programa desarrollado como parte del ejercicio para el lector 2.6.2 para calcular el polinomio que mejor ajusta los datos en $\mathcal{D}$ en el sentido de los mínimos cuadrados.

2.6. Mini Proyecto de Aplicación: Ajuste por Mínimos Cuadrados

Considerando el problema determinado por el ajuste de una colección de datos $\mathcal{D} = \{(x_j, y_j)\}_{j=1}^N \subset \mathbb{R}^2$ a través de una curva determinada por una expresión $y = f(x)$ donde $f(x) = a_0 + a_1x + a_2x^2 + \cdots a_nx^n$.

Considerando además la matriz $X \in \mathbb{R}^{N\times(n+1)}$ definida por la expresión.

$$X = \begin{bmatrix} x_1^n & x_1^{n-1} & \cdots & x_1^2 & x_1 & 1 \\ x_2^n & x_2^{n-1} & \cdots & x_2^2 & x_2 & 1 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ x_N^n & x_N^{n-1} & \cdots & x_N^2 & x_N & 1 \end{bmatrix} \tag{2.6.1}$$

Para este proyecto se hace la siguiente suposición.

Suposición 2.6.1. Se supone que $N \geq n+1$, y que las columnas de la matriz $X \in \mathbb{R}^{N\times(n+1)}$ definida por (2.6.1) son linealmente independientes.

Notación 2.6.2. Sean $a \in \mathbb{R}^{n+1}$ y $y \in \mathbb{R}^n$ los vectores definidos por las expresiones.

$$a = \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_1 \\ a_0 \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N-1} \\ y_N \end{bmatrix}, \tag{2.6.2}$$

Ejercicio para el lector 2.6.1. Demostrar o refutar que que el problema:

$$a = \text{argmin}_{b\in\mathbb{R}^{n+1}} \frac{1}{2}\|Xb - y\|_2^2 \tag{2.6.3}$$

es equivalente a resolver el siguiente sistema.

$$X^\top Xa = X^\top y \tag{2.6.4}$$

Ejercicio para el lector 2.6.2. Desarrollar un programa Octave que resuelva el problema (2.6.3), para las condiciones adecuadas de $X \in \mathbb{R}^{N\times(n+1)}$.

Ejercicio para el lector 2.6.3. Dado un número entero $n > 0$ (determinado por el usuario). Desarrollar un programa Octave que genere conjuntos de datos $\mathcal{D} = \{(x_j, y_j)\}_{j=1}^{N} \subset \mathbb{R}^2$ tales que la suposición 2.6.1 se cumple. Aplicar el programa desarrollado como parte del ejercicio para el lector 2.6.2 para calcular el polinomio que mejor ajusta los datos en $\mathcal{D}$ en el sentido de los mínimos cuadrados.

2.7. Mini Proyecto: Ajuste de curvas por métodos de mńimos cuadrados no lineales

Si se consideran los siguientes conjuntos de datos:

$$T = \{t_j\}_{j=1}^{8} = \{0{,}055, 0{,}181, 0{,}245, 0{,}342, 0{,}419, 0{,}465, 0{,}593, 0{,}752\}$$
$$Y = \{y_j\}_{j=1}^{8} = \{2{,}80, 1{,}76, 1{,}61, 1{,}21, 1{,}25, 1{,}13, 0{,}52, 0{,}28\}$$

El problema de encontrar la aproximación de mínimos cuadrados $\phi(t) = x_1 + x_2 t + x_3 t^2 + x_4 e^{-x_5 t}$ (con coeficientes $x_1, x_2, \ldots, x_5$) correspondiente a los pares de datos $\{(t_j, y_j)\}_{j=1}^{8}$, puede resolverse aplicando el siguiente procedimiento de optimización numérica.

Solución

El problema de ajuste de los datos considerados anterioremente equivale al problema de optimización.

$$x = \text{argmin}_{x \in \mathbb{R}^5} \sum_{j=1}^{8} |x_1 + x_2 t_j + x_3 t_j^2 + x_4 \exp(-x_5 t_j) - y_j|^2$$

Es posible ingresar los datos t_j, y_j correspondientes a las variables t, y utilizando la siguiente secuencia de comandos.

```
>> t = [
   0.055000
   0.181000
   0.245000
   0.342000
   0.419000
   0.465000
   0.593000
   0.75200];

>> y =[
   2.80000
   1.76000
   1.61000
   1.21000
   1.25000
   1.13000
   0.52000
   0.28000];
```

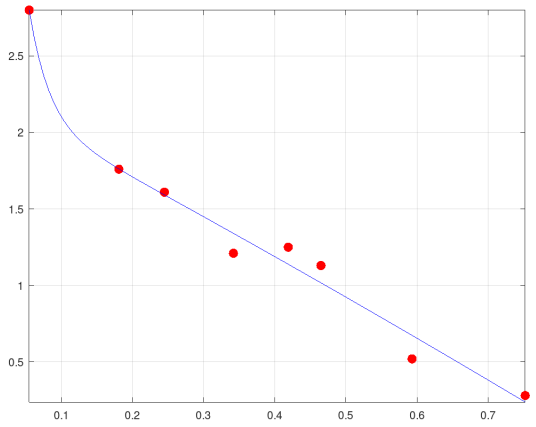


Figura 2.1: Salida gráfica de la solución del computacional del mini-proyecto de ajuste de datos por mínimos cuadrados no lineales.

Para resolver el problema:

$$x = \text{argmin}_{x\in\mathbb{R}^5} \sum_{j=1}^{8} |x_1 + x_2 t_j + x_3 t_j^2 + x_4 \exp(-x_5 t_j) - y_j|^2$$

Es posible definir ahora la función objetivo $\Phi \in C(^2\mathbb{R}^5, \mathbb{R})$ utilizando la siguiente secuencia de comandos.

```
>> Phi=@(x)norm(x(1)+x(2)*t+x(3)*t.^2+x(4)*exp(-x(5)*t)-y,2);
```

Una vez definida la función objetivo es posible aplicar el comando `sqp` de Octave para resolver el problema de optimización correspondiente, utilizando la siguiente secuencia de comandos.

```
>> x0=ones(5,1);
>> [x, obj, info, iter, nf, lambda] = sqp (x0, Phi, []);
```

Para visualizar los resultados es posible utlizar la siguiente secuencia de comandos.

```
>> tt=t(1):(t(8)-t(1))/100:t(8);
>> phi=@(t,x)x(1)+x(2)*t+x(3)*t.^2+x(4)*exp(-x(5)*t);
>> plot(t,y,'r.','markersize',25,tt,phi(tt,x),'b','markersize',15);
>> grid on
>> axis tight
```

La visualización de resultados correspondiente se muestra en la fig. 2.1.

Ejercicio para el lector 2.7.1. Para los datos anteriormente considerados en esta sección. Resolver el problema de ajuste de datos correspondiente al problema de optimización.

$$x = \text{argmin}_{x\in\mathbb{R}^5} \max_{1\leq j\leq 8} |x_1 + x_2 t_j + x_3 t_j^2 + x_4 \exp(-x_5 t_j) - y_j|$$

Visualizar resultados y "comparar con resultados previos", al visualizar las curvas de ajuste producidas por los problemas de optimización correspondientes, junto con los datos, en una misma figura.

Capítulo 3

Métodos de Diferencias Finitas

3.1. Modelos Discretos en Diferencias Finitas y Aproximación de Funciones

3.1.1. Solución Numérica de Ecuaciones Diferenciales por Diferencias Finitas

Problemas de Valor de Frontera y Diferencias Finitas:

Estudiar la solución de las siguientes formas generales de ecuaciones diferenciales utilizando diferencias finitas, desarrollando un algoritmo computacional e implementando el mismo en Octave.

Problema General 1D.

$$\begin{cases} -\frac{d^2u}{dx^2} = f(x), x \in [0,1] \\ u(0) = u(1) = 0 \end{cases}$$

SOLUCIÓN:

ALGORITMO DE SOLUCIÓN PARA EL PROBLEMA GENERAL 1D:

1. Consideremos las fórmulas de tipo $O(h^2)$ estudiadas en clase de la forma:

$$\frac{d^2u}{dx^2}(x) \approx \frac{1}{h^2}(u(x+h) - 2u(x) + u(x-h))$$

2. Consideremos la partición de $[0,1]$ definida por:

$$0 = x_0 < x_1 < x_2 < \cdots < x_{N-1} < x_N = 1$$

donde $x_k = kh$ para $h = 1/N$ y $0 \leq k \leq N$.

3. Definamos $u_k := u(x_k) = u(kh)$, $0 \leq k \leq N$. Tenemos que $u_0 = u(0) = 0$, $u_N = u(x_N) = u(1) = 0$. Aplicando los pasos [1] y [2] podemos obtener una formulación discreta del problema general de la forma:

$$-\frac{1}{h^2}(u_{k-1} - 2u_k + u_{k+1}) = f(x_k), 1 \leq k \leq N-1.$$

4. Podemos ahora utlizar el paso [3] para obtener un sistema de ecuaciones lineales de la forma:

$$\mathbf{L}_2\mathbf{u} = \mathbf{f},$$

donde $\mathbf{u} = [u_1, u_2, \ldots, u_{N-1}]^\top$, $\mathbf{f} = [f(x_1), f(x_2), \ldots, f(x_{N-1})]^\top$ y donde $\mathbf{L}_2$ es la matriz de coeficientes dada por la ecuación:

$$\mathbf{L}_2 = -\frac{1}{h^2}\begin{bmatrix} -2 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \cdots & \vdots \\ 0 & \cdots & 0 & 1 & -2 & 1 & 0 \\ 0 & \cdots & 0 & 0 & 1 & -2 & 1 \\ 0 & \cdots & 0 & 0 & 0 & 1 & -2 \end{bmatrix}$$

5. Resolvemos el sistema del paso [4] utilizando un procedimiento numérico de solución de sistemas de ecuaciones.

6. Representamos las solución exacta y aproximada de la ecuación en forma general en un gráfico combinado. Calculamos y representamos el error relativo.

Práctica 1.

Resolver la siguiente ecuación implementando el algoritmo de solución para el problema general 1D:

$$\begin{cases} -\frac{d^2u}{dx^2} = 8, x \in [0,1] \\ u(0) = u(1) = 0 \end{cases}$$

SOLUCIÓN:

Implementaremos el algoritmo de solución utilizando Octave:

Construimos la malla/partición de $[0,1]$:

```
>> N=1000;
>> h=1/N;
>> x=0:h:1;
```

Construimos la matriz $\mathbf{L}_2$ de coeficientes del sistema:

```
>> L2=spdiags (ones(N-1,1)*[1 -2 1],-1:1,N-1,N-1);
>> L2=-1/h^2*L2;
```

Construimos el vector de forzamiento $\mathbf{f}$:

```
>> f=8*ones((N-1),1);
```

Resolvemos el sistema $\mathbf{L}_2\mathbf{u} = \mathbf{f}$:

```
>> u=L2\f;
```

Definimos y calculamos la solución exacta U_e y la solución aproximada U, incorporando las condiciones de frontera $u(0) = u(1) = 0$ del problema original.

```
>> U=zeros(N+1,1);
>> U(2:N)=u;
>> Ue=4*x.*(1-x);
>> plot(x,Ue,x,U,'g.')
>> plot(x,abs(U-Ue'))
```

Calculamos el error absoluto entre U_e y U:

```
>> norm(U-Ue',inf)
ans =    2.5380e-13
```

Problema General 2D.

$$\begin{cases} -(\partial_x^2 u + \partial_y^2 u) = f(x,y), (x,y) \in [0,1]^2 \\ u(x,y) = 0, (x,y) \in \partial[0,1]^2 \end{cases}$$

SOLUCIÓN:

ALGORITMO DE SOLUCIÓN PARA EL PROBLEMA GENERAL 2D:

1. Consideremos las fórmulas de tipo $O(h^2)$ estudiadas en clase de la forma:

$$\begin{aligned} (\partial_x^2 + \partial_y^2)u(x,y) &\approx \frac{1}{h^2}(u(x+h,y) - 2u(x,y) + u(x-h,y)) \\ &+ \frac{1}{h^2}(u(x,y+h) - 2u(x,y) + u(x,y-h)) \\ &\approx \frac{1}{h^2}(u(x-h,y) + u(x,y-h) - 4u(x,y) + u(x+h,y) + u(x,y+h)) \end{aligned}$$

2. Consideremos la partición de $[0,1]^2$ definida por los pares:

$$(x_k, y_j) \in [0,1]$$

donde $x_k = kh$, $y_j = jh$ para $h = 1/N$ y $0 \leq k, j \leq N$.

3. Definamos $u_{k,j} := u(x_k, y_j) = u(kh, jh)$, $0 \leq k \leq N$. Tenemos que $u_{k,j} = 0$, para $k, j \in 0, 1$. Aplicando los pasos [1] y [2] podemos obtener una formulación discreta del Problema General 2D de la forma:

$$\frac{1}{h^2}(u_{k-1,j} + u_{k,j-1} - 4u_{k,j} + u_{k+1,j} + u_{k,j+1}) = f(x_k, y_j), 1 \leq k, j \leq N-1.$$

4. Podemos ahora utlizar el paso [3] para obtener un sistema de ecuaciones lineales de la forma:

$$\mathbf{L}_{2D}\mathbf{u} = \mathbf{f},$$

donde $\mathbf{u} = [u_{1,1}, u_{1,2}, \ldots, u_{N-1,N-1}]^\top$, $\mathbf{f} = [f(x_1,y_1), f(x_1,y_2), \ldots, f(x_{N-1},y_{N-1})]^\top$ y donde $\mathbf{L}_{2D}$ es la matriz de coeficientes dada por la ecuación:

$$\mathbf{L}_{2D} = \mathbf{I}_{N-1} \otimes \mathbf{L}_2 + \mathbf{I}_{N-1} \otimes \mathbf{L}_2,$$

para $\mathbf{L}_2$ definida por la expresión:

$$\mathbf{L}_2 = -\frac{1}{h^2}\begin{bmatrix} -2 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \cdots & \vdots \\ 0 & \cdots & 0 & 1 & -2 & 1 & 0 \\ 0 & \cdots & 0 & 0 & 1 & -2 & 1 \\ 0 & \cdots & 0 & 0 & 0 & 1 & -2 \end{bmatrix}$$

5. Resolvemos el sistema del paso [4] utilizando un procedimiento numérico de solución de sistemas de ecuaciones.

6. Representamos las solución exacta y aproximada de la ecuación en forma general en un gráfico combinado. Calculamos y representamos el error relativo.

PRÁCTICA 1.

Resolver la siguiente ecuación implementando el algoritmo de solución para el problema general 1D:

$$\begin{cases} -(\partial_x^2 u + \partial_y^2 u) = 32x(1-x) + 32y(1-y), (x,y) \in [0,1]^2 \\ u(x,y) = 0, (x,y) \in \partial[0,1]^2 \end{cases}$$

SOLUCIÓN:

Implementaremos el algoritmo de solución utilizando Octave:

Construimos la malla/partición de $[0,1]^2$:

```
>> N=1000;
>> h=1/N;
>> x=0:h:1;
>> [X,Y]=meshgrid (x);
```

Construimos la matriz $\mathbf{L}_2$ de coeficientes del sistema:

```
>> E=speye(N-1);
>> L2=spdiags (ones(N-1,1)*[1 -2 1],-1:1,N-1,N-1);
>> L2=-1/h^2*L2;
>> L2=kron(E,L2)+kron(L2,E);
```

Construimos el vector de forzamiento $\mathbf{f}$:

```
>> Xi=X(2:N,2:N);
>> Yi=Y(2:N,2:N);
>> f=32*Xi.*(1-Xi)+32*Yi.*(1-Yi);
>> f=f(:);
```

Resolvemos el sistema $\mathbf{L}_2\mathbf{u} = \mathbf{f}$:

```
>> tic,u=L2\f;toc
Elapsed time is 112.165 seconds.
```

Definimos y calculamos la solución exacta U_e y la solución aproximada U, incorporando las condiciones de frontera $u(0) = u(1) = 0$ del problema original.

```
>> U=zeros(N+1,N+1);
>> U(2:N,2:N)=reshape(u,N-1,N-1);
>> Ue=16*X.*(1-X).*Y.*(1-Y);
>> surf(X,Y,Ue)
>> surf(X,Y,abs(Ue-U))
>> shading interp
```

Calculamos el error absoluto entre U_e y U:

```
>> norm(U(:)-Ue'(:),inf)
ans =    4.8561e-12
```

3.1.2. Series de Taylor e Inteporlación de Lagrange

3.1.3. Problemas de Aproximación de funciones:

Estudiar la solución de los siguientes problemas de aproximación de funciones.

PROBLEMA GENERAL 1. Dada una función analítica $f : \mathbb{R} \to \mathbb{R}$, escribiremos $T_a[f](x)$ para denotar el desarrollo de f en series de Taylor de la forma:

$$T_a[f](x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k$$

Escribimos $T_a^n[f](x)$ para denotar el polinomio de la forma:

$$T_a^n[f](x) = \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!}(x-a)^k = f(a) + f'(a)(x-a) + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n$$

La expresión $T_a^n[f](x)$ recibe el nombre de polinomio de Taylor de f cerca de a. Es posible ver que:

$$|T_a^n[f](x) - f| = O(|x-a|^{n+1})$$

cuando $x \approx a$.

PRÁCTICA 1. Sea $f(x) = 1 - x^2$. Calcular $T_0^2[f](x)$.
SOLUCIÓN:

$$T_0^2[f](x) = f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 = 1 + 0x + \frac{-2}{2}x^2 = 1 - x^2$$

PROBLEMA GENERAL 2. Sea f una función determinada por sus valores $f(x_1), \ldots, f(x_n)$ en n puntos $x_1, \ldots, x_n$ en $\mathbb{R}$. Escribimos $L_n[f](x)$ para denotar el polinomio:

$$L_n[f](x) = \sum_{k=1}^{n} f(x_k)\ell_k(x)$$

donde cada $\ell_k(x)$ está definido por la ecuación

$$\ell_k(x) = \prod_{j=1, j\neq k}^{n} \frac{(x-x_j)}{(x_k-x_j)}$$

para cada $k = 1, 2, \cdots, n$. El polinomio $L_n[f](x)$ recibe el nombre de interpolante de Lagrange de f.

PRÁCTICA 1. Calcular $L_3[f](x)$ respecto de $x_1 = -1, x_2 = 0, x_3 = 1$, si $f(x) = 1 - x^2$. SOLUCIÓN: Tenemos que:

$$\ell_1(x) = \frac{(x-x_2)}{(x_1-x_2)}\frac{(x-x_3)}{(x_1-x_3)} = \frac{(x-0)}{(-1-0)}\frac{(x-1)}{(-1-1)} = \frac{1}{2}x(x-1)$$

$$\ell_2(x) = \frac{(x-x_1)}{(x_2-x_1)}\frac{(x-x_3)}{(x_2-x_3)} = \frac{(x-(-1))}{(0-(-1))}\frac{(x-1)}{(0-1)} = 1 - x^2$$

$$\ell_3(x) = \frac{(x-x_1)}{(x_3-x_1)}\frac{(x-x_2)}{(x_3-x_2)} = \frac{(x-(-1))}{(1-(-1))}\frac{(x-0)}{(1-0)} = \frac{1}{2}x(x+1)$$

Aplicando la fórmula para $L_3[f](x)$ obtenemos:

$$L_3[f](x) = f(-1)\ell_1(x) + f(0)\ell_2(x) + f(1)\ell_3(x) = \ell_2(x) = 1 - x^2$$

PRÁCTICA 2. Si sabemos que un determinado modelo toma los valores $f_1 = -1, f_2 = 0, f_3 = 1, f_4 = 0, f_5 = -1$, correspondientes a los valores $s_1 = -2, s_2 = -1, s_3 = 0, s_4 = 1, s_5 = 2$. Calcular $L_5[f](x)$ utilizando Octave.

SOLUCIÓN:
Comenzamos por ingresar los valores del parámetro s:

```
>> s=[-2 -1 0 1 2];
>> f=[-1 0 1 0 -1];
```

Luego calculamos el interpolante de Lagrange

```
>> Lf=polyfit (s,f,length(s)-1);
```

Ahora graficamos el interpolante de lagrange junto con los puntos de referencia de modelo.

```
>> ss=-2:4/100:2;
>> plot(s,f,'ro',ss,polyval(Lf,ss),'g')
```

SOLUCIÓN ALTERNATIVA:
Comenzamos por ingresar los valores del parámetro s:

```
>> s=[-2 -1 0 1 2];
>> f=[-1 0 1 0 -1];
```

Luego calculamos definimos el interpolante de Lagrange

```
>> Lf=@(x)interp1(s,f,x,"spline");
```

Ahora graficamos el interpolante de lagrange junto con los puntos de referencia de modelo.

```
>> ss=-2:4/100:2;
>> plot(s,f,'ro',ss,Lf(ss),'g')
```

3.1.4. Ejercicios de Práctica

1. Resolver las ecuaciones diferenciales implementando diferencias finitas en Octave con $N_x = N_y = 100,000$. En el caso de problemas de tipo VP, calcular los 100 automodos correspondientes a los autovalores más cercanos a 0.

   (a)
   $$\begin{cases} -\frac{d^2u}{dx^2} = 4x(1-x)sen(\pi x), x \in [0,1] \\ u(0) = u(1) = 0 \end{cases}$$

(b)

$$\begin{cases} -\frac{d^2u}{dx^2} = \pi^2\lambda u, x \in [0,1] \\ u(0) = u(1) = 0 \end{cases}$$

(c)

$$\begin{cases} -(\partial_x^2 u + \partial_y^2 u) = sin(\pi xy), (x,y) \in [0,1]^2 \\ u(x,y) = 0, (x,y) \in \partial[0,1]^2 \end{cases}$$

(d)

$$\begin{cases} -(\partial_x^2 u + \partial_y^2 u) = \pi^2\lambda u, (x,y) \in [0,1]^2 \\ u(x,y) = 0, (x,y) \in \partial[0,1]^2 \end{cases}$$

2. Sea $f(x) = sen(\pi x)$. Calcular $T^7_{1/2}[f](x)$.

3. Sea $g(x) = cos(\pi x/2)$. Calcular $L_5[g](x)$ respecto de los puntos $x_1 = -1, x_2 = -1/2, x_3 = 0, x_4 = 1/2, x_5 = 1$. Verificar que $T_0[L_5[g]](x) = L_5[g](x)$ para toda $x \in [-1,1]$.

4. Si el rango promedio de oscilación R en cm. de un puente, en el horario de 7:00 a 8:00 A.M., se ha registrado y promediado cada 5 minutos a partir de las 7:00 A.M. por un período de varios días, obteniendo los siguientes resultados:

$$R = \{0{,}9\ ;\ 1{,}01\ ;\ 0{,}7\ ;\ 0{,}6\ ;\ 1{,}1\ ;\ 0{,}66\ ;\ 0{,}77\ ;\ 0{,}95\ ;\ 0{,}98\ ;\ 0{,}7\ ;\ 0{,}6\ ;\ 0{,}9\ ;\ 1{,}2\}$$

   (a) Calcule una función que permita estimar el rango de oscilación del puente en cualquier momento entre las 7 y las 8 A.M.

   (b) Calcule el rango de oscilación estimado correpondiente a las: 7:02, 7:31, 7:33, 7:42 y 7:57 A.M.

3.2. Interpolación en Dimensiones Superiores y Matrices de Diferenciación

3.2.1. Interpolación en 2D

Estudiar la solución de problemas de interpolación de la Forma:

PROBLEMA GENERAL DE INTERPOLACIÓN EN 2D. Dada una colección de puntos en $\mathbb{R}^2$ de la forma:

		x			
		x_1	x_2	$\cdots$	x_m
y	y_1	(x_1, y_1)	(x_2, y_1)	$\cdots$	(x_m, y_1)
	y_2	(x_1, y_2)	(x_2, y_2)	$\cdots$	(x_m, y_2)
	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
	y_n	(x_1, y_n)	(x_2, y_n)	$\cdots$	(x_m, y_n)

(3.2.1)

junto con una colección de valores correspondientes de la forma:

		x			
		x_1	x_2	$\cdots$	x_m
y	y_1	$f(x_1, y_1)$	$f(x_2, y_1)$	$\cdots$	$f(x_m, y_1)$
	y_2	$f(x_1, y_2)$	$f(x_2, y_2)$	$\cdots$	$f(x_m, y_2)$
	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
	y_n	$f(x_1, y_n)$	$f(x_2, y_n)$	$\cdots$	$f(x_m, y_n)$

Calcular el interpolante de Lagrange $L_{m,n}[f]$ en 2D, que cumple con las condiciones $L_{m,n}[f](x_j, y_k)$ para cada (x_j, y_k) en (3.2.1).

SOLUCIÓN:
Podemos calcular el interpolante de Lagrange utilizando la siguiente ecuación:

$$L_{m,n}[f](x, y) = \sum_{k=1}^{m} \sum_{k=1}^{m} f(x_k, y_j) \ell_{k,j}(x, y)$$

donde $\ell_{k,j}(x, y) = \ell_k(x)\ell_j(y)$ para cada $1 \leq k \leq m$ y cada $1 \leq j \leq n$.
ALGUNAS PROPIEDADES DE LAS FUNCIONES ℓ:

1. Tal como fue establecido en las lecturas de la primer semana dada una colección de puntos $x_1, x_2, \ldots, x_n$:

$$\ell_k(x) = \prod_{j=1, j \neq k}^{n} \frac{(x - x_j)}{(x_k - x_j)}$$

2. Tenemos que:

$$\ell_k(x_j) = \begin{cases} 1, & k = j \\ 0, & k \neq j \end{cases}$$

3. Tenemos que:

$$\ell_{k,l}(x_j, y_i) = \begin{cases} 1, & k = j \text{ y } l = i \\ 0, & \text{otro caso} \end{cases}$$

PRÁCTICA 1. Calcular el interpolante de Lagrange correspondiente a la tabla de datos utilizando Octave:

		x			
		-1	$-0{,}5$	$0{,}5$	1
y	0	-1	-1	-1	-1
	$0{,}25$	$-0{,}5$	$-1{,}5$	$1{,}5$	$-0{,}5$
	$0{,}75$	$0{,}5$	$1{,}5$	$-1{,}5$	$0{,}5$
	1	1	1	1	1

SOLUCIÓN:

1. Generamos la malla correspondiente a la tabla (3.2.1).

```
>> x=[-1 -0.5 0.5 1];
>> y=[0 0.25 0.75 1];
>> [X,Y]=meshgrid (x,y);
```

2. Ingresamos los valores de la tabla:

```
>> Z=[-1 -1 -1 -1;-0.5 -1.5 1.5 -0.5;0.5 1.5 -1.5 0.5;1 1 1 1];
```

3. Construimos los polinomios interpolantes $L_1[f](x), \ldots, L_4[f](x)$ que resuelven los problemas:

		x			
		-1	$-0{,}5$	$0{,}5$	1
y	0	-1	-1	-1	-1

		x			
		-1	$-0{,}5$	$0{,}5$	1
y	$0{,}25$	$-0{,}5$	$-1{,}5$	$1{,}5$	$-0{,}5$

		x			
		-1	$-0{,}5$	$0{,}5$	1
y	$0{,}75$	$0{,}5$	$1{,}5$	$-1{,}5$	$0{,}5$

y

		x			
		-1	$-0{,}5$	$0{,}5$	1
y	1	1	1	1	1

respectivamente.

```
>> for k=1:4,Lfx(k,:)=polyfit (x,Z(k,:),length(x)-1);end
>> format rat
>> Lfx
Lfx =

          0           0           0          -1
         -4        -2/3           4         1/6
          4         2/3          -4        -1/6
          0           0           0           1
```

4. Construimos los polinomios base $\ell_1(y), \ldots, \ell_4(y)$:

```
>> for k=1:4,ly(k,:)=polyfit (y,y==y(k),length(y)-1);end
>> ly
ly =

      -16/3        32/3      -19/3           1
       32/3      -56/3           8          -0
      -32/3        40/3       -8/3           0
       16/3      -16/3           1           0
```

5. Ensamblamos $L_{4,4}[f](x)$ en la forma:

$$L_{4,4}[f](x,y) = \sum_{j=1}^{4} L_j[f](x)\ell_j(y)$$

```
>> Lf=@(x,y)polyval(Lfx(1,:),x).*polyval(ly(1,:),y)+...
polyval(Lfx(2,:),x).*polyval(ly(2,:),y)+...
polyval(Lfx(3,:),x).*polyval(ly(3,:),y)+...
polyval(Lfx(4,:),x).*polyval(ly(4,:),y);
```

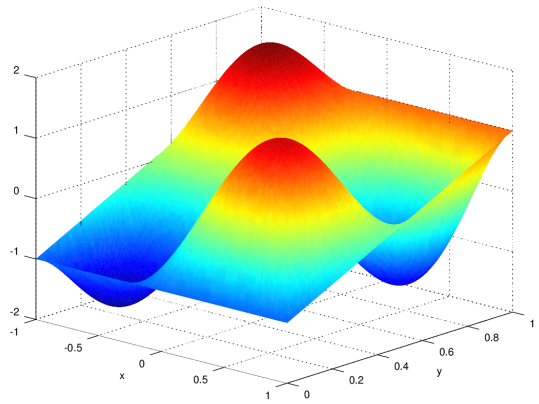
6. Verificamos la condición de interpolación:

```
>> Lf(X,Y)
ans =

         -1          -1          -1          -1
       -1/2        -3/2         3/2        -1/2
        1/2         3/2        -3/2         1/2
          1           1           1           1
```

7. Graficamos el interpolante de Lagrange en una malla más fina.

```
>> [xx,yy]=meshgrid (-1:2/100:1,0:1/100:1);
>> surf(xx,yy,Lf(xx,yy))
>> shading interp
```



SOLUCIÓN ALTERNATIVA:

1. Generamos la malla correspondiente a la tabla (3.2.1).

```
>> x=[-1 -0.5 0.5 1];
>> y=[0 0.25 0.75 1];
>> [X,Y]=meshgrid (x,y);
```

2. Ingresamos los valores de la tabla:

```
>> Z=[-1 -1 -1 -1;-0.5 -1.5 1.5 -0.5;0.5 1.5 -1.5 0.5;1 1 1 1];
```

3. Construimos $L_{4,4}[f](x)$ en la forma:

```
>> Lf=@(x,y)interp2(X,Y,Z,x,y,"spline");
```

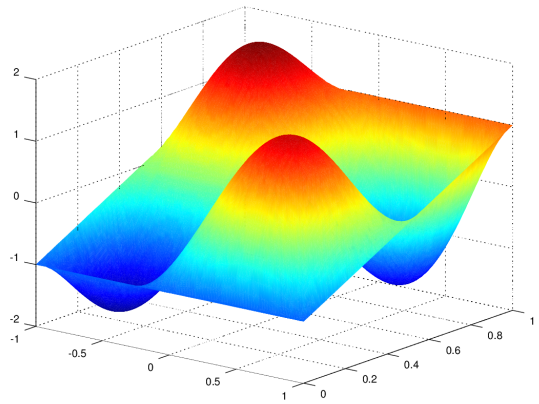
4. Verificamos la condición de interpolación:

```
>> Lf(X,Y)
ans =

          -1          -1          -1          -1
        -1/2        -3/2         3/2        -1/2
         1/2         3/2        -3/2         1/2
           1           1           1           1
```

5. Graficamos el interpolante de Lagrange en una malla más fina.

```
>> [xx,yy]=meshgrid (-1:2/100:1,0:1/100:1);
>> surf(xx,yy,Lf(xx,yy))
>> shading interp
```



3.2.2. Matrices de Diferenciación

Estudiar la solución de problemas de aproximación de la derivada de una función diferenciable de la forma:

PROBLEMA GENERAL DIFERNCIACIÓN APROXIMADA POR MATRICES DE DIFERENCIACIÓN. Dada una Tabla de valores conocidos/calculables de un modelo f en $\mathbb{R}^2$ dada por la expresión:

x	x_1	x_2	$\cdots$	x_n
$f(x)$	$f(x_1)$	$f(x_2)$	$\cdots$	$f(x_n)$

Calcuar n fórmulas que permitan estimar $f'(x_j)$ para $j = 1, \ldots, n$, en la forma:

$$f'(x_k) = \sum_{j=1}^{n} d_{k,j} f(x_j)$$

y que además sean exactas cuando f es un polinomio de grado al menos $n-1$. Podemos representar las ecuaciones anteriores matricialmente como:

$$\begin{bmatrix} f'(x_1) \\ f'(x_2) \\ \vdots \\ f'(x_n) \end{bmatrix} = \begin{bmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,n} \\ d_{2,1} & d_{2,2} & \cdots & d_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n,1} & d_{n,2} & \cdots & d_{n,n} \end{bmatrix} \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{bmatrix}$$

donde cada coeficiente $d_{k,j}$ está dado por la fórmula:

$$d_{k,j} = \ell_j'(x_k) \tag{3.2.2}$$

la matriz $\mathbf{D} = [d_{k,j}]_{n\times n}$ recibe el nombre de matriz de diferenciación.

PRÁCTICA 1. Dados los puntos $x_1 = -1, x_2 = 0, x_3 = 1$. Calcular la matriz de diferenciación correspondiente a la solución del problema general de diferenciacón aproximada.

SOLUCIÓN:

1. Calcular los polinomios base de interpolación ℓ_1, ℓ_2, ℓ_3 respecto de los puntos x_1, x_2, x_3:

$$\ell_1(x) = \frac{(x-x_2)}{(x_1-x_2)}\frac{(x-x_3)}{(x_1-x_3)} = \frac{(x-0)}{(-1-0)}\frac{(x-1)}{(-1-1)} = \frac{1}{2}x^2 - \frac{1}{2}x$$

$$\ell_2(x) = \frac{(x-x_1)}{(x_2-x_1)}\frac{(x-x_3)}{(x_2-x_3)} = \frac{(x-(-1))}{(0-(-1))}\frac{(x-1)}{(0-1)} = -x^2 + 1$$

$$\ell_3(x) = \frac{(x-x_1)}{(x_3-x_1)}\frac{(x-x_2)}{(x_3-x_2)} = \frac{(x-(-1))}{(1-(-1))}\frac{(x-0)}{(1-0)} = \frac{1}{2}x^2 + \frac{1}{2}x$$

2. Calcular las derivadas de los polinomios base:

$$\ell_1'(x) = x - \frac{1}{2}$$

$$\ell_2'(x) = -2x$$

$$\ell_3'(x) = x + \frac{1}{2}$$

3. Calcular la matrix de diferenciación $\mathbf{D}$ de acuerdo a la fórmula (3.2.2):

$$\mathbf{D} = \begin{bmatrix} \ell_1'(-1) & \ell_2'(-1) & \ell_3'(-1) \\ \ell_1'(0) & \ell_2'(0) & \ell_3'(0) \\ \ell_1'(1) & \ell_2'(1) & \ell_3'(1) \end{bmatrix} = \begin{bmatrix} -3/2 & 2 & -1/2 \\ -1/2 & 0 & 1/2 \\ 1/2 & -2 & 3/2 \end{bmatrix}$$

PRÁCTICA 2. Verificar las fórmulas de la PRÁCTICA 1 con la función $f(x) = 3x^2 - 4x + 1$.

SOLUCIÓN:

1. Tenemos que la función tiene una representación tabular respecto de x_1, x_2, x_3 de la forma:

x	-1	0	1
$f(x)$	8	1	0

2. Tenemos que $f'(x) = 6x - 4$ tiene una representación tabular:

x	-1	0	1
$f'(x)$	-10	-4	2

3. En forma matricial tenemos que:

$$\mathbf{Df} = \begin{bmatrix} -3/2 & 2 & -1/2 \\ -1/2 & 0 & 1/2 \\ 1/2 & -2 & 3/2 \end{bmatrix} \begin{bmatrix} 8 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -10 \\ -4 \\ 2 \end{bmatrix}$$

el resultado de la operación anterior verifica predicción acerca del nivel de aproximación de las fórmulas.

PRÁCTICA 3. Crear un archivo con nombre `dmat.m` utilizando el editor de Octave, dicho archivo generará una matrix de diferenciación a partir de cualquier colección de puntos $x_1, x_2, \ldots, x_n$ que reciba.

SOLUCIÓN:

1. El archivo `dmat.m` puede crearse implementando las siguientes líneas de código de Octave:

```
function D=dmat(x)
n=length(x);
for i=1:n
l(i,:)=polyfit(x,x==x(i),n-1);
endfor
for k=1:n
for j=1:n
D(k,j)=polyval(polyder(l(j,:)),x(k));
endfor
endfor
endfunction
```

PRÁCTICA 4. Utilizar el archivo `dmat.m` para desarrollar la práctica 3.

SOLUCIÓN:

1. Generamos los puntos $x_1 = -1, x_2 = 0, x_3 = 1$:

```
>> x=[-1 0 1];
```

2. Generamos la matrix **D** con `dmat.m`:

```
>> format rat
>> D=dmat(x)
D =

       -3/2          2       -1/2
       -1/2          0        1/2
        1/2         -2        3/2
```

3. Generamos el vector **f**:

```
>> f=3*x.^2-4*x+1;
>> f=f'
f =

          8
          1
          0
```

4. Calculamos la diferenciación exacta $\mathbf{f}'$:

```
>> fp=6*x-4;
>> fp=fp'
fp =

        -10
         -4
          2
```

5. Calculamos la diferenciación aproximada **Df**:

```
>> D*f
ans =

        -10
         -4
          2
```

6. Verificamos el error de aproximación:

```
>> norm(fp-D*f,inf)
ans = 0
```

PRÁCTICA 5. Calcular analítica y computacionalmente la sub-matriz $\mathbf{D}_2$ (el renglón 2) de la matrix $\mathbf{D}$ de la práctica 1.

SOLUCIÓN:
Analíticamente:

$$\mathbf{D}_2 = [d_{2,1}\ \ d_{2,2}\ \ d_{2,3}] = [\ell_1'(0)\ \ \ell_2'(0)\ \ \ell_3'(0)] = \left[-\frac{1}{2}\ \ 0\ \ \frac{1}{2}\right]$$

Computacionalmente:

```
>> x=[-1 0 1];
>> D=dmat(x);
>> D(2,:)
ans =

       -1/2           0         1/2
```

3.2.3. Representaciones matriciales alternativas del operador de diferenciación

Consideremos nuevamente el operador $D : C^{n+1}([a,b]) \to C^n([a,b]) : f \mapsto Df$, para $n \in \mathbb{Z}_0^+$. Tenemos que la siguiente propiedad que puede ser deducida por el lector a manera de ejercicio:

Propiedad: Si $x_0, x_1, \ldots, x_N \in \mathbb{R}$ son distintos, entonces la función cardinal $c_j(x)$ definida por

$$c_j(x) = \frac{1}{p_j} \prod_{k=0, k\neq k}^{N} (x - x_k), \tag{3.2.3}$$

$$p_j = \prod_{k=0, k\neq k}^{N} (x_j - x_k) \tag{3.2.4}$$

es el único interpolante polinómico de grado N que cumple la condición.

$$c_j(x_k) = \begin{cases} 1, & j = k \\ 0, & j \neq k \end{cases}, 0 \leq j, k \leq N$$

Tenemos además que la matriz de diferenciación $D \in \mathbb{C}^{N\times N}$ correspondiente a $\{c_0(x), c_1(x), c_2(x), \ldots, c_N(x)\}$ tiene una representación alternativa determinada por las fórmulas:

$$D_{ij} = c_j'(x_i) = \frac{1}{p_j} \prod_{k=0, k\neq i,j}^{N} (x_i - x_k) = \frac{p_i}{p_j(x_i - x_j)} \quad (i \neq j) \tag{3.2.5}$$

$$D_{ii} = c_i'(x_i) = \sum_{k=0, k\neq i}^{N} (x_i - x_k)^{-1}. \tag{3.2.6}$$

El siguiente código Octave/MATLAB que puede ser creado como el archivo `Dmat.m` calcula una matriz de diferenciación D correspondiente a una partición $x_0 < x_1 < \cdots < x_{N-1} < x_N$.

```
function [D,x]=Dmat(x)
x=x(:);
N=length(x)-1;
D=repmat(x,1,N+1);
E=eye(N+1);
D=D'-D+E;
p=diag(prod(D));
s=diag(sum(1./D)-1);
D=p*(1./D')/p-E+s;
endfunction
```

El lector puede ahora repetir las prácticas anteriores correspondientes al cómputo con matrices de diferenciación, para verificar la equivalencia entre representaciones.

3.2.4. Ejercicios de Práctica

1. Verificar la propiedad 3 de las funciones base ℓ:

$$\ell_{k,l}(x_j, y_i) = \begin{cases} 1, & k = j \text{ y } l = i \\ 0, & \text{otro caso} \end{cases}$$

2. Probar que para cualquier $\ell_j(x)$ correspondiente a una colección de puntos $x_1, x_2, \ldots, x_n \in \mathbb{R}$. $T_{x_k}^{n-1}[\ell_j](x) = \ell_j(x)$ para cada punto x_j en la colección.

3. Resolver analíticamente el problema de interpolación:

		x	
		-1	1
y	0	-1	1
	1	1	-1

4. Se denominan puntos de Chebyshev de orden n a cada colección de puntos $x_1, \ldots, x_{n+1}$ definidos por $x_k = \cos(\pi(k-1)/n)$ para $1 \leq k \leq (n+1)$. Calcular la matrix de diferenciación **D** correspondiente a los puntos de Chebyshev de orden 6.

3.3. Método de Diferencias Finitas (MDF) en Dimensiones Superiores

3.3.1. Solución Numérica de Modelos Dinámicos

MDF para Problemas de Valor Inicial y de Frontera:

Estudiar la solución de las siguientes formas generales de ecuaciones diferenciales utilizando diferencias finitas, desarrollando un algoritmo computacional e implementando el mismo en Octave/SciLab.

PROBLEMA GENERAL DE MOVIMIENTO ONDULATORIO EN 1D: Utilizamos este tipo de modelos para simular la deflexión de **varillas** representativas de una estructura dada.

$$\begin{cases} \partial_t^2 u - \alpha^2 \partial_x^2 u = f(x,t), (x,t) \in [0,1] \times [0,\infty) \\ u(0,t) = u(1,t) = 0 \\ u(x,0) = u_0(x) \\ \partial_t u(x,0) = v_0(x) \end{cases}$$

SOLUCIÓN:

ALGORITMO DE SOLUCIÓN PARA EL PROBLEMA GENERAL DE MOVIMIENTO ONDULATORIO EN 1D:

1. Consideremos las fórmulas estudiadas en clase de la forma:

$$\partial_t u(x,t) \approx \frac{1}{h_t}(u(x,t+h) - u(x,t))$$

$$\partial_x^2 u(x,t) \approx \frac{1}{h^2}(u(x-h,t) - 2u(x,t) + u(x+h,t))$$

2. Consideremos las particiones de $[0,1] \times [0,T]$ definidas por:

$$0 = x_0 < x_1 < x_2 < \cdots < x_{N-1} < x_N = 1$$

$$0 = t_0 < x_1 < x_2 < \cdots < t_{M-1} < t_M = T$$

donde $x_k = kh$, $t_j = jh_t$ para $h = 1/N$, $h_t \leq h^2/\alpha^2$ y $0 \leq k \leq N$.

3. Reduzcamos el orden del modelo diferencial a la forma:

$$\begin{cases} \partial_t u = v, \\ \partial_t v = \alpha^2 \partial_x^2 u + f(x,t), (x,t) \in [0,1] \times [0,\infty) \\ u(0,t) = u(1,t) = 0 \\ u(x,0) = u_0(x) \\ v(x,0) = v_0(x) \end{cases}$$

Llamamos a las ecuaciones resultantes de esta reducción, ecuaciones dinámicas representativas de primer orden, o solo ecuaciones dinámicas por simplicidad.

4. Definamos $u_{k,j} := u(x_k, t_j) = u(kh, jh_t)$ y $v_{k,j} := v(x_k, t_j) = v(kh, jh_t)$, $0 \leq k, j \leq N$. Aplicando los pasos [1], [2] y [3] podemos obtener una formulación discreta del problema general de la forma:

$$\begin{cases} u_{k,j+1} = u_{k,j} + h_t v_{k,j} \\ v_{k,j+1} = v_{k,j} + \frac{h_t \alpha^2}{h^2}(u_{k-1,j} - 2u_{k,j} + u_{k+1,j}) + h_t f(x_k, t_j) \end{cases}, 1 \leq k \leq N-1.$$

5. Podemos ahora utlizar el paso [4] para obtener una representación matricial de la discretización de la forma:

$$\begin{cases} \mathbf{u}_{j+1} = \mathbf{u}_j + h_t \mathbf{v}_j \\ \mathbf{v}_{j+1} = \mathbf{v}_j + h_t \alpha^2 \mathbf{L}_{1d} \mathbf{u}_j + h_t \mathbf{f}_j \end{cases}, 1 \leq k \leq N-1.$$

donde $\mathbf{u}_j = [u_{1,j}, u_{2,j}, \ldots, u_{N-1,j}]^\top$, $\mathbf{f}_j = [f(x_1, t_j), f(x_2, t_j), \ldots, f(x_{N-1}, t_j)]^\top$ y donde $\mathbf{L}_{1d}$ es la matriz de coeficientes dada por la ecuación:

$$\mathbf{L}_{1d} = \frac{1}{h^2} \begin{bmatrix} -2 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \cdots & \vdots \\ 0 & \cdots & 0 & 1 & -2 & 1 & 0 \\ 0 & \cdots & 0 & 0 & 1 & -2 & 1 \\ 0 & \cdots & 0 & 0 & 0 & 1 & -2 \end{bmatrix}$$

6. Resolvemos el sistema del paso [5] utilizando un procedimiento nuérico de integración numérica en el tiempo.

7. Representamos las solución aproximada de la ecuacón en forma general en un gráfico dinámico.

PRÁCTICA 1.

Simular la deflexión de una varilla representativa descrita por una ecuación de la forma:

$$\begin{cases} \partial_t^2 u - \alpha^2 \partial_x^2 u = 0, (x,t) \in [0,1] \times [0, \infty) \\ u(0,t) = u(1,t) = 0 \\ u(x,0) = Asen(\pi m x) \\ \partial_t u(x,0) = 0 \end{cases}$$

SOLUCIÓN: Implementaremos el algoritmo de solución utilizando Octave:

1. Creamos el archivo `*.m` cuyo código se presenta a continuación:

```
function [U,x,T]=varilla_simple(M,N,c2,m,A)
h=1/N;
ht=1/M;
x=0:h:1;
L2=spdiags (ones(N-1,1)*[1 -2 1],-1:1,N-1,N-1);
L2=1/h^2*L2;
u0=A*sin(pi*m*x(2:N));
v0=zeros(N-1,1);
```

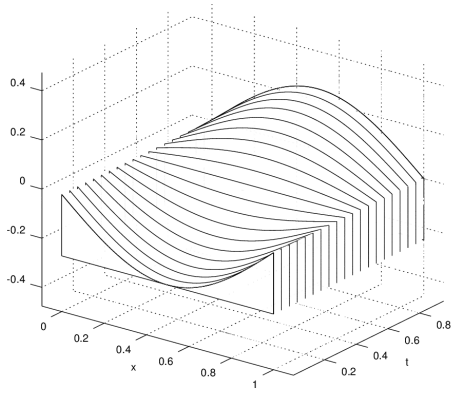
```
var=@(xx,u)interp1(x,u,xx,"spline");
u=u0(:);
v=v0(:);
U=[];
T=U;
for j=1:M
u=u+ht*v;
v=v+ht*c2*L2*u;
if mod(j-1,floor(M/20))==0
U=[U [0;u;0]];
T=[T j*ht];
endif
plot(x,[0;u;0],0:1/10:1,var(0:1/10:1,[0;u;0]),'ro');
axis([0 1 -1 1]);
pause(.2);
endfor
endfunction
```

2. Realizamos un experimento computacional con los parámetros $N = 100, M = 400, A = -0{,}25, \alpha^2 = c^2 = 1$ y $m = 1$.

```
>> [u,x,t]=varilla_simple(400,100,1,1,-.25);
```

3. Creamos un gráfico dinámico que muestra la evolución en el tiempo de la varilla representativa.

```
>> [X,T]=meshgrid (x,t);
>> waterfall (X,T,u')
>> axis equal
>> xlabel ('x')
>> ylabel ('t')
```



SOLUCIÓN: Implementaremos ahora el algoritmo de solución utilizando el método de Crank-Nicolson en Octave:

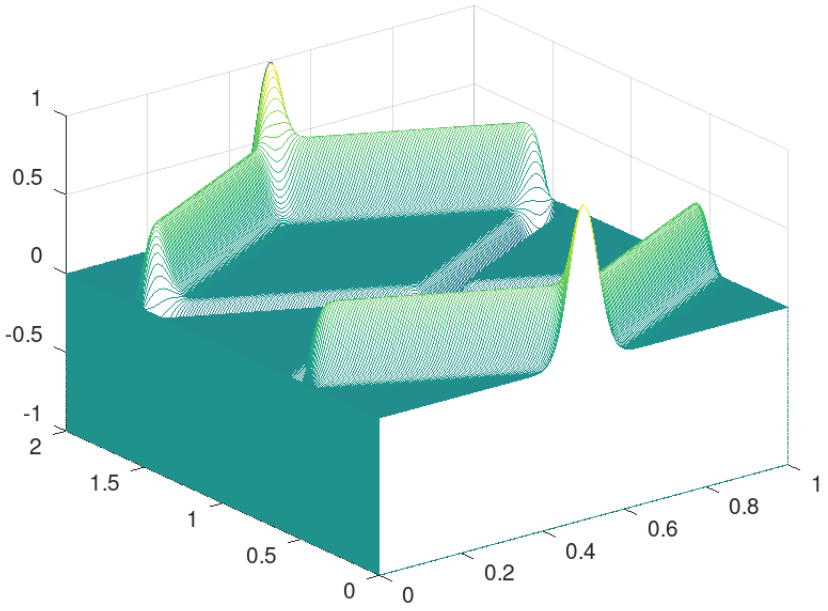
1. Creamos el archivo `*.m` cuyo código se presenta a continuación:

```
function [t,x,w,wdata]=Wave1DCN(c,L,N,m,t,ss)
%%
% Examples:
% [t,x,w,wdata]=Wave1DCN(1,1,400,800,[0 2],400);
%%
hx=L/N;
ht=diff(t)/m;
L1D=c*spdiags(ones(N-1,1)*[1 -2 1],-1:1,N-1,N-1)/hx^2;
L1D=[sparse(N-1,N-1),speye(N-1);L1D,sparse(N-1,N-1)];
L1D=ht*L1D;
E=2*speye(2*(N-1));
x=0:hx:L;
w0=4*exp(-(400/L)*(x-L/2).^2).*x.*(L-x)/L^2;
w=w0';
w0=w(2:N);
w0=[w0;zeros(N-1,1)];
wdata=w0;
ss=ceil(m/ss);
H0=(E+L1D);
H1=(E-L1D);
for k=1:m
plot(x,[0;w0(1:(N-1),1);0]);
axis([0,L,-1.5,1.5]);
pause(.2);
w0=H0*w0;
w0=H1\w0;
if mod(k-1,ss)==0
w=[w,[0;w0(1:(N-1),1);0]];
wdata=[wdata,w0];
end
end
w=[w,[0;w0(1:(N-1),1);0]];
wdata=[wdata,w0];
Nw=size(w,2);
[x,t]=meshgrid(x,t(1):diff(t)/(Nw-1):t(2));
waterfall(x,t,w')
end
```

2. Realizamos un experimento computacional con los parámetros $N = 100, M = 400, \alpha^2 = c^2 = 1$ y $L = T = 1$.

```
>> [t,x,w,wdata]=Wave1DCN(1,1,400,800,[0 2],400);
```

3. El programa desarrolla una animación de la evolución de la deflexión de la varilla representativa y calcula un gráfico dinámico que muestra la evolución en el intervalo de tiempo $[0, 1]$.



PROBLEMA GENERAL DE CÁLCULO DE MODOS DE VIBRACIÓN DE VARILLAS REPRESENTATIVAS (Representación de Helmholtz): Dado un problema de deflexión de varillas de la forma:

$$\begin{cases} \partial_t^2 u - \alpha^2 \partial_x^2 u = f(x,t), (x,t) \in [0,1] \times [0,\infty) \\ u(0,t) = u(1,t) = 0 \\ u(x,0) = u_0(x) \\ \partial_t u(x,0) = v_0(x) \end{cases}$$

Hacemos las suposiciones $u(x,t) = v(x)e^{i\omega t}$ y $f(x,t) = g(x)e^{i\omega t}$, donde $i = \sqrt{-1}$. Bajo condiciones físicamente apropiadas y realizables tenemos que las ecuaciones anteiores producen una expresión de la forma:

$$\begin{cases} e^{i\omega t}(i\omega)^2 v(x) - e^{i\omega t}\alpha^2 \partial_x^2 v(x) = g(x)e^{i\omega t}, (x,t) \in [0,1] \times [0,\infty) \\ v(0)e^{i\omega t} = v(0)e^{i\omega t} = 0 \end{cases}$$

Luego de simplificar la expresión anterior obtenemos el siguiente resultado:

$$\begin{cases} -\alpha^2 \partial_x^2 v(x) - \omega^2 v(x) = g(x), x \in [0,1] \\ v(0) = v(1) = 0 \end{cases}$$

La expresión anterior se denomina representación de Helmholtz del problema de deflexión, y corresponde a un problema de valor de frontera que podemos resolver con el MDF utilizando el siguiente algoritmo:

SOLUCIÓN:
El procedimiento de solución es el siguiente:

1. Calcular la representación de Helmholtz del problema de deflexión para obetener la expresión:

$$\begin{cases} -\alpha^2 \partial_x^2 v(x) - \omega^2 v(x) = g(x), x \in [0,1] \\ v(0) = v(1) = 0 \end{cases}$$

2. Discretizar la expresión del paso [1] utilizando las técnicas de las lecturas de la semana 1 para obtener:
$$-\frac{\alpha^2}{h^2}(v_{k-1} - 2v_k + v_{k+1}) - \omega^2 v_k = g(x_k), 1 \leq k \leq N-1,$$
donde $x_k = kh$, $h = 1/N$, para $N \in \mathbb{Z}^+$.

3. Calcular la representación matricial de la discretización del paso [2] de la forma:
$$-\alpha^2 \mathbf{L}_{1d}\mathbf{v} - \omega^2 \mathbf{v} = \mathbf{g}$$
donde $\mathbf{L}_{1d}$ está determinada por la expresión
$$\mathbf{L}_{1d} = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \cdots & \vdots \\ 0 & \cdots & 0 & 1 & -2 & 1 & 0 \\ 0 & \cdots & 0 & 0 & 1 & -2 & 1 \\ 0 & \cdots & 0 & 0 & 0 & 1 & -2 \end{bmatrix}$$
y donde $\mathbf{v} = [v_1, v_2, \ldots, v_{N-1}]^\top$, $\mathbf{g} = [g(x_1), g(x_2), \ldots, g(x_{N-1})]^\top$.

4. Resolver el problema de álgebra lineal del paso [3].

PRÁCTICA 2.

Utilizar el MDF implementado en Octave para resolver numéricamente el problema de Helmholtz correspondiente a la ecuación:
$$\begin{cases} \partial_t^2 u - \pi^2 \partial_x^2 u = 100 sen(\pi x) e^{i\omega t}, (x,t) \in [0,1] \times [0,\infty) \\ u(0,t) = u(1,t) = 0 \\ u(x,0) = u_0(x) \\ \partial_t u(x,0) = v_0(x) \end{cases}$$
Suponiendo que $\omega = 1$. SOLUCIÓN:

1. La representación de Helmholtz del problema esta dada por:
$$\begin{cases} -\pi^2 \partial_x^2 v(x) - \omega^2 v(x) = 100 sen(\pi x), x \in [0,1] \\ v(0) = v(0) = 0 \end{cases}$$

2. La expresión del paso [2] tiene una discretización de la forma:
$$-\frac{\pi^2}{h^2}(v_{k-1} - 2v_k + v_{k+1}) - \omega^2 v_k = 100 sen(\pi x_k), 1 \leq k \leq N-1,$$
donde $x_k = kh$, $h = 1/N$, para $N \in \mathbb{Z}^+$.

3. La forma matricial de la discretización del paso [2] es:
$$-\pi^2 \mathbf{L}_{1d}\mathbf{v} - \omega^2 \mathbf{v} = \mathbf{g}$$

donde $\mathbf{L}_{1d}$ está determinada por la expresión

$$\mathbf{L}_{1d} = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \cdots & \vdots \\ 0 & \cdots & 0 & 1 & -2 & 1 & 0 \\ 0 & \cdots & 0 & 0 & 1 & -2 & 1 \\ 0 & \cdots & 0 & 0 & 0 & 1 & -2 \end{bmatrix}$$

y donde $\mathbf{v} = [v_1, v_2, \ldots, v_{N-1}]^\top$, $\mathbf{g} = [100sen(\pi x_1), 100sen(\pi x_2), \ldots, 100sen(\pi x_{N-1})]^\top$.

4. Supongamos que $\omega = 1$, podemos resolver el problema numéricamente utilizando los pasos anteriores y el siguiente procedimiento en Octave:

   4.1 Calculamos el mallado de $[0, 1]$:

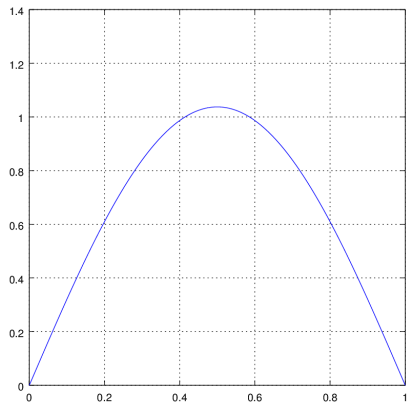
```
>> N=1000;
>> h=1/N;
>> x=0:h:1;
```

   4.2 Calculamos la representación matricial de la discretización:

```
>> alfa2=pi^2;
>> omega2=1;
>> L1d=spdiags (ones(N-1,1)*[1 -2 1],-1:1,N-1,N-1);
>> L1d=1/h^2*L1d;
>> H=-alfa2*L1d-omega2*speye(N-1);
>> g=100*sin(pi*x(2:N))';
```

   4.3 Reolvemos el problema matricial del paso [4.2] y graficamos la solución numérica:

```
>> v=H\g;
>> plot(x,[0;v;0])
```



4.4 Calculamos una malla más fina de $[0, 1]$:

```
>> M=10000;
>> he=1/M;
>> xe=0:he:1;
```

4.5 Recalculamos la discretización en la malla más fina:

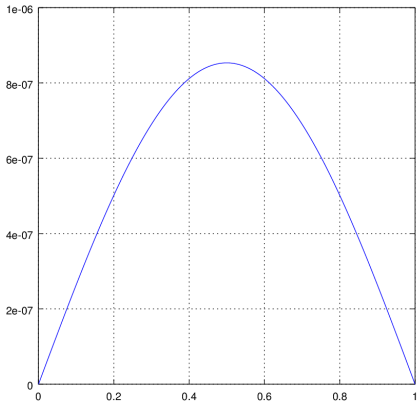
```
>> L1e=spdiags (ones(M-1,1)*[1 -2 1],-1:1,M-1,M-1);
>> L1e=1/he^2*L1e;
>> He=-alfa2*L1e-omega2*speye(M-1);
>> ge=100*sin(pi*xe(2:M))';
```

4.6 Recalculamos la solución con mayor precisión:

```
>> ve=He\ge;
```

4.7 Utilizamos este cómputo para estimar el error de aproximación:

```
Ve=@(x)interp1(xe,[0;ve;0],x,"spline");
>> norm(Ve(x)'-[0;v;0],inf)
ans =    8.5333e-07
>> plot(x,abs(Ve(x)'-[0;v;0]))
```



3.3.2. Ejercicios de Práctica

1. Calcular la discretización correspondiente y resolver el problema de deflexión de varillas implementando MDF en Octave.

$$\begin{cases} \partial_t^2 u - 4\partial_x^2 u = 0, (x,t) \in [0,1] \times [0,\infty) \\ u(0,t) = u(1,t) = 0 \\ u(x,0) = sen(\pi x) \\ \partial_t u(x,0) = 0 \end{cases}$$

2. Calcular la discretización correspondiente y resolver el problema de Helmholtz correspondiente al modelo de deflexión de varillas implementando MDF en Octave.

$$\begin{cases} \partial_t^2 u - 4\partial_x^2 u = sen(\pi x/4)e^{it}, (x,t) \in [0,1] \times [0,\infty) \\ u(0,t) = u(1,t) = 0 \\ u(x,0) = u_0(x) \\ \partial_t u(x,0) = v_0(x) \end{cases}$$

3. Calcular la discretización correspondiente y resolver el problema de deflexión **amortiguada/incrementada** de varillas implementando MDF en Octave.

$$\begin{cases} \partial_t^2 u + c\partial_t u - \alpha^2 \partial_x^2 u = 0, (x,t) \in [0,1] \times [0,\infty) \\ u(0,t) = u(1,t) = 0 \\ u(x,0) = Asen(\pi m x) \\ \partial_t u(x,0) = 0 \end{cases}$$

Idea: Reformular las ecuaciones dinámicas y la discretización del caso simple de la práctica 1, modificando convenientemente el archivo `*.m` correspondiente a la varilla simple de la práctica 1. Cuál es el efecto de un coeficiente c positivo/negativo/cero?

3.4. Método de Diferencias Finitas (MDF) en Dimensiones Superiores

3.4.1. Solución Numérica de Modelos Dinámicos

Método de Diferencias Finitas (MDF) en Dimensiones Superiores

MDF para Problemas de Valor Inicial y de Frontera: Estudiar la solución de las siguientes formas generales de ecuaciones diferenciales utilizando diferencias finitas, desarrollando un algoritmo computacional e implementando el mismo en Octave/SciLab.

PROBLEMA GENERAL DE MOVIMIENTO ONDULATORIO EN 2D: Utilizamos este tipo de modelos para simular la deflexión de **membranas** representativas de una estructura dada.

$$\begin{cases} \partial_t^2 u - \alpha^2(\partial_x^2 u + \partial_y^2 u) = f(x,t), (x,y,t) \in [0,1]^2 \times [0,\infty) \\ u(0,y,t) = u(1,y,t) = 0 \\ u(x,0,t) = u(x,0,t) = 0 \\ u(x,y,0) = u_0(x,y) \\ \partial_t u(x,y,0) = v_0(x,y) \end{cases}$$

SOLUCIÓN:

ALGORITMO DE SOLUCIÓN PARA EL PROBLEMA GENERAL DE MOVIMIENTO ONDULATORIO EN 1D:

1. Consideremos las fórmulas estudiadas en clase de la forma:

$$\partial_t u(x,y,t) \approx \frac{1}{h_t}(u(x,y,t+h) - u(x,y,t))$$

$$\partial_x^2 u(x,y,t) \approx \frac{1}{h^2}(u(x-h,y,t) - 2u(x,y,t) + u(x+h,y,t))$$

$$\partial_y^2 u(x,y,t) \approx \frac{1}{h^2}(u(x,y-h,t) - 2u(x,y,t) + u(x,y+h,t))$$

2. Consideremos las particiones de $[0,1] \times [0,T]$ definidas por:

$$0 = x_0 < x_1 < x_2 < \cdots < x_{N-1} < x_N = 1$$

$$0 = y_0 < y_1 < y_2 < \cdots < y_{N-1} < y_N = 1$$

$$0 = t_0 < x_1 < x_2 < \cdots < t_{M-1} < t_M = T$$

donde $x_k = kh$, $y_k = kh$, $t_j = jh_t$ para $h = 1/N$, $h_t \leq h^2/\alpha^2$ y $0 \leq k \leq N$.

3. Reduzcamos el orden del modelo diferencial a la forma:

$$\begin{cases} \partial_t u = v, \\ \partial_t v = \alpha^2(\partial_x^2 u + \partial_y^2 u) + f(x,t), (x,t) \in [0,1]^2 \times [0,\infty) \\ u(0,y,t) = u(1,y,t) = 0 \\ u(x,0,t) = u(x,0,t) = 0 \\ u(x,y,0) = u_0(x,y) \\ v(x,y,0) = v_0(x,y) \end{cases}$$

Llamamos a las ecuaciones resultantes de esta reducción, ecuaciones dinámicas representativas de primer orden, o solo ecuaciones dinámicas por simplicidad.

4. Definamos $u_{k,j,l} := u(x_k, y_j, t_l) = u(kh, jh, lh_t)$ y $v_{k,j,l} := v(x_k, y_j, t_j) = v(kh, jh, lh_t)$, $0 \leq k, j \leq N$, $1 \leq l \leq M$. Aplicando los pasos [1], [2] y [3] podemos obtener una formulación discreta del problema general de la forma:

$$\begin{cases} u_{k,j,l+1} = u_{k,j,l} + h_t v_{k,j,l} \\ v_{k,j,l+1} = v_{k,j,l} + \frac{h_t \alpha^2}{h^2}(u_{k-1,j,l} + u_{k,j-1,l} - 4u_{k,j,l} + u_{k+1,j,l} + u_{k,j+1,l}) + h_t f(x_k, y_j, t_l) \end{cases},$$

$1 \leq k, j \leq N-1$.

5. Podemos ahora utlizar el paso [4] para obtener una representación matricial de la discretización de la forma:

$$\begin{cases} \mathbf{u}_{j+1} = \mathbf{u}_j + h_t \mathbf{v}_j \\ \mathbf{v}_{j+1} = \mathbf{v}_j + h_t \alpha^2 \mathbf{L}_{2d} \mathbf{u}_j + h_t \mathbf{f}_j \end{cases}, 1 \leq k, j \leq N-1.$$

donde $\mathbf{u}_j = [u_{1,1,j}, u_{1,2,j}, \ldots, u_{N-1,N-1,j}]^\top$, $\mathbf{f}_j = [f(x_1, y_1, t_j), f(x_1, y_2, t_j), \ldots, f(x_{N-1}, y_{N-1}, t_j)]^\top$ y donde $\mathbf{L}_{2d}$ es la matriz de coeficientes dada por la ecuación:

$$\mathbf{L}_{2d} = \mathbf{L}_{1d} \otimes \mathbf{I}_{N-1} + \mathbf{I}_{N-1} \otimes \mathbf{L}_{1d}$$

donde $\mathbf{L}_{1d}$ es la matriz definida por la expresión:

$$\mathbf{L}_{1d} = \frac{1}{h^2} \begin{bmatrix} -2 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \cdots & \vdots \\ 0 & \cdots & 0 & 1 & -2 & 1 & 0 \\ 0 & \cdots & 0 & 0 & 1 & -2 & 1 \\ 0 & \cdots & 0 & 0 & 0 & 1 & -2 \end{bmatrix}$$

6. Resolvemos el sistema del paso [5] utilizando un procedimiento numérico de integración numérica en el tiempo.
7. Representamos las solución aproximada de la ecuacón en forma general en un gráfico dinámico.

PRÁCTICA 1.

Simular la deflexión de una membrana representativa descrita por una ecuación de la forma:

$$\begin{cases} \partial_t^2 u - \alpha^2(\partial_x^2 u + \partial_y^2 u) = 0, (x,y,t) \in [0,1] \times [0,\infty) \\ u(0,y,t) = u(1,y,t) = 0 \\ u(x,0,t) = u(x,1,t) = 0 \\ u(x,y,0) = Ae^{-k((x-1/2)^2+(y-1/2)^2)} sen(\pi m x) sen(\pi m y) \\ \partial_t u(x,y,0) = 0 \end{cases}$$

SOLUCIÓN: Implementaremos el algoritmo de solución utilizando Octave:

1. Creamos el archivo `*.m` cuyo código se presenta a continuación:

```
function [W,Wg,x,y]=membrana_simple(M,N,c2,m,k,A)
h=1/N;
ht=1/M;
x=0:h:1;
[x,y]=meshgrid(x);
L2=spdiags (ones(N-1,1)*[1 -2 1],-1:1,N-1,N-1);
L2=1/h^2*L2;
E=speye(N-1);
L2=kron(L2,E)+kron(E,L2);
u0=A*exp(-k*((x(2:N,2:N)-.5).^2+(y(2:N,2:N)-.5).^2)).*...
sin(pi*m*x(2:N,2:N)).*sin(pi*m*y(2:N,2:N));
v0=zeros(N-1,N-1);
u=u0(:);
v=v0(:);
W=zeros(N+1);
[xx,yy]=meshgrid(0:1/25:1);
Wg=@(xx,yy,u)interp2(x,y,u,xx,yy,"spline");
for j=1:M
u=u+ht*v;
v=v+ht*c2*L2*u;
W(2:N,2:N)=reshape(u,N-1,N-1);
colormap([0 0 0]);
mesh(xx,yy,Wg(xx,yy,W));
axis([0 1 0 1 -1 1]);
pause(.2);
endfor
endfunction
```

2. Realizamos un experimento computacional con los parámetros $N = 100, M = 400, A = 0{,}9, \alpha^2 = c^2 = 1$, $m = 1$ y $k = 20$.

```
[W,Wg,x,y]=membrana_simple(100,25,1,1,20,.9);
```

PROBLEMA GENERAL DE CÁLCULO DE MODOS DE VIBRACIÓN DE VARILLAS REPRESENTATIVAS (Representación de Helmholtz): Dado un problema de deflexión de varillas de la forma:

$$\begin{cases} \partial_t^2 u - \alpha^2(\partial_x^2 u + \partial_y^2 u) = f(x,y,t), (x,y,t) \in [0,1]^2 \times [0,\infty) \\ u(0,y,t) = u(1,y,t) = 0 \\ u(x,0,t) = u(x,0,t) = 0 \\ u(x,y,0) = u_0(x,y) \\ \partial_t u(x,y,0) = v_0(x,y) \end{cases}$$

Hacemos las suposiciones $u(x,y,t) = v(x,y)e^{i\omega t}$ y $f(x,y,t) = g(x,y)e^{i\omega t}$, donde $i = \sqrt{-1}$. Bajo condiciones físicamente apropiadas y realizables tenemos que las ecuaciones anteiores producen una expresión de la forma:

$$\begin{cases} e^{i\omega t}(i\omega)^2 v(x,y) - e^{i\omega t}\alpha^2(\partial_x^2 v(x,y) + \partial_y^2 v(x,y)) = g(x,y)e^{i\omega t}, (x,t) \in [0,1] \times [0,\infty) \\ v(0,y)e^{i\omega t} = v(1,y)e^{i\omega t} = 0 \\ v(x,0)e^{i\omega t} = v(x,1)e^{i\omega t} = 0 \end{cases}$$

Luego de simplificar la expresión anterior obtenemos el siguiente resultado:

$$\begin{cases} -\alpha^2(\partial_x^2 v(x,y) + \partial_y^2 v(x,y)) - \omega^2 v(x,y) = g(x,y), x \in [0,1] \\ v(0,y) = v(1,y) = 0 \\ v(x,0) = v(x,1) = 0 \end{cases}$$

La expresión anterior se denomina representación de Helmholtz del problema de deflexión, y corresponde a un problema de valor de frontera que podemos resolver con el MDF utilizando el siguiente algoritmo:

SOLUCIÓN:

El procedimiento de solución es el siguiente:

1. Calcular la representación de Helmholtz del problema de deflexión para obetener la expresión:

$$\begin{cases} -\alpha^2(\partial_x^2 v(x,y) + \partial_y^2 v(x,y)) - \omega^2 v(x,y) = g(x,y), x \in [0,1]^2 \\ v(0,y) = v(1,y) = 0 \\ v(x,0) = v(x,1) = 0 \end{cases}$$

2. Discretizar la expresión del paso [1] utilizando las técnicas de las lecturas de la semana 1 para obtener:

$$-\frac{\alpha^2}{h^2}(v_{k-1,j} + v_{k,j-1} - 4v_{k,j} + v_{k+1,j} + v_{k,j+1}) - \omega^2 v_{k,j} = g(x_k, y_j), 1 \leq k, j \leq N-1,$$

donde $x_k = kh$, $y_j = jh$, $h = 1/N$, para $N \in \mathbb{Z}^+$.

3. Calcular la representación matricial de la discretización del paso [2] de la forma:

$$-\alpha^2 \mathbf{L}_{2d}\mathbf{v} - \omega^2 \mathbf{v} = \mathbf{g}$$

donde $\mathbf{L}_{2d}$ está determinada por la expresión

$$\mathbf{L}_{2d} = \mathbf{L}_{1d} \otimes \mathbf{I}_{N-1} + \mathbf{I}_{N-1} \otimes \mathbf{L}_{1d}$$

con $\mathbf{L}_{1d}$ definida por:

$$\mathbf{L}_{1d} = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \cdots & \vdots \\ 0 & \cdots & 0 & 1 & -2 & 1 & 0 \\ 0 & \cdots & 0 & 0 & 1 & -2 & 1 \\ 0 & \cdots & 0 & 0 & 0 & 1 & -2 \end{bmatrix}$$

y donde $\mathbf{v} = [v_{1,1}, v_{1,2}, \ldots, v_{N-1,N-1}]^\top$, $\mathbf{g} = [g(x_1, y_1), g(x_1, y_2), \ldots, g(x_{N-1}, y_{N-1})]^\top$.

4. Resolver el problema de álgebra lineal del paso [3].

PRÁCTICA 2.

Utilizar el MDF implementado en Octave para resolver numéricamente el problema de Helmholtz correspondiente a la ecuación:

$$\begin{cases} \partial_t^2 u - (\partial_x^2 u + \partial_y^2 u) = 0, (x,t) \in [0,1]^2 \times [0,\infty) \\ u(0,y,t) = u(1,y,t) = 0 \\ u(x,0,t) = u(x,1,t) = 0 \\ u(x,y,0) = u_0(x,y) \\ \partial_t u(x,y,0) = v_0(x,y) \end{cases}$$

SOLUCIÓN:

1. La representación de Helmholtz del problema esta dada por:

$$\begin{cases} -(\partial_x^2 v(x,y) + \partial_y^2 v(x,y)) - \omega^2 v(x,y) = 0, (x,y) \in [0,1]^2 \\ v(0,y) = v(1,y) = 0 \\ v(x,0) = v(x,1) = 0 \end{cases}$$

2. La expresión del paso [2] tiene una discretización de la forma:

$$-\frac{\pi^2}{h^2}(v_{k-1,j} + v_{k,j-1} - 4v_{k,j} + v_{k+1,j} + v_{k,j+1}) = \omega^2 v_{k,j}, 1 \leq k \leq N-1,$$

donde $x_k = kh$, $y_j = jh$, $h = 1/N$, para $N \in \mathbb{Z}^+$.

3. La forma matricial de la discretización del paso [2] es:

$$-\mathbf{L}_{2d}\mathbf{v} = \omega^2 \mathbf{v}$$

donde $\mathbf{L}_{2d}$ está determinada por la expresión

$$\mathbf{L}_{2d} = \mathbf{L}_{1d} \otimes \mathbf{I}_{N-1} + \mathbf{I}_{N-1} \otimes \mathbf{L}_{1d}$$

con $\mathbf{L}_{1d}$ definida por:

$$\mathbf{L}_{1d} = \frac{1}{h^2} \begin{bmatrix} -2 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \cdots & \vdots \\ 0 & \cdots & 0 & 1 & -2 & 1 & 0 \\ 0 & \cdots & 0 & 0 & 1 & -2 & 1 \\ 0 & \cdots & 0 & 0 & 0 & 1 & -2 \end{bmatrix}$$

y donde $\mathbf{v} = [v_{1,1}, v_{1,2}, \ldots, v_{N-1,N-1}]^\top$.

4. Podemos resolver el problema de valores propios correspondiente numéricamente, utilizando los pasos anteriores y el siguiente procedimiento en Octave:

   4.1 Calculamos el mallado de $[0, 1]$:

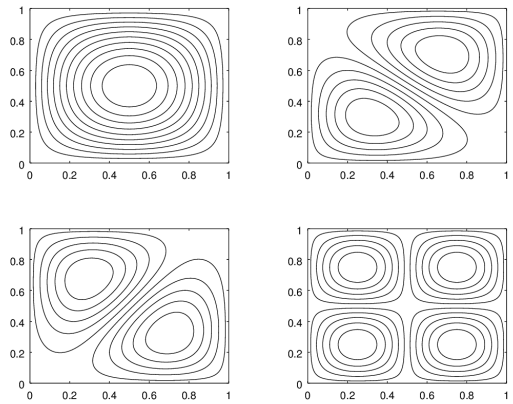
```
>> N=100;
>> h=1/N;
>> x=0:h:1;
>> [x,y]=meshgrid (x);
```

   4.2 Calculamos la representación matricial de la discretización:

```
>> alfa2=1;
>> L1d=spdiags (ones(N-1,1)*[1 -2 1],-1:1,N-1,N-1);
>> L1d=1/h^2*L1d;
>> E=speye(N-1);
>> L2d=kron(E,L1d)+kron(L1d,E);
>> L2d=-alfa2*L2d;
```

   4.3 Reolvemos el problema matricial del paso [4.2] y graficamos la solución numérica de los cuatro modos de vibración básicos:

```
>> [v,l]=eigs(L2d,4,0);
>> colormap([0 0 0]);
>> W=zeros(N+1);
>> for k=1:4, W(2:N,2:N)=reshape (v(:,k),N-1,N-1);...
subplot(2,2,k);contour(x,y,W);end
```



3.4.2. Ejercicios de Práctica

1. Calcular la discretización correspondiente y resolver el problema de deflexión de membranas implementando MDF en Octave.

$$\begin{cases} \partial_t^2 u - 4(\partial_x^2 u + \partial_y^2 u) = 0, (x, y, t) \in [0, 1]^2 \times [0, \infty) \\ u(0, y, t) = u(1, y, t) = 0 \\ u(x, 0, t) = u(x, 1, t) = 0 \\ u(x, y, 0) = sen(\pi m x) sen(\pi m y) \\ \partial_t u(x, y, 0) = 0 \end{cases}$$

2. Calcular la discretización correspondiente y resolver el problema de Helmholtz correspondiente al modelo de deflexión de varillas implementando MDF en Octave.

$$\begin{cases} \partial_t^2 u - 4(\partial_x^2 u + \partial_y^2 u) = 0, (x, y, t) \in [0, 1]^2 \times [0, \infty) \\ u(0, y, t) = u(1, y, t) = 0 \\ u(x, 0, t) = u(x, 1, t) = 0 \\ u(x, y, 0) = u_0(x, y) \\ \partial_t u(x, y, 0) = v_0(x, y) \end{cases}$$

3. Calcular la discretización correspondiente y resolver el problema de deflexión **amortiguada/incrementada** de varillas implementando MDF en Octave.

$$\begin{cases} \partial_t^2 u + c\partial_t u - 4(\partial_x^2 u + \partial_y^2 u) = 0, (x, y, t) \in [0, 1]^2 \times [0, \infty) \\ u(0, y, t) = u(1, y, t) = 0 \\ u(x, 0, t) = u(x, 1, t) = 0 \\ u(x, y, 0) = sen(\pi m x) sen(\pi m y) \\ \partial_t u(x, y, 0) = 0 \end{cases}$$

Idea: Reformular las ecuaciones dinámicas y la discretización del caso simple de la práctica 1, modificando convenientemente el archivo `*.m` correspondiente a la varilla simple de la práctica 1. Cuál es el efecto de un coeficiente c positivo/negativo/cero?

Capítulo 4

Elementos de Cómputo de Modelos Lineales Aproximantes

4.1. Principios de Modelos Lineales Aproximantes Estáticos

4.1.1. Nociones Modelos Lineales Aproximates Estáticos

Definición 4.1.1. Dadas $X_j \in \mathbb{C}^{n_j \times r}$ para $j = 1, \ldots, n$, en este curso se escribirá $\mathrm{col}(X_1, \ldots, X_n)$ para denotar la operación determinada por la siguiente expresión.

$$\mathrm{col}(X_1, \ldots, X_n) := \begin{bmatrix} X_1 \\ \vdots \\ X_n \end{bmatrix}$$

Definición 4.1.2. Dado un conjunto de datos $\mathscr{D}$ correspondientes a resultados de un experimento $\mathbb{E}$, en un conjunto universal $\mathscr{U}$ de posibles resultados. Se denomina modelo aproximante lineal estático correspondiente al experimento $\mathscr{E}$ al conjunto $\mathscr{M}_{\mathbb{E}}$ determinado por la siguiente expresión.

$$\mathscr{M}_{\mathbb{E}} := \{d \in \mathscr{U} : f_E(d) = 0\} \tag{4.1.1}$$

donde $f_E \in (\mathbb{C}^m)^{\mathscr{U}}$ (f_E es una función de $\mathscr{U}$ a $\mathbb{C}^m$ para algún entero positivo m). La expresión (4.1.1) recibirá el nombre de representación en este curso.

Notación 4.1.3. Cuando el experimento $\mathbb{E}$ al que hace referencia un modelo $\mathscr{M}_{\mathbb{E}}$ es claro en el contexto de un problema, se omitirá la referencia explícita a $\mathbb{E}$ en el modelo y se escribirá solamente $\mathscr{M}$ en lugar de $\mathscr{M}_{\mathbb{E}}$.

4.2. Modelos Lineales Aproximantes Estáticos de la Forma: $Y = AX$

Dados $X \in \mathbb{C}^{n\times r}$ e $Y \in \mathbb{C}^{n\times r}$ es claro que los modelos de la forma $Y = AX$ basados en datos $\mathscr{D}_N := \{(Y_j, X_j)\}_{j=1}^N \subset \mathscr{U}$ en un universo de resultados $\mathscr{U}$ puede ser representado en términos de datos en un universo $\hat{\mathscr{U}}$ relacionado con $\mathscr{U}$ a través de la relación $\mathrm{col}(Y, X) \in \hat{\mathscr{U}} \Leftrightarrow (Y, X) \in \mathscr{U}$, y de la siguiente representación.

$$\mathscr{M}_{AX} := \left\{\mathrm{col}(Y, X) \in \hat{\mathscr{U}} : \begin{bmatrix} I & -A \end{bmatrix} \mathrm{col}(Y, X) = 0\right\} \tag{4.2.1}$$

4.2.1. Mínimos cuadrados y cómputo de modelos lineales estáticos aproximantes de la forma: $Y = AX$

El cómputo de modelos *exactos* $\mathscr{M}_{AX}$ de la forma (4.7.1) es poco realista, en lugar de una representación exacta de la forma (4.7.1), en la práctica se consideran representaciones alternativas aproximadas basadas en una muestra $\hat{\mathscr{D}}_N = \{\text{col}(Y_j, X_j)\} \subset \mathscr{U}$ de la forma.

$$\tilde{\mathscr{M}}_{AX} := \left\{ A \in \mathbb{C}^{n\times m} : A = \arg \min_{\hat{A}\in\mathbb{C}^{n\times m}} \sum_{j=1}^{N} \left\| \begin{bmatrix} I & -\hat{A} \end{bmatrix} \text{col}(Y_j, X_j) \right\|_2^2 \right\} \tag{4.2.2}$$

Un razonamiento similar al presentado en la sección [4, §Matrix least squares (pág. 223)] permite obtener el siguiente lema técnico.

Lema 4.2.1. *Dada una muestra $\hat{\mathscr{D}}_N = \{(Y_j, X_j)\}_{j=1}^N \subset \mathscr{U}$ en un universo de resultados $\mathscr{U}$ como el considerado previamente. Si se define $Y = \begin{bmatrix} Y_1 & \cdots & Y_N \end{bmatrix}$ y $X = \begin{bmatrix} X_1 & \cdots & X_N \end{bmatrix}$, entonces $A \in \tilde{\mathscr{M}}_{AX}$, si y solo si $A = YX^+$.*

Demostración. Ejercicio para el lector. □

Ejercicio Resuelto 1. Sea $A \in \mathbb{R}^{3\times3}$ la matriz definida por la expresión:

$$A = \begin{bmatrix} -2 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -2 \end{bmatrix}$$

Dada una matriz de datos $X \in \mathbb{R}^{3\times3}$ cuyas columnas han sido generadas al azar verificar (de forma aproximada) computacionalmente el lema 4.5.1 con respecto al vector de datos $Y = AX$.

Solución. Para realizar esta verificación utilizaremos Octave. Es posible ingresar/generar $A, X, Y \in \mathbb{R}^{3\times3}$ utilizando las siguientes secuencias de comandos.

```
>> A=[-2 1 0;1 -2 1;0 1 -2];
>> X=randn(3);
>> Y=A*X;
```

La primer parte de la verificación puede realizarse calculando una matriz $A_1 \in \tilde{\mathscr{M}}_{AX}$ implementando directamente la definición a través programación cuadrática secuencia, utilizando de la siguiente secuencia de comandos:

```
>> phi=@(a)norm(Y-reshape(a,3,3)*X,'fro')^2;
>> [A1,obj,info,iter,nf,lambda]=sqp(zeros(9,1),phi,[],...
> [],[],[],[],1e-12);
>> A1=reshape (A1,3,3)
A1 =

  -1.9999999975692   0.9999999763382   0.0000000408414
   1.0000000020822  -2.0000000226371   1.0000000385234
   0.0000000021409   0.9999999771977  -1.9999999611062
```

Ahora calcularemos $A_1 \in \tilde{\mathscr{M}}_{AX}$ aplicando el lema 4.5.1 a través de la siguiente secuencia de comandos.

```
>> A2=Y*pinv(X)
A2 =

  -2.0000e+00   1.0000e+00  -4.4409e-16
   1.0000e+00  -2.0000e+00   1.0000e+00
  -6.1062e-16   1.0000e+00  -2.0000e+00
```

4.3. Modelos Lineales Aproximantes Estáticos de la Forma: $Y = AX+B$

Dados $X \in \mathbb{C}^{m\times r}$ e $Y \in \mathbb{C}^{n\times r}$ es claro que los modelos de la forma $Y = AX + B$ basados en datos $\mathscr{D}_N := \{(Y_j, X_j)\}_{j=1}^N \subset \mathscr{U}$ en un universo de resultados $\mathscr{U}$ pueden ser representados en términos de datos en un universo $\hat{\mathscr{U}}$ relacionado con $\mathscr{U}$ a través de la relación $\mathrm{col}(Y, X, I_B) \in \hat{\mathscr{U}} \Leftrightarrow (Y, X) \in \mathscr{U}$, y de la siguiente representación,

$$\mathscr{M}_{(A|B)X} := \left\{ \mathrm{col}(Y, X) \in \hat{\mathscr{U}} : \begin{bmatrix} I & -A & B \end{bmatrix} \mathrm{col}(Y, X, I_B) = 0 \right\} \tag{4.3.1}$$

donde I_B es determinada de tal forma que $BI_B = B$. Por simplicidad, en este estudio consideraremos la matriz I_B igual a la matriz identidad de $r \times r$ donde r es el entero correspondiente al número de columnas de B, X e Y.

4.3.1. Mínimos cuadrados y cómputo de modelos lineales estáticos aproximantes de la forma: $Y = AX + B$

Para estudiar la formulación alternativa aproximada de las estrategias de cómputo de modelos *exactos* $\mathscr{M}_{(A|B)X}$ de la forma (4.7.1), en lugar de una representación exacta de la forma (4.7.1), iniciaremos considerando representaciones alternativas aproximadas basadas en una muestra $\hat{\mathscr{D}}_N = \{\mathrm{col}(Y_j, X_j)\} \subset \mathscr{U}$ de problemas de la forma.

$$\tilde{\mathscr{M}}_{(A|B)X} := \left\{ (A, B) : (A, B) = \arg \min_{(\hat{A},\hat{B})\in\mathbb{C}^{n\times m}\times\mathbb{C}^{n\times r}} \sum_{j=1}^{N} \left\| \begin{bmatrix} I & -\hat{A} & -\hat{B} \end{bmatrix} \begin{bmatrix} Y_j \\ X_j \\ I_B \end{bmatrix} \right\|_2^2 \right\} \tag{4.3.2}$$

Una variación del razonamiento presentado en la sección [4, §Matrix least squares (pág. 223)] permite obtener el siguiente lema técnico.

Lema 4.3.1. *Dada una muestra $\hat{\mathscr{D}}_N = \{(Y_j, X_j)\}_{j=1}^N \subset \mathscr{U}$ en un universo de resultados $\mathscr{U}$ como el considerado previamente. Sea $r \in \mathbb{Z}$ el número de columnas de cada X_j. Si se define $Y = \begin{bmatrix} Y_1 & \cdots & Y_N \end{bmatrix}$ y $X = \begin{bmatrix} \hat{X}_1 & \cdots & \hat{X}_N \end{bmatrix}$ donde $\hat{X}_j = \mathrm{col}(X_j, I_r)$, entonces $(A, B) \in \tilde{\mathscr{M}}_{AX}$, si y solo si $\begin{bmatrix} A|B \end{bmatrix} = YX^+$, donde $[A|B]$ denota la matriz aumentada determinada por las matrices A, B.*

Ejercicio para el lector 4.3.1. Demostrar el lema 4.5.1.

Ejercicio Resuelto 2. Sean $A \in \mathbb{R}^{3\times3}$ y $B \in \mathbb{R}^{3\times2}$ las matrices definidas por las expresiones:

$$A = \begin{bmatrix} -2 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -2 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 1 \\ 3 & -1 \\ -2 & 2 \end{bmatrix}$$

Si se considera el modelo lineal $Y = AX + B$, para $X \in \mathbb{R}^{3\times 2}$. Dada una muestra $\{(Y_j, X_j) \in \mathbb{R}^{3\times 2} \times \mathbb{R}^{3\times 2}\}_{j=1}^{3}$ donde las matrices $X_j \in \mathbb{R}^{3\times 2}$ son generadas al azar y donde $Y_j = AX_j + B$, aproximar computacionalmente el par $(\tilde{A}, \tilde{B}) \in \tilde{\mathscr{M}}_{(A|B)X}$ aplicando el lema 4.5.1 con respecto la muestra $\{(Y_j, X_j)\}_{j=1}^{3}\mathbb{R}^{3\times 2} \times \mathbb{R}^{3\times 2}\}_{j=1}^{3}$.

Solución. Para realizar esta cómputo utilizaremos Octave. Es posible ingresar/generar $A, X, Y \in \mathbb{R}^{3\times 3}$ utilizando las siguientes secuencias de comandos.

```
>> A=[-2 1 0;1 -2 1;0 1 -2];
>> B=[1 3 -2;1 -1 2].';
>> X=randn(3,6);
>> Y=A*X+repmat(B,1,3);
>> X=[X;repmat(eye(2),1,3)];
```

Ahora es posible calcular $(A_1, B_1) \in \tilde{\mathscr{M}}_{(A|B)X}$ aplicando el lema 4.5.1 a través de la siguiente secuencia de comandos.

```
>> AB1=Y/X;
>> A1=AB1(:,1:3)
A1 =

  -2.0000e+00   1.0000e+00   1.5821e-15
   1.0000e+00  -2.0000e+00   1.0000e+00
   3.1191e-15   1.0000e+00  -2.0000e+00

>> B1=AB1(:,4:5)
B1 =

   1.00000   1.00000
   3.00000  -1.00000
  -2.00000   2.00000
```

4.4. Descomposiciones en Valores Singulares en Cómputo Aproximado de Modelos

Si se considera el problema

$$x_{LS} = \arg \min_{x\in\mathbb{R}^n} \|Ax - b\|_2 \tag{4.4.1}$$

para A, b dados. La descomposición en valores singulares (SVD) permite calcular la solución x_{LS} del problema (4.4.1) de forma bastance elegante y eficiente, tal como se ilustra en el siguiente teorema.

Teorema 4.4.1. *[16, Teorema 5.5.1]. Suponiendo que $U^\top AV = \Sigma$ es la SVD de $A \in \mathbb{R}^{m\times n}$ con $r = \mathrm{rk}(A)$. Si $U = [u_1, \ldots, u_m]$ y $V = [v_1, \ldots, v_n]$ son particionamientos de columnnas y $b \in \mathbb{R}^m$, entonces*

$$x_{LS} = \sum_{j=1}^{r} \frac{u_j^\top b}{\sigma_j} v_j \tag{4.4.2}$$

minimiza $\|Ax - b\|_2$ y tiene la norma-2 más pequeña de todos los minimizadores. Además

$$\|Ax_{LS} - b\|_2^2 = \sum_{j=r+1}^{m} (u_j^\top b)^2. \tag{4.4.3}$$

Demostración. La demostración está disponible en [16, §5.5.3]. □

4.4.1. Pseudoinversas y SVD

Con base en el teorema 4.4.1 es posible observar que la pseudoinversa $A^+ \in \mathbb{R}^{n\times m}$ puede representarse de forma alternativa a través de la expresión

$$A^+ = V\Sigma^+U^\top \tag{4.4.4}$$

donde

$$\Sigma^+ = \operatorname{diag}\left(\frac{1}{\sigma_1},\ldots,\frac{1}{\sigma_r},0,\ldots,0\right) \in \mathbb{R}^{n\times m}, \quad r = \operatorname{rk}(A)$$

con base en (4.4.4), (4.4.2) y (4.4.3), es posible confirmar que $x_{LS} = A^+b$ y que $\|Ax_{LS} - b\|_2 = \|(I - AA^+)b\|_2$

Ejercicio para el lector 4.4.1. Reproducir el experimento correspondiente al **ejercicio resuelto** 2 recalculando los elementos $(A, B) \in \tilde{\mathscr{M}}_{(A|B)X}$ aplicando eficientemente el teorema 4.4.1 y la identidad (4.4.4).

4.5. Modelos Lineales Aproximantes Estáticos de la Forma: $Y = UX$, $U^*U = I$

Dados $X \in \mathbb{C}^{n\times r}$ e $Y \in \mathbb{C}^{n\times r}$ es claro que los modelos de la forma $Y = UX, U^*U = I$ basados en datos $\mathscr{D}_N := \{(Y_j, X_j)\}_{j=1}^N \subset \mathscr{U}$ en un universo de resultados $\mathscr{U}$ puede ser representado en términos de datos en un universo $\hat{\mathscr{U}}$ relacionado con $\mathscr{U}$ a través de la relación $\operatorname{col}(Y, X) \in \hat{\mathscr{U}} \Leftrightarrow (Y, X) \in \mathscr{U}$, y de la siguiente representación.

$$\mathscr{M}_{UX} := \left\{\operatorname{col}(Y,X) \in \hat{\mathscr{U}} : \begin{bmatrix} I & -U\end{bmatrix}\operatorname{col}(Y,X) = 0,\ U^*U = I\right\} \tag{4.5.1}$$

4.5.1. Mínimos cuadrados y cómputo de modelos lineales estáticos aproximantes de la forma: $Y = UX$, $U^*U = I$

El cómputo de modelos *exactos* $\mathscr{M}_{UX}$ de la forma (4.7.1) es poco realista, en lugar de una representación exacta de la forma (4.7.1), en la práctica se consideran representaciones alternativas aproximadas basadas en una muestra $\hat{\mathscr{D}}_N = \{\operatorname{col}(Y_j, X_j)\} \subset \mathscr{U}$ de la forma.

$$\tilde{\mathscr{M}}_{UX} := \left\{ U \in \mathbb{C}^{n\times m} : U = \arg \min_{\substack{\hat{U}\in\mathbb{C}^{n\times m}\\ \hat{U}^*\hat{U}=I}} \sum_{j=1}^{N} \left\| \begin{bmatrix} I & -\hat{U}\end{bmatrix}\operatorname{col}(Y_j,X_j)\right\|_2^2 \right\} \tag{4.5.2}$$

Un razonamiento similar al presentado en la sección [4, §Matrix least squares (pág. 223)] permite obtener el siguiente lema técnico.

Lema 4.5.1. *Dada una muestra $\hat{\mathscr{D}}_N = \{(Y_j, X_j)\}_{j=1}^N \subset \mathscr{U}$ en un universo de resultados $\mathscr{U}$ como el considerado previamente. Si se define $Y = \begin{bmatrix} Y_1 & \cdots & Y_N\end{bmatrix}$ y $X = \begin{bmatrix} X_1 & \cdots & X_N\end{bmatrix}$, entonces $U \in \tilde{\mathscr{M}}_{UX}$, si y solo si $U = WV^*$ para $W\Sigma V^* = YX^+$.*

Demostración. Ejercicio para el lector.

Idea: Aplicar ideas implemantadas en el caso de modelos $\mathscr{M}_{AX}$ (sin restricciones adicionales para A), junto con una variación para matrices complejas de las técnicas presentadas en [16] para resolver el problema *orthogonal Procrustes problem* determinado por la expresión.

$$Q = \arg \min_{\substack{\hat{Q} \in \mathbb{R}^{n\times m} \\ \hat{Q}^\top \hat{Q} = I}} \|A - B\hat{Q}\|_F$$

□

Ejemplo 8. **Compresión de una señal con ruido aleatorio.** Considerando una muestra de una señal de la forma $\{y_t\}_{t=1}^{101}$ donde $y_t = \text{sen}(2\pi(t-1)/100)+r_t$ para r_t un parámetro de simulación de ruido pseudoaleatorio. Una muestra de esta forma puede simularse en Octave con la siguiente secuencia de comandos.

```
>> t=0:1/100:1;
>> y=sin(2*pi*t)+.5e-1*randn(1,101);
>> plot(t,y,'k.-')
```

la representación gráfica de la muestra de la señal se ilustra en la fig. 4.1.

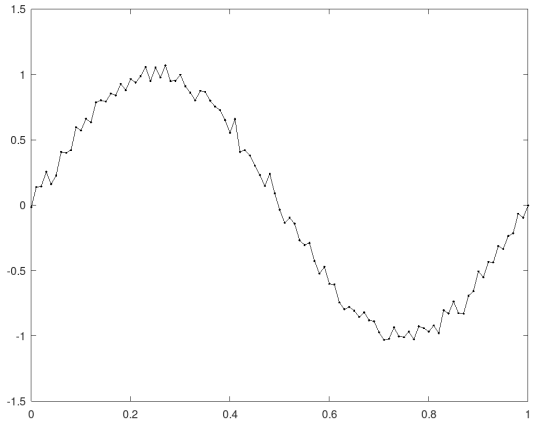


Figura 4.1: Representación gráfica de la muestra de la señal.

La transformada disceta de Fourier de la señal puede calcularse utilizando la siguiente secuencua de comandos.

```
>> Y=fft(y)/sqrt(101);
```

Si consideramos los elemenentos en el espectro de la señal correspondiente al umbral $\mathcal{U} = \{1 \leq t \leq 101 : |Y_t| > 0{,}2\}$. Es posible aplicar Octave para calcular el umbral y extraer la representación comprimida $\tilde{y}_t = \sum_{t\in\mathcal{U}} Y_t F_t$ de la señal, donde F_t es la t-ésima columna de la matriz de transformación discreta de Fourier correspondiente.

```
>> f=find(abs(Y)>.2);
>> Yr=Y(f).';
```

```
>> W=dftmtx (101)/sqrt(101);
>> W=W(f,:)';
>> plot(t,W*Yr,'k.-',t,y,'r.-')
```

la representación gráfica de la compresión de la señal se ilustra en la fig. 4.2.

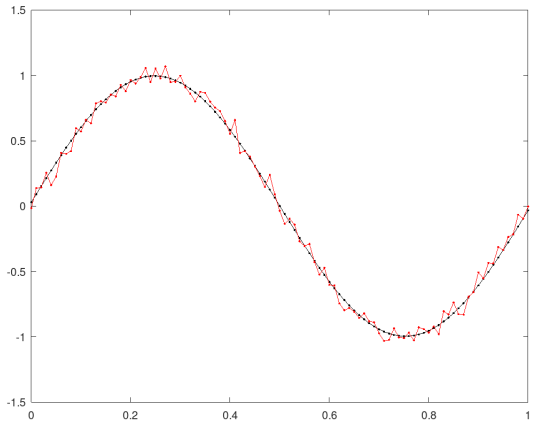


Figura 4.2: Representación gráfica de compresión de la muestra de la señal.

La siguiente secuencia de comandos de Octave permite calcular el error $\|W^*W - I\|_F$.

```
>> norm(W'*W-eye(size(W,2)),'fro')
ans =    3.1404e-16
```

4.6. Estudio de Caso

Considerando el modelo genérico $\mathscr{M}_{UX}$, remitir una propuesta de demostración del lema 4.5.1, o una implementación del modelo genérico en la solución de un problema de aplicación, por ejemplo en compresión de señales, desarrollando un algoritmo/programa que permita identificar la matriz W en el modelo $\mathscr{M}_{Uz}$, $z \in \mathbb{C}^{r\times 1}$ correspondiente al ejemplo 8.

4.7. Modelos Lineales Aproximantes Estáticos de la Forma: $AY = BX + C$

Dados $X \in \mathbb{C}^{n\times r}$ e $Y \in \mathbb{C}^{n\times r}$ es claro que los modelos de la forma $AY = BX + C$, basados en datos $\mathscr{D}_N := \{(Y_j, X_j)\}_{j=1}^N \subset \mathscr{U}$ en un universo de resultados $\mathscr{U}$ puede ser representado en términos de datos en un universo $\hat{\mathscr{U}}$ relacionado con $\mathscr{U}$ a través de la relación $\mathrm{col}(Y, X) \in \hat{\mathscr{U}} \Leftrightarrow (Y, X) \in \mathscr{U}$, y de la siguiente representación.

$$\mathscr{M}_{A|B|C} := \left\{ \mathrm{col}(Y, X) \in \hat{\mathscr{U}} : \begin{bmatrix} A & -B & -C \end{bmatrix} \mathrm{col}(Y, X, I_r) = 0 \right\} \tag{4.7.1}$$

donde r es el número de columnas de C.

4.7.1. Mínimos cuadrados y cómputo de modelos lineales estáticos aproximantes de la forma: $AY = BX + C$

Para estudiar una formulación alternativa aproximada de las estrategias de cómputo de modelos *exactos* $\mathscr{M}_{A|B|C}$ de la forma (4.7.1), en lugar de una representación exacta de la forma (4.7.1), iniciaremos considerando representaciones alternativas aproximadas basadas en una muestra $\hat{\mathscr{D}}_N = \{\text{col}(Y_j, X_j)\} \subset \hat{\mathscr{U}}$ de problemas de la forma.

$$\tilde{\mathscr{M}}_{A|B|C} := \left\{ (A,B,C) : (A,B,C) = \arg \min_{(\hat{A},\hat{B},\hat{C}) \in \mathbb{C}^{n\times m}\times(\mathbb{C}^{n\times r})^2} \sum_{j=1}^{N} \left\| \begin{bmatrix} \hat{A} \\ -\hat{B} \\ -\hat{C} \end{bmatrix}^{\top} \begin{bmatrix} Y_j \\ X_j \\ I_r \end{bmatrix} \right\|_F^2 \right\} \tag{4.7.2}$$

4.8. Ejercicios

1. Formular y demostrar un lema técnico que describa las solubilidad computabilidad de los elementos (representaciones matriciales de modelos) en el conjunto determinado por la expresión (4.7.2).
2. Diseñar un algoritmo que permita calcular los elementos cuya existencia está determinada por el lema desarrollado al resolver el problema previo.
3. Escribir un programa Octave que genere datos significativos correspondientes a un modelo de la forma $\mathscr{M}_{A|B|C}$ e identifique el modelo correspondiente a los datos generados.

4.9. Principios de Homotopías y Cómputo de Índices de Bucles en $\mathbb{C}\backslash\{0\}$

4.9.1. Principios de Homotopías

Notación 4.9.1. Dados dos conjuntos X, Y escribiremos Y^X para denotar el sub-conjunto de $X \times Y$ determinado por el conjunto de todas las funciones de X a Y.

Notación 4.9.2. Dados dos espacios métricos (topológicos) se denota por $C(X,Y)$ el conjunto de funciones continuas de X a Y.

Definición 4.9.3. Un espacio métrico (topológico) X se denomina conexo por trayectorias o CPT si para cada $x, x' \in X$, existe $\gamma \in C([0,1], X)$ tal que $\gamma(0) = x$ y $\gamma(1) = x'$.

Definición 4.9.4. Dados dos espacios topológicos X, Y y dadas $f_0, f_1 \in C(X,Y)$, una **homotopía** de f_0 a f_1 es una familia (**net**) de funciones $\{\hat{f}_t\}_{t\in[0,1]} \subset C(X,Y)$ tal que $\hat{f}_0 = f_0$ y $\hat{f}_1 = f_1$.

Observación 4.9.5. Con base en la definición previa es posible observar que dados dos espacios topolóficos, una homotopía entre dos mapas $f, g \in C(X,Y)$ puede interpretarse como una función $h \in C(X \times [0,1], Y)$ tal que.

$$\begin{cases} h(x,s) \in Y,\ s \in [0,1] \\ h(x,0) = f(x) \\ h(x,1) = g(x) \end{cases} , \ x \in X \tag{4.9.1}$$

Homotopías entre mapas arbitrarios f, g que solamente complen la condición (4.9.1) se denominan **homotopías libres**, dado que las únicas restricciones están dadas por la continuidad de $h : X \times [0,1] \to Y$ y por (4.9.1).

Suposición 4.9.6. Dada la importancia del espacio topológico $[0,1] \subset \mathbb{R}^1$ en el trabajo de clasificación de espacios topológicos correspondiente a este curso, a partir de este punto se asumirá cierto (sin necesidad de verificación/demostración) que $[0,1]$ **es un sub-conjunto compacto del espacio topológico** $\mathbb{R}^1$ (con respecto a la topología métrica usual inducida por la métrica $d : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ determinada por $d(x,y) = |x-y|, x, y \in R$).

En el caso de mapas (trayectorias) en $C([0,1], X)$ en un espacio topológico X, además de homotopías libres, se considerarán homotopías con algunas restricciones adicionales. Un ejemplo gráfico de homotopía libre entre dos trayectorias en $\mathbb{C}\backslash\{0\}$ se ilustra en la fig. 4.3.

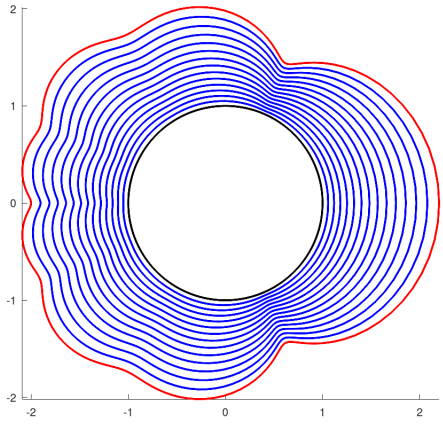


Figura 4.3: Ilustración gráfica de una homomtopía libre $\{f_s\}_{s\in[0,1]} \subset C([0,1], \mathbb{C}\backslash\{0\})$ en $\mathbb{C}\backslash\{0\}$ entre dos trayectorias $\gamma : [0,1] \to \mathscr{S}$ (representada por curva coloreada en negro) y $\gamma_p = p \circ \gamma : [0,1] \to p(\mathscr{S})$ (representada por curva coloreada en rojo), para $\mathscr{S} = \{z \in \mathbb{C} : |z| = 1\} \subset \mathbb{C}\backslash\{0\}$, $\gamma(t) = \exp(2\pi i t)$ para $t \in [0,1]$, $p \in \mathbb{C}[z]$ definido por $p(z) = \frac{1}{10}z^8 + \frac{1}{10}z^7 + \frac{1}{10}z^6 + \frac{1}{10}z^5 + \frac{1}{10}z^4 + \frac{1}{10}z^3 + \frac{1}{2}z^2 + \frac{9}{5}z - \frac{7}{10}$ y $f_s(t) = \frac{\gamma_p(t)}{|\gamma_p(t)|^s}$, $0 \leq s, t \leq 1$.

Observación 4.9.7. Es importante observar que la expresión$f_s(t) = \frac{\gamma_p(t)}{|\gamma_p(t)|^s}$, $0 \leq s, t \leq 1$ define una homotopía dado que $\gamma([0,1]), \gamma_p([0,1]) \subset \mathbb{C}\backslash\{0\}$ de manera que $0 \notin \gamma([0,1]), \gamma_p([0,1])$, si no se considera esta restricción para los rangos de γ, γ_p la expresión $f_t(s)$ podría no estar bien definida y no podría definir una homotopía. Esta es una de las razones por las que el conjunto en el que se está considerando la homotopía es importante.

4.9.2. Trayectorias Homotópicas

Definición 4.9.8. Dadas dos trayectorias $\gamma_0, \gamma_1 \in C([0,1], X)$ en un espacio topológico X tales que $\gamma_0(0) = \gamma_1(0)$ y $\gamma_0(1) = \gamma_1(1)$, se define una homotopía con puntos extremos fijos como una homotopía $\{\hat{\gamma}_t\}_{t\in[0,1]} \subset C([0,1], X)$ que cumple las siguiente restricciones.

$$\begin{cases} \hat{\gamma}_s(t) \in X \\ \hat{\gamma}_s(0) = \gamma_0(0) = \gamma_1(0) \\ \hat{\gamma}_s(1) = \gamma_0(1) = \gamma_1(1) \quad , \quad s,t \in [0,1] \\ \hat{\gamma}_0(t) = \gamma_0(t) \\ \hat{\gamma}_1(t) = \gamma_1(t) \end{cases} \tag{4.9.2}$$

Notación 4.9.9. Dadas dos trayectorias $\gamma_0, \gamma_1 \in C([0,1], X)$ en un espacio topológico X, se escribirá que γ_0, γ_1 son **homotópicas con puntos extremos fijos** si $\gamma_0(0) = \gamma_1(0)$ y $\gamma_0(1) = \gamma_1(1)$ y si existe una homotopía $\{\hat{\gamma}_t\}_{t\in[0,1]} \subset C([0,1], X)$ que cumple las restricciones (4.9.2), la condición de γ_0, γ_1 de ser homotópicas con puntos extremos fijos se denotará de forma abreviada en este curso como $\gamma_0 \sim_h \gamma_1$.

Un ejemplo gráfico de homotopía con puntos extremos fijos entre dos trayectorias en $\mathbb{C}\backslash\{0\}$ se ilustra en la fig. 4.4.

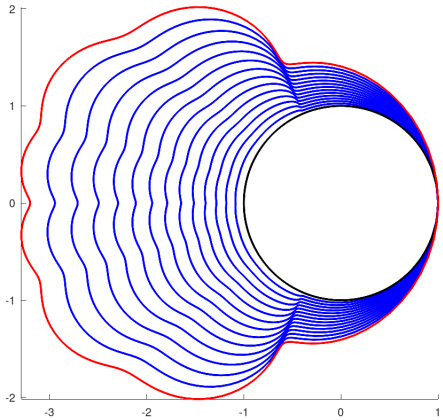


Figura 4.4: Ilustración gráfica de una homomtopía con puntos extremos fijos $\{f\}_{s\in[0,1]} \subset C([0,1], \mathbb{C}\backslash\{0\})$ en $\mathbb{C}\backslash\{0\}$ entre dos trayectorias $\gamma : [0,1] \to \mathscr{S}$ (representada por curva coloreada en negro) y $\gamma_p : [0,1] \to p(\mathscr{S})$ (representada por curva coloreada en rojo), para $\mathscr{S} = \{z \in \mathbb{C} : |z| = 1\} \subset \mathbb{C}\backslash\{0\}$, $\gamma(t) = \exp(2\pi i t)$ para $t \in [0,1]$, $p \in \mathbb{C}[z]$ definido por $p(z) = \frac{1}{10}z^8 + \frac{1}{10}z^7 + \frac{1}{10}z^6 + \frac{1}{10}z^5 + \frac{1}{10}z^4 + \frac{1}{10}z^3 + \frac{1}{2}z^2 + \frac{9}{5}z - \frac{7}{10}$, $\gamma_p(t) = p(\exp(2\pi i t)) - p(1) + 1$ para $t \in [0,1]$ y $f_s(t) = \frac{\gamma_p(t)}{|\gamma_p(t)|^s}$, $0 \leq s, t \leq 1$.

Lema 4.9.10. *La relación $\sim_h$ es una relación de equivalencia entre trayectorias en un espacio topológico.*

Demostración. Sea X un espacio topológico. Dada $\alpha \in C([0,1], X)$, es claro que la familia de funciones $\{\alpha_t\}_{t\in[0,1]} \subset X^{[0,1]}$ definida por $\alpha_t(s) = \alpha(s)$ para cada $t, s \in [0,1]$ es una homotopía de α a α. En efecto, $\alpha_t = \alpha \in C([0,1], X)$ para cada $t \in [0,1]$, $\alpha_0 = \alpha$, $\alpha_1 = \alpha$ y $\alpha_t(0) = \alpha(0)$ y $\alpha_t(1) = \alpha(1)$ para cada $t \in [0,1] \Longrightarrow \alpha \sim_h \alpha$.

Dadas $\alpha, \beta \in C([0,1], X)$ tales que $\alpha \sim_h \beta$, se cumple que existe una homotopía con puntos extremos fijos $\{\hat{\alpha}_t\}_{t\in[0,1]} \subset C([0,1], X)$ de α a β. Es claro con base en la definición de la operación $\sim_h$ que la familia $\{\hat{\alpha}_{1-t}\}_{t\in[0,1]} \subset C([0,1], X)$ define una homotopía de β a $\alpha \Longrightarrow \beta \sim_h \alpha$.

Dadas $\alpha, \beta, \gamma \in C([0,1], X)$ tales que $\alpha \sim_h \beta$ y $\beta \sim_h \gamma$, se cumple que existen homotopías con puntos extremos fijos $\{\hat{\alpha}_t\}_{t\in[0,1]}, \{\hat{\beta}_t\}_{t\in[0,1]} \subset C([0,1], X)$ de α a β y de β a γ, respectivamente. Es claro con base en la definición de la operación $\sim_h$ que la familia $\{\hat{\gamma}_t\}_{t\in[0,1]} \subset C([0,1], X)$ determinada por la expresión

$$\hat{\gamma}_t(s) = \begin{cases} \hat{\alpha}_{2t}(s), & t \in [0, 1/2] \\ \hat{\beta}_{2t-1}(s), & t \in [1/2, 1] \end{cases}$$

define una homotopía de α a $\gamma \Longrightarrow \alpha \sim_h \gamma$. Los argumentos previos implican que la relación $\sim_h$ es reflexiva, simétrica y transitiva, por tanto $\sim_h$ es una relación de equivalencia. □

Notación 4.9.11. Dada una trayectoria $\alpha \in C([0,1], X)$ en un espacio topológico X, en este curso se escribirá $[\alpha]_h$ para denotar la clase de equivalencia (de homotopía) de α, la cual está determinada por la expresión $[\alpha]_h = \{\gamma \in C([0,1], X) : \gamma \sim_h \alpha\}$.

Tanto en la fig. 4.3 como en la fig. 4.4 se ilustran tipos de trayectorias que jugarán un papel fundamental en el trabajo de clasificación de espacios topológicos correspondiente a este curso.

Definición 4.9.12. Dado un espacio topológico X y un punto $x \in X$. Una trayectoria $\gamma \in C([0,1], X)$ se denomina un **bucle** o **lazo** en X con base en x, si $\gamma(0) = x = \gamma(1)$. El conjunto de todos los bucles en X con base en x se denotará por $\pi(X, x)$ en este curso.

Observación 4.9.13. Dado un espacio topológico X y dado $x \in X$. Es claro que $\pi(X, x) = \{\alpha \in C([0,1], X) : \alpha(0) = x = \alpha(1)\}$

Notación 4.9.14. Dados dos espacios topológicos X, Y y $y \in Y$, en este curso se escribirá $\hat{y}$ para denotar la función $\hat{y} : X \to Y$ definida por $\hat{y}(x) = y$ para cada $x \in X$.

4.9.3. Algunas propiedades del cómputo de índices en $\pi_1(\mathbf{S}^1, 1)$

Notación 4.9.15. $\mathbf{S}^1 = \{z \in \mathbb{C} : |z| = 1\}$.

Notación 4.9.16. $\pi_1(\mathbf{S}^1, 1) = \pi(\mathbf{S}^1, 1)/\sim_h = \{[\alpha]_h : \alpha \in \pi(\mathbf{S}^1, 1)\}$.

Definición 4.9.17. Dada $\omega \in C([0,1], \mathbb{C})$ tal que $\omega(0) = \omega(1)$ y dado $z_0 \in \mathbb{C}$, el **número de giros** o **índice** de ω con respecto a z_0 se define como el número entero correspondiente a la expresión.

$$N_{z_0}(\omega) = \frac{1}{2\pi i} \oint_{\omega} \frac{dz}{z - z_0}$$

Observación 4.9.18. Es importante observar que la integral correspondiente al índice $N_{z_0}(\omega)$ debe calcularse en el sentido positivo (anti-horario) de recorrido de $\omega([0,1])$.

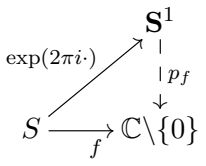
Definición 4.9.19. Dada $\omega \in C([0,1], \mathbf{S}^1)$ en $\mathbf{S}^1$ tal que $\omega(0) = \omega(1) = 1$, se define el índice $\mathrm{ind}([\omega]_h)$ de $[\omega]_h$, como el número entero definido por la expresión $\mathrm{ind}([\omega]_h) = N_0(\omega)$.

Proposición 4.9.20. *Dadas $[\alpha]_h, [\beta]_h \in \pi_1(\mathbf{S}^1, 1)$ el cómputo de índices en $\pi_1(\mathbf{S}^1, 1)$ cumple las siguientes propiedades:*

1. $\mathrm{ind} : \pi_1(\mathbf{S}^1, 1) \to \mathbb{Z}$ *está bien definida.*
2. $\mathrm{ind}([\alpha]_h) = \mathrm{ind}([\beta]_h) \Longrightarrow [\alpha]_h = [\beta]_h$.
3. *Dado $z \in \mathbb{Z}$ existe $[\omega_z]_h \in \pi_1(\mathbf{S}^1, 1)$ tal que $\mathrm{ind}([\omega_z]_h) = z$.*

Observación 4.9.21. Es importante observar que la función ind está bien definida en $\pi_1(\mathbf{S}^1, 1)$, antes de calcular el índice de un elemento de un grupo fundamental dado, es importante determinar si la operación podría estar definida.

Definición 4.9.22. Dado $S \subset \mathbb{R}$, se dice que $f \in C(S, \mathbb{C}\backslash\{0\})$ está topológicamente controlado por $\mathbf{S}^1$ si existe $p_f \in C(\mathbf{S}^1, \mathbb{C}\backslash\{0\})$ tal que p_f *resuelve* el siguiente diagrama conmutativo.



4.9.4. Proyectos Computacionales

Visión Computarizada y Detección de movimiento periódico

En esta sección se resolverá el problema correspondiente al desarrollo de un algoirtmo topológico elemental de visión computacional asistida y detección de movimiento periódico. Este algoritmo topológico basado en la función $\mathrm{ind} : \pi_1(\mathbf{S}^1, 1) \to \mathbb{Z}$ permite a la coputadora detectar el movimiento periódico observado a través de una secuencia de imágenes que simulan un proceso de visión computacional asistida. Para este propósito se utilizará el programa `PMotionID.m` desarrollado por Fredy Vides como parte del proyecto **ACRPkG** actualmente en desarrollo en el **CICC-UNAH**.

El código Octave del programa `PMotionID.m` se muestra a continuación.

```
function [Wt,Wtr,T]=PMotionID(N,M,tol)
% Programador: Fredy Vides
% Proyecto: ACRPkG/CICC-UNAH 2020
% Example:
% [Wt,Wtr,T]=PMotionID(33,200,1e-11);
t=0:2/(N-1):2;
x=-1:2/(M-1):1;
z=@(t)cos(2*pi*t).*(1-x.^2).^3;
for k=1:N
plot(x,z(t(k)),'linewidth',5,[-1 1],[0 0],'r.','markersize',20);
axis ([-1 1 -1 1]);
axis square;
axis off;
print("-dpng",['wave-',num2str(k),'.png']);
end
W=double(imread (['wave-1.png']));
imshow(uint8(W));
pause(.1);
R=sparse(size(W,1),size(W,2));
W=W(:,:,1);
DW =del2(W);
f=find(DW);
Wr=R;
Wr(f)=W(f);
Wt=Wr(:);
for k=2:N
W=double(imread (['wave-',num2str(k),'.png']));
imshow(uint8(W));
pause(.1);
W=W(:,:,1);
DW =del2(W);
f=find(DW);
Wr=R;
Wr(f)=W(f);
Wt=[Wt Wr(:)];
end
```

```
[u,s,~]=svd(Wt,0);
s=diag(s)';
f=find(s>=tol);
Wtr=u(:,f)'*Wt;
wt=abs(fft(Wtr(1,:)))/sqrt(N);
Th=min(wt);
f=find(wt>=Th);
F=@(z)(z.^(floor(N/f(2))))/sqrt(N);
T=NGiros(F,0,N);
end
```

El programa `PMotionID.m` aplica el programa Octave `NGiros.m` para calcular el índice del bucle elemental de identificación correspondiente. El código del programa `NGiros.m` se muestra a continuación.

```
function N=NGiros(f,z0,n)
%%
% Programador: Fredy Vides
% Proyecto: ACRPkG/CICC-UNAH 2020
% Example:
% f=@(x)polyval([.1 .1 .1 .1 .1 .1 .5 1.8 -.7],x);
% N=NGiros(f)
%%
if nargin<2
n=14;
z0=0;
end
if n<=1, n=2;end
t=0:1/n:1;
z=exp(2*pi*i*t(1:n));
pf=polyfit(z,f(z),n);
dpf=polyder(pf);
dpf=@(t)(exp(2*pi*i*t).*polyval(dpf,exp(2*pi*i*t))./...
(polyval(pf,exp(2*pi*i*t))-z0);
N=real(quadl(dpf,0,1,1e-14));
```

Un ejemplo de ejecución del programa `PMotionID.m` se muestra a continuación.

```
>> [Wt,Wtr,T]=PMotionID(33,200,1e-11);
>> T
T =  16.000
```

Identificación Dinámica en $\mathbf{S}^1_{GT} = \{z \in \mathbb{C} : |\mathrm{Re}(z)| + |\mathrm{Im}(z)| = 1\}$ topológicamente controlada por $\mathbf{S}^1$

En esta sección se utiliza el programa Octave `SystemIDTG.m` desarrollado por Fredy Vides como parte del proyecto **ACRPkG**, con el propósito de identificar dinámica topológica periódica en el subespacio topológico $\mathbf{S}^1_{GT} = \{z \in \mathbb{C} : |\mathrm{Re}(z)| + |\mathrm{Im}(z)| = 1\} \subset \mathbb{C}$. El programa requiere la aplicación del programa `NGiros.m` previamente descrito, y requiere también la implementación

de una función Octave que permita calcular el mapeo $\mathscr{S} : (\mathbf{S}^1_{GT}, 1) \to (\mathbf{S}^1, 1)$, la implementación computational de Octave de $\mathscr{S}$ ha sido implementada por Fredy Vides como el programa Octave `S1gt.m`, pero el código no se comparte en este punto dado que el cálculo y la implementación computacional en Octave del mapa $\mathscr{S} : (\mathbf{S}^1_{GT}, 1) \to (\mathbf{S}^1, 1)$ es un **ejercicio para el lector**.

El código Octave del programa `SystemIDTG.m` se muestra a continuación.

```
function [Zr,Ind]=SystemIDTG(I,T,N)
% Programador: Fredy Vides
% Proyecto: ACRPkG/CICC-UNAH 2020
% Examples:
% [Z,Ind]=SystemIDTG([0 .5],2,101);
% [Z,Ind]=SystemIDTG([0 1],2,201);

f = @(t)S1gt(exp(2*pi*i*T*t));

dI=diff(I);
t=I(1):dI/(N-1):I(2);

S1GT=f(t);

S1r=S1GT./abs(S1GT);

S1r0=S1r(1:(N-1));
S1r1=S1r(2:N);
Zr=S1r1/S1r0;
F=@(z)z.^floor((N*angle(Zr))/(2*pi*dI));

Ind=NGiros(F,0,floor(N/10));

S1GTr=S1gt(F(exp(2*pi*i*t)));

for k=1:N
subplot(121);
Z=S1GT(1:k);
plot(real(Z),imag(Z),'b','linewidth',4,...
real(Z(1)),imag(Z(1)),'g.','markersize',...
24,real(Z(k)),imag(Z(k)),'r.','markersize',24);
axis ([-1 1 -1 1]);
axis square;
title('Dinámica observada')
pause(.1);
subplot(122);
Z=S1GTr(1:k);
plot(real(Z),imag(Z),'b','linewidth',4,...
real(Z(1)),imag(Z(1)),'g.','markersize',...
24,real(Z(k)),imag(Z(k)),'r.','markersize',24);
axis ([-1 1 -1 1]);
axis square;
```

```
title('Dinámica identificada')
pause(.1);
end
end
```

Un ejemplo de implementación del programa `SystemIDTG.m` se muestra a continuación.

```
>> [Z,Ind]=SystemIDTG([0 1],2,201);
>> Ind
Ind =  2.0000
```

Una de las salidas gráficas producidas por el programa `SystemIDTG.m` se ilustra en la fig. 4.5.

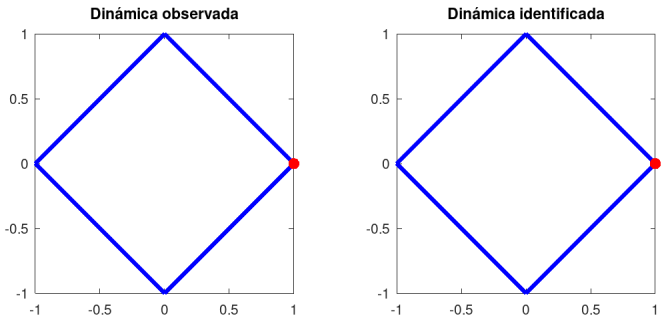


Figura 4.5: Salida gráfica del programa `SystemIDTG.m`.

Ejercicios Adicionales para el Lector

Ejercicio para el lector 4.9.1. Adaptar el código `SystemIDTG.m` para desarrollar un programa Octave para realizar identificación de dinámica periódica en el espacio $\mathbf{S}^1_{max} = \{z \in \mathbb{C} : \text{máx}\{|\text{Re}(z)|, |\text{Im}(z)|\} = 1\}$.

Ejercicio para el lector 4.9.2. Adaptar el código `SystemIDTG.m` para desarrollar un programa Octave para realizar identificación de dinámica periódica en el espacio $\mathbf{S}^1_{\mathbb{R}} = \{x \in \mathbb{R}^2 : \|x\|_2 = 1\}$.

Ejercicio para el lector 4.9.3. Adaptar el código `SystemIDTG.m` para desarrollar un programa Octave para realizar identificación de dinámica periódica en el espacio $\mathbf{S}^1_{GT,\mathbb{R}} = \{(x, y) \in \mathbb{R}^2 : |x|+|y| = 1\}$.

Capítulo 5

Técnicas Avanzadas de Simulación Numérica con GNU Octave

5.1. Elementos de Desarrollo de Archivos-Oct

Los archivos-oct permiten, entre otras posibilidades, integrar en un mismo programa elementos de funciones y programas de Octave, con rutinas y librerías de C++ y Fortran, en esta sección se considera en particular la siguiente descripción particular.

$$\text{Archivo} - \text{Oct} = Octave + Fortran + C + + + ...$$

5.1.1. Referencias Web

En esta sección se utilizarán elementos de las siguientes referencias web:

- `https://octave.org/doc/v6.1.0/Oct_002dFiles.html`
- `https://octave.org/doxygen/stable/`
- `https://octave.sourceforge.io/parallel/`

5.1.2. Cómputo Matricial Básico con Archivos-Oct

Proyecto 1: Suma de Matrices Complejas

En esta sección se desarrollará un programa-oct que realiza la suma de dos matrices complejas. El código C++ del programa será nombrado `SumaMatrices.cc`.

```
1  #include <octave/oct.h>
2
3  DEFUN_DLD (SumaMatrices, args, , "Calcula A + B")
4  {
5    if (args.length () != 2)
6      print_usage ();
7
8    ComplexMatrix A = args(0).complex_matrix_value ();
9    ComplexMatrix B = args(1).complex_matrix_value ();
10
```

```
11   return octave_value (A+B);
12 }
```

Para construir el archivo `SumaMatrices.oct` basta ejecutar la siguiente secuencia de comandos en la consola de Octave o del intérprete de Linux.

```
$ mkoctfile-6.1.0 SumaMatrices.cc
```

Para verificar el funcionamiento del programa `SumaMatrices.oct` basta iniciar una sesión Octave en el directorio donde se guardó el programa-oct y ejecutar las siguientes secuencias de comandos.

```
>> a=randn(4,7)+randn(4,7)*i;\\
>> b=randn(4,7)+randn(4,7)*i;\\
>> norm(a+b-SumaMatrices(a,b),'fro')\\
ans = 0
```

Proyecto 2: Producto de Matrices Complejas

En esta sección se desarrollará un programa-oct que realiza el producto de dos matrices complejas. El código C++ del programa será nombrado `ProductoMatrices.cc`.

```
1  #include <octave/oct.h>
2
3  DEFUN_DLD (ProductoMatrices, args, , "Multiplica A * B")
4  {
5    if (args.length () != 2)
6      print_usage ();
7
8    ComplexMatrix A = args(0).complex_matrix_value ();
9    ComplexMatrix B = args(1).complex_matrix_value ();
10
11   return octave_value (A*B);
12 }
```

Para generar el archivo `ProductoMatrices.oct` basta ejecutar la siguiente secuencia de comandos en la consola de Octave o del intérprete de Linux.

```
$ mkoctfile-6.1.0 ProductoMatrices.cc
```

Para verificar el funcionamiento del programa `ProductoMatrices.oct` basta iniciar una sesión Octave en el directorio donde se guardó el programa-oct y ejecutar las siguientes secuencias de comandos.

```
>> a=randn(4,7)+randn(4,7)*i;
>> b=randn(7,5)+randn(7,5)*i;
>> norm(a*b-ProductoMatrices(a,b),'fro')
ans = 0
```

Proyecto 3:

En esta sección se desarrollará un programa-oct que resuelve el problema:

$$X = \arg \min_{Y \in \mathbb{C}^{n \times m}} \|AY - B\|_F$$

para $A \in \mathbb{C}^{p \times n}$, $B \in \mathbb{C}^{p \times m}$ dadas.

El código C++ del programa será nombrado `LSSolver.cc`.

```
#include <octave/oct.h>

DEFUN_DLD (LSSolver, args, , "Calcula solución X de mínimos cuadrados de AX = B")
{
  if (args.length () != 2)
    print_usage ();

  ComplexMatrix A = args(0).complex_matrix_value ();
  ComplexMatrix B = args(1).complex_matrix_value ();

  return octave_value (A.lssolve(B));
}
```

Para generar el archivo `LSSolver.oct` basta ejecutar la siguiente secuencia de comandos en la consola de Octave o del intérprete de Linux.

```
$ mkoctfile-6.1.0 LSSolver.cc
```

Para verificar el funcionamiento del programa `LSSolver.oct` basta iniciar una sesión Octave en el directorio donde se guardó el programa-oct y ejecutar las siguientes secuencias de comandos.

```
>> A=randn(500,7000)+randn(500,7000)*i;
>> b=randn(500,100)+randn(500,100)*i;
>> tic,x=LSSolver(A,b);toc
Elapsed time is 2.19737 seconds.
>> norm(A*x-b,'fro')
ans = 1.0529e-12
```

5.2. Archivos-Oct para Cómputo de Modelos Autrorregresivos

5.2.1. Proyecto de Cómputo:

Desarrollar un archivo-oct `ARModelSolver.oct` para resolver el problema

$$A = \arg \min_{\hat{A} \in \mathbb{C}^{1 \times L}} \sum_{t=L}^{T-1} \left| Y(t+1) - \sum_{k=0}^{L-1} \hat{A}_{k+1} Y(t-k) \right|^2$$

para una secuencia de datos $\{Y(t) : 1 \leq t \leq T\} \subset \mathbb{C}$ correspondiente a una serie de tiempo $\{Y(t) : t \geq 1\} \subset \mathbb{C}$, para $1 \leq L \leq T-1$. Donde se considera la posiblilidad de que la señal $\{Y(t) : t \in \mathbb{Z}^+\}$ está determinada por una perturbación por ruido (pseudo)aleatorio de una señal estacionaria.

El código fuente de referencia para `ARModelSolver.cc` se presenta a continuación.

```
#include <iostream>
#include <octave/oct.h>
#include <octave/builtin-defun-decls.h>
#include <octave/parse.h>

DEFUN_DLD (ARModelSolver, args,,"Compute AR Model from time series data.")
{
  int nargin = args.length ();
  if (nargin < 1) print_usage ();
  int n=args(0).length();
  int L=args(1).int_value()+1;
  ComplexMatrix X;
  ComplexMatrix A;
  ComplexMatrix y;
  X=args(0).complex_matrix_value();
  X=X.transpose();
  A=X.linear_slice(0,n-L);
  for (int k=1;k<=(L-2);k++) A=ComplexMatrix(X.linear_slice(k,k+n-L)).append(A);
  y=X.linear_slice(L-1,n-1);
  A=A.lssolve(y).transpose();
  return octave_value(A);
}
```

Para generar el archivo `ARModelSolver.oct` basta ejecutar la siguiente secuencia de comandos en la consola de Octave o del intérprete de Linux.

```
$ mkoctfile-6.1.0 ARModelSolver.cc
```

Para verificar el funcionamiento del programa `ARModelSolver.oct`, se puede ejecutar el archivo `ARSolverDemo.m`, cuyo código fuente se presenta a continuación.

```
function ARSolverDemo(N)
t=0:2/34:12;
y=sin(pi*t)+.5*sin(3*pi*t)+.25*sin(7*pi*t)+.15*cos(7*pi*t)-.1*sin(9*pi*t);
z=y+1e-4*randn(1,length(y));
H=hankel(z(1:N),z(N:end));
H0=H(:,1:(end-1));
H1=H(N,2:end);
[u,s]=svd([H0;H1(end,:)]',0);
rk=sum(diag(s)>1e-3);
u=u(:,1:rk);
H00=(H0*u).';
H11=(H1*u).';
disp("Tiempo de ejecución del método estándar:")
tic,A0=ARModelSolver(z,N);toc
disp("Tiempo de ejecución del método de orden reducido:")
tic,A1=fliplr(LSSolver(H00,H11).');toc
v0=flipud(H(:,1));
w0=v0;
```

```
v1=H(:,1).';
w1=v1;
for k=1:(length(z)-N),
z0=A0*v0;
v0=circshift(v0,1);
v0(1)=z0;
v1=[v1 z0];
z0=A1*w0;
w0=circshift(w0,1);
w0(1)=z0;
w1=[w1 z0];
end
s=0:1/100:1;
S1=exp(2*pi*i*s);
figure(1);
subplot(411);plot(t,z,'b',t,v1(1,:),'r.-');
title('Método de Modelación Estándar');
subplot(412);plot(t,abs(z-v1(1,:)),'r.-');
subplot(212);plot(S1,'b'),hold on,plot(roots([1 -A0]),'r.','markersize',9);
hold off;
axis tight;axis square;
figure(2);
subplot(411);plot(t,z,'b',t,w1(1,:),'r.-');
title('Método de Modelación de Orden Reducido');
subplot(412);plot(t,abs(z-w1(1,:)),'r.-');
subplot(212);plot(S1,'b'),hold on,plot(roots([1 -A1]),'r.','markersize',9);
hold off;
axis tight;axis square;
disp("Número de elementos distintos de cero en A0:"),disp(nnz(A0))
disp("Número de elementos distintos de cero en A1:"),disp(nnz(A1))
end
```

Para verificar el funcionamiento del programa `ARModelSolver.oct`, en una sesión Octave en el directorio donde se guardó el programa-oct previamente, se puede ejecutar el programa `ARSolverDemo.m` utilizando las siguientes secuencias de comandos.

```
>> ARSolverDemo(32)
Tiempo de ejecución del método estándar:
Elapsed time is 0.00447106 seconds.
Tiempo de ejecución del método de orden reducido:
Elapsed time is 0.00316095 seconds.
Número de elementos distintos de cero en A0:
32
Número de elementos distintos de cero en A1:
32
```

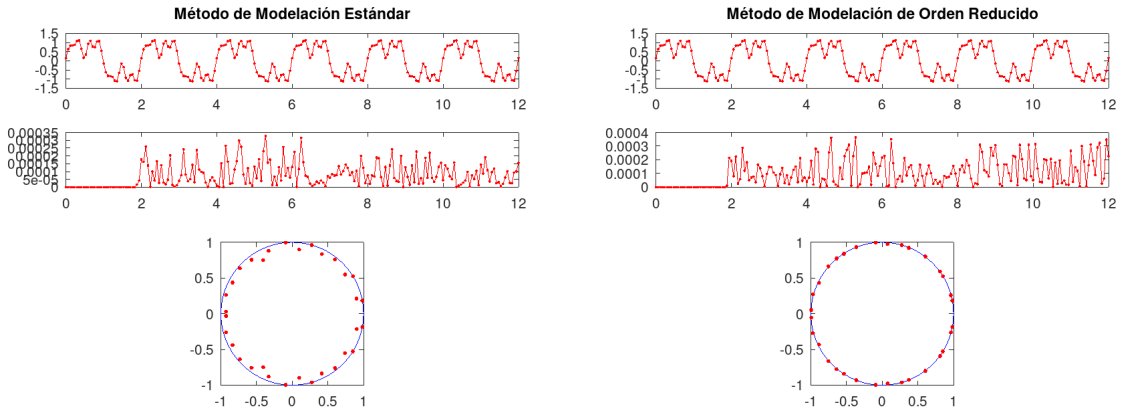


Figura 5.1: Salida gráfica correspondiente al método estándar implementado utilizando el programa `ARModelSolver.oct` (izquierda). Salida gráfica correspondiente al método estándar con reducción de orden implementado utilizando el programa `LSSolver.oct` (derecha).

5.3. Elementos de Cómputo Paralelo con GNU Octave

En esta sección se utilizará el paquete de cómputo paralelo `parallel` de Octave. El paquete puede instalarse en Octave ejecutando la siguiente secuencia de comandos en la consola de Octave.

```
>> pkg install -forge parallel
```

Una vez instalado el paquete, es posible cargar el paquete en un espacio de trabajo activo de Octave utilizando la siguiente secuencia de comandos.

```
>> pkg load parallel
```

5.3.1. Cómputo paralelo con Octave en computadores multiprocesador

En esta sección se implementará la función de cómputo paralelo `pararrayfun` del paquete `parallel`.

Cómputo Paralelo de Modelos Autorregresivos

Dado un entero positivo $N \geq 1$. En esta sección se desarrollar un archivo-m `PARModelSolver.m` basado en el programa `ARModelSolver.oct` desarrollado previamente, para resolver la familia de problemas

$$A_j = \arg \min_{\hat{A} \in \mathbb{C}^{1\times L}} \sum_{t=L}^{T-1} \left| Y_j(t+1) - \sum_{k=0}^{L-1} \hat{A}_{k+1} Y_j(t-k) \right|^2$$

para una familia de secuencias de datos $\{Y_j(t) : 1 \leq t \leq T\} \subset \mathbb{C}$ correspondiente a una serie de tiempo $\{Y_j(t) : t \geq 1\} \subset \mathbb{C}$, para $1 \leq L \leq T-1$, para $j = 1, \ldots, N$. Donde se considera la posiblilidad de cada señal $\{Y_j(t) : t \in \mathbb{Z}^+\}$ está determinada por una perturbación por ruido (pseudo)aleatorio de una señal estacionaria, para cada $j = 1, \ldots, N$.

El código fuente de referencia del programa `PARModelSolver.m` se presenta a continuación.

```
function A=PARModelSolver(y,L)
        A=pararrayfun(nproc,@(j)ARModelSolver(y{j},L),...
        1:size(y,2),"UniformOutput",0);
end
```

A continuación se presenta también el código fuente de referencia del programa `ARForcast.m`.

```
function y=ARForcast(A,y0,N)
        y0=y0(:);
        y=y0.';
        y0=flipud(y0);
        for k=1:N
                z0=A*y0;
                y0=circshift(y0,1);
                y0(1)=z0;
                y=[y z0];
        end
end
```

Este programa puede aplicarse para calcular un horizonte predictivo con base en el modelo matricial identificado. Un ejemplo de código Octave denominado PARModelDemo.m para el cómputo paralelo aplicado en identificación de señales se presenta a continuación.

```
function PARModelDemo(N,L,n)
        t=0:12/n:12;
        y=sin(pi*t)+.5*sin(3*pi*t)+.25*sin(7*pi*t)+...
        .15*cos(7*pi*t)-.1*sin(9*pi*t);
        z=y+1e-4*randn(1,length(y));
        Y=y+randn(N,size(y,2))*1e-5;
        for k=1:N, Yc{k}=Y(k,:);end
        tic,A=PARModelSolver(Yc,L);toc
        M=floor(N/3);
        K=[1 M 2*M 3*M];
        yk=pararrayfun(nproc,@(j)ARForcast(A{K(j)},Yc{K(j)}(1:L),n-L+1),...
                        1:4,"UniformOutput",0);
        j=1;
        figure(1),
        for k=[1 M 2*M 3*M]
                subplot(4,1,j), plot(t,Yc{k},'b',t,yk{j},'r.-');
                title('Forcasting')
                j++;
        end
        figure(2),
        j=1;
        for k=[1 M 2*M 3*M]
                subplot(4,1,j), plot(t,abs(Yc{k}-yk{j}),'r.-');
                title('Forcasting Absolute Error')
                j++;
        end
```

```
27          s=0:1/100:1;
28          S1=exp(2*pi*i*s);
29          figure(3),
30          j=1;
31          for k=[1 M 2*M 3*M]
32                  subplot(2,2,j);plot(S1,'b'),hold on,
33                  plot(roots([1 -A{k}]),'r.','markersize',9);
34                  hold off;
35                  title('Transition Operator Spectrum')
36                  axis square;axis tight;
37                  j++;
38          end
39  end
```

Un ejemplo de ejecución del programa `PARModelDemo.m` se muestra a continuación.

```
>> PARModelDemo(200,101,204)
Elapsed time is 1.37572 seconds.
```

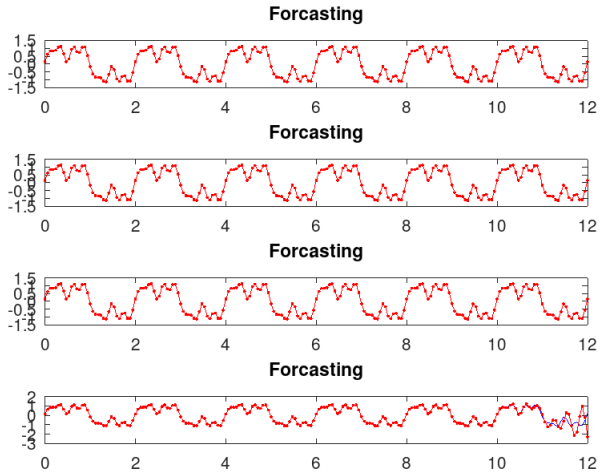


Figura 5.2: Salidas gráificas correspondientes a los horizontes predictivos generados por el programa `PARModelDemo.m`.

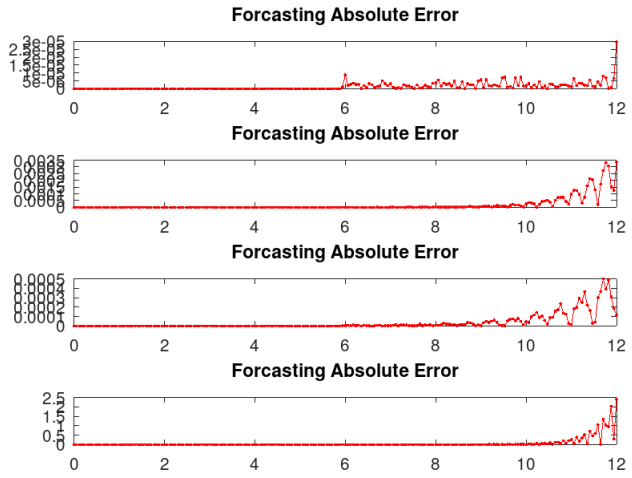


Figura 5.3: Salidas gráficas correspondientes a los errores de predicción generados por el programa `PARModelDemo.m`.

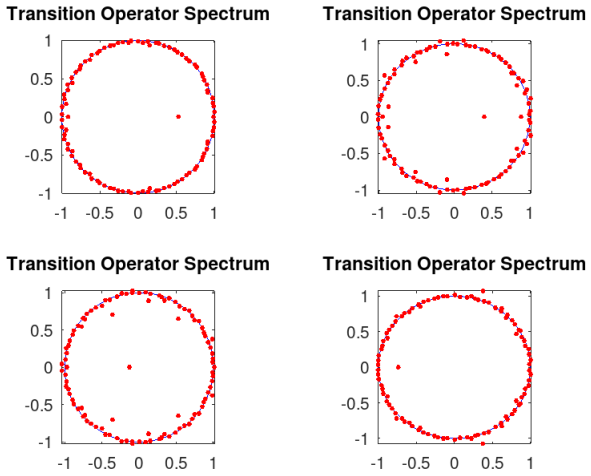


Figura 5.4: Salidas gráficas correspondientes a los espectros de los operadores de transición identificados por el programa `PARModelDemo.m`.

5.4. Ejercicios

1. Desarrollar un archivo-oct `SVDFactor.oct` para resolver el problema en **formato económico**

$$(U, S, V) = \arg \min_{(\hat{U},\hat{S},\hat{V}) \in \mathbb{PI}_* \times \mathbb{NN}_* \times \mathbb{PI}_*} \|A - \hat{U}\hat{S}\hat{V}\|_F$$

   para cualquier $A \in \mathbb{C}^{r \times m}$.

2. Desarrollar un archivo-oct `SVDSolver.oct` para resolver el problema $X = \arg \min_Y \|AY - B\|_F$ para matrices A, B dadas aplicando la descomposición económica en valores singulares

de A.

3. Desarrollar un programa Octave `ParSVDSolver.m` para resolver en paralelo N problemas de la forma $X_j = \arg\min_{Y_j} \|A_j Y_j - B_j\|_F$ para matrices A_j, B_j dadas para $j = 1, \ldots, N$, aplicando la descomposición económica en valores singulares de cada A_j, para $j = 1, \ldots, N$.

Parte II

FreeFEM y CalculiX

Capítulo 6

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.

6.1. 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 6.1.

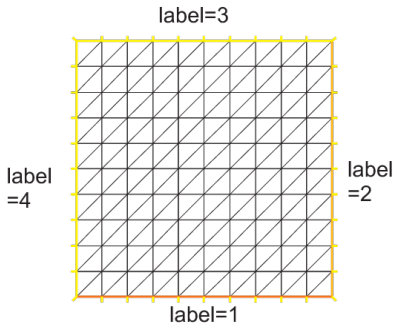


Figura 6.1: Malla rectangular básica genérica.

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 6.2.

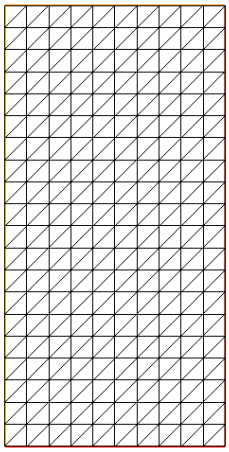


Figura 6.2: Malla rectangular básica genérica generada por `malla2D.edp`.

6.1.1. 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 6.3.

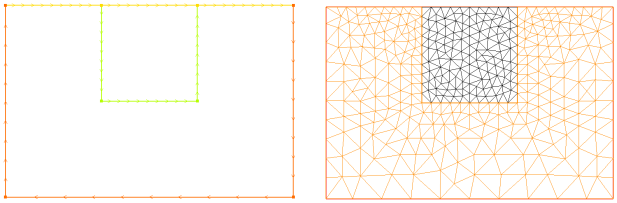


Figura 6.3: Malla rectangular basada en bordes generada por `ConstMalla2D.edp`.

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 6.4.

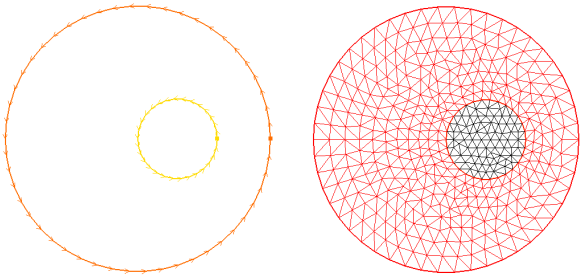


Figura 6.4: Malla bidimensional basada en bordes generada por `MallaMatComb2D.edp`.

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 6.5.

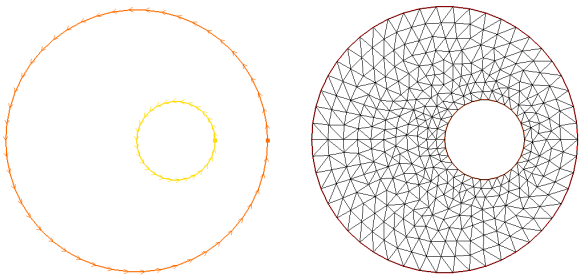


Figura 6.5: Malla bidimensional perforada basada en bordes generada por `MallaMatPer2D.edp`.

6.1.2. 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 6.6.

6.2. 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;
```

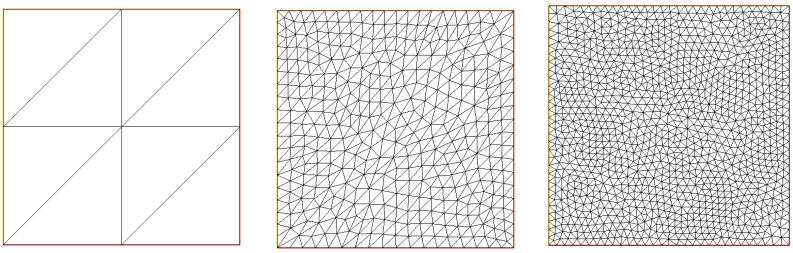


Figura 6.6: Malla bidimensional junto con dos niveles de refinamiento generados por `MallaRef2D.edp`.

```
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 6.7.

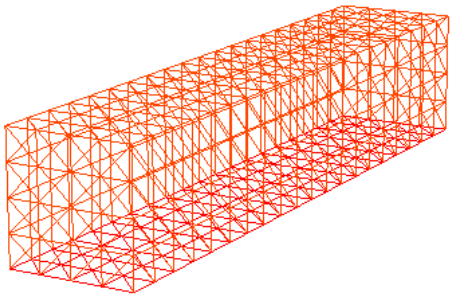


Figura 6.7: Malla tridimensional generada por `malla3D.edp`.

6.2.1. 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 6.8.

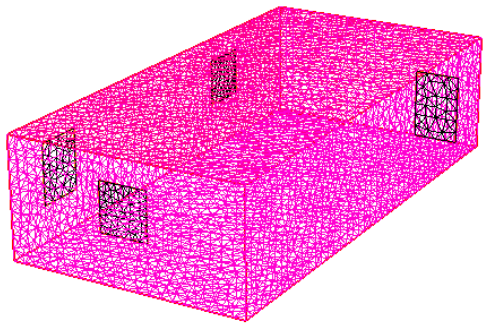


Figura 6.8: Modelo geométrico tridimensional de sala.

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

```
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 6.9.

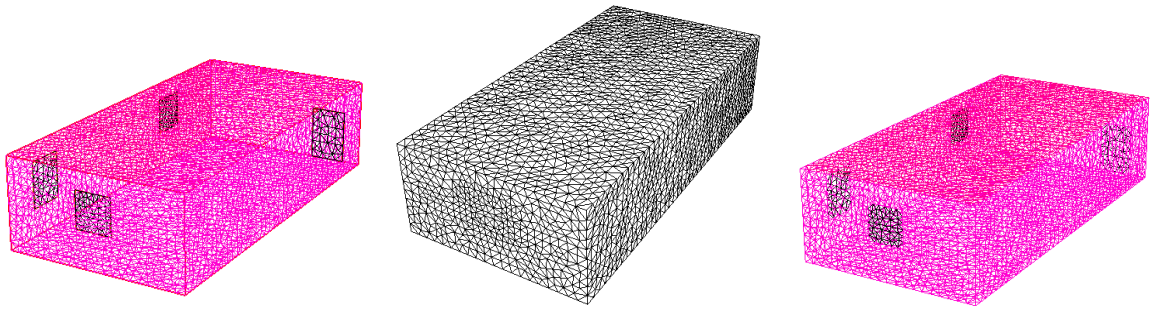


Figura 6.9: Malla tridimensional basada en bordes para la sala.

Cálculo de una Malla de un Material Tridimensional Perforado

Consideremos una pieza de material perforado como la descrita por la figura 6.10.

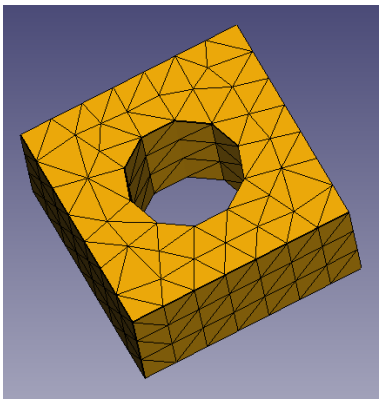


Figura 6.10: Representación geométrica de pieza de material tridimensional perforado.

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 6.11.

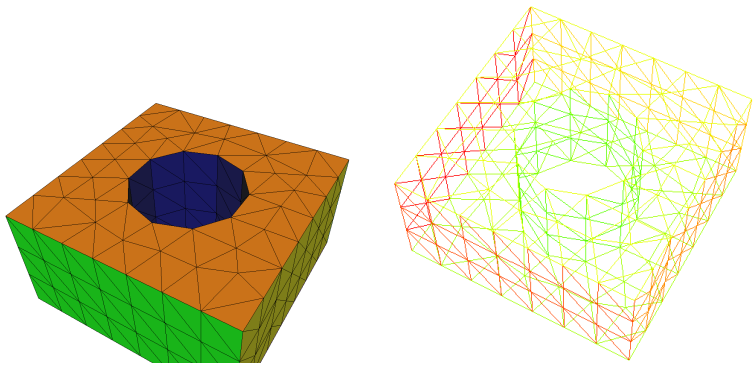


Figura 6.11: Malla tridimensional basada en bordes para la pieza.

6.2.2. 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 6.12.

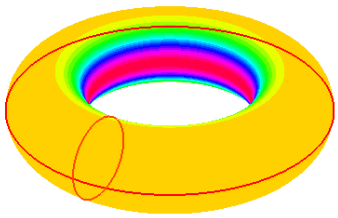


Figura 6.12: Superficie de material $\mathcal{T}$ en un medio continuo tridimensional.

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 6.13.

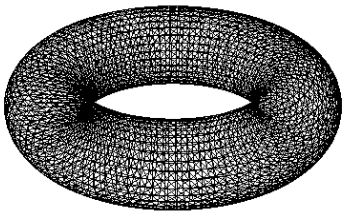


Figura 6.13: Malla superficial $\mathcal{T}_h$ del material $\mathcal{T}$.

Capítulo 7

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.

7.1. 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 7.1.

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 7.2.

7.1.1. 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

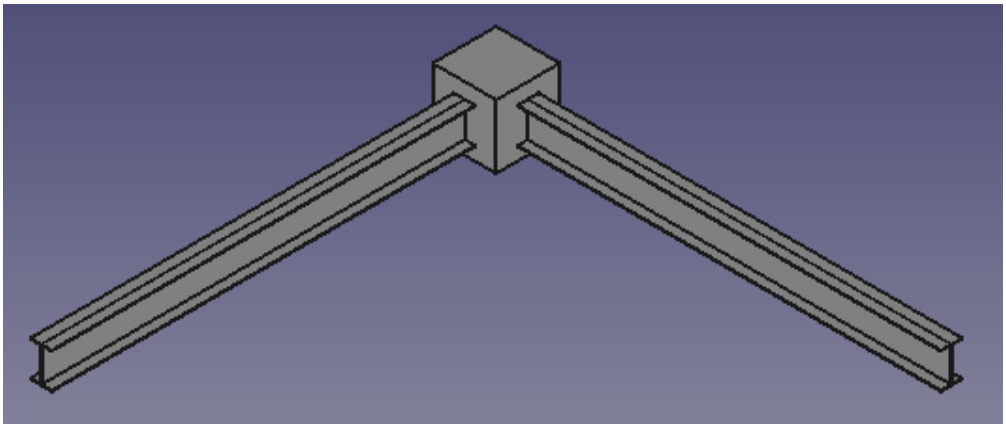


Figura 7.1: Arreglo de vigas de perfil I.

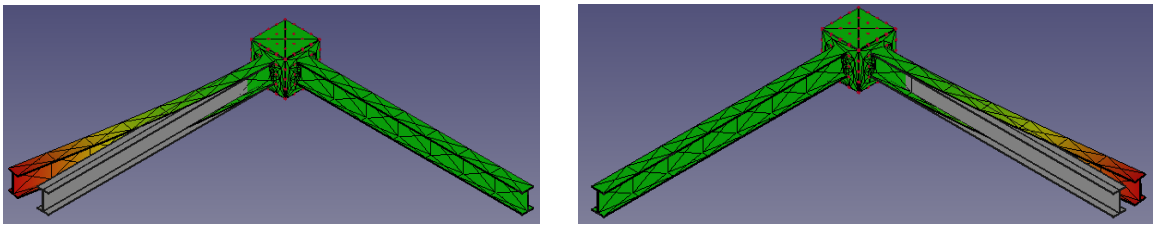


Figura 7.2: Deformaciones estáticas aproximadas del arreglo de vigas de perfil I.

describirse en la forma:

$$\mathcal{D}(\mathbf{X},t) = \mathbf{X} + \mathbf{u}(\mathbf{X},t) \tag{7.1.1}$$

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 (7.1.1) obtenemos la siguiente regla de transformación.

$$\mathbf{F} = \frac{\partial \mathcal{D}(\mathbf{X},t)}{\partial \mathbf{X}} = \frac{\partial \mathbf{X}}{\partial \mathbf{X}} + \frac{\partial \mathbf{u}(\mathbf{X},t)}{\partial \mathbf{X}} = \mathbf{1}_3 + \mathbf{G} \tag{7.1.2}$$

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

$$\mathbf{G} = \frac{\partial \mathbf{u}(\mathbf{X}, t)}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial u_1}{\partial X_1} & \frac{\partial u_1}{\partial X_2} & \frac{\partial u_1}{\partial X_3} \\ \frac{\partial u_2}{\partial X_1} & \frac{\partial u_2}{\partial X_2} & \frac{\partial u_2}{\partial X_3} \\ \frac{\partial u_3}{\partial X_1} & \frac{\partial u_3}{\partial X_2} & \frac{\partial u_3}{\partial X_3} \end{bmatrix} \tag{7.1.3}$$

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

$$\varepsilon(\mathbf{X}, t) = \frac{1}{2}\left(\mathbf{G} + \mathbf{G}^\top\right) \tag{7.1.4}$$

donde $\mathbf{G}$ es el gradiente de desplazamiento determinado por (7.1.3). 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

$$\sigma = \mathcal{C} : \varepsilon \tag{7.1.5}$$

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

$$\sigma = \lambda \mathrm{tr}(\varepsilon)\mathbf{1}_3 + 2\mu\varepsilon \tag{7.1.6}$$

donde

$$\begin{cases} \lambda = \frac{\nu E}{(1+\nu)(1-2\nu)} \\ \mu = \frac{E}{2(1+\nu)} \end{cases} \tag{7.1.7}$$

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 (7.1.6) tendremos que la ecuación de Cauchy para el balance de la cantidad de movimiento toma la forma.

$$\nabla \cdot \sigma(\mathbf{x}, t) + \rho_0 \mathbf{b}(\mathbf{x}, t) = \rho_0 \partial_t^2 \mathbf{u}(\mathbf{x}, t) \tag{7.1.8}$$

Al sustituir la ecuación (7.1.6) en (7.1.8) obtenemos la ecuación de Navier para el desplazamiento $\mathbf{u}(\mathbf{x}, t)$ del material $\mathcal{M}$, la cual estará determinada por la expresión.

$$\begin{cases} (\lambda + \mu)\nabla(\nabla \cdot \mathbf{u}) + \mu\nabla^2\mathbf{u} + \rho_0\mathbf{b} = \rho_0\partial_t^2\mathbf{u}(\mathbf{x}, t) \\ \mathbf{u}(\mathbf{x}, t) = \mathbf{u}^*(\mathbf{x}, t), \mathbf{x} \in \partial\mathcal{M} \\ \mathbf{u}(\mathbf{x}, 0) = u_0(\mathbf{x}) \\ \partial_t\mathbf{u}(\mathbf{x}, t) = u_1(\mathbf{x}) \end{cases} \tag{7.1.9}$$

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:

$$\mathbf{N}_{\lambda,\mu} = (\lambda + \mu)\mathbf{L}_h + \mu\mathbf{K}_h \tag{7.1.10}$$

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 (7.1.9). 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 (7.1.9) puede representarse aproximadamente por la expresión.

$$\mathbf{N}_{\lambda,\mu}\mathbf{u}_h + \rho_0\mathbf{b}_h = \rho_0\frac{d\mathbf{u}_h}{dt^2} \tag{7.1.11}$$

7.2. 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,

$$\sigma_{ij}(\mathbf{u}) = (\lambda \nabla \cdot \mathbf{u})\mathbf{1} + 2\mu\varepsilon(\mathbf{u}) \tag{7.2.1}$$

donde $\varepsilon(\mathbf{u}) = (1/2)(\mathbf{F} + \mathbf{F}^\top)$, para λ, μ definidas en (7.1.7).

Consideremos nuevamente la ecuación (7.1.8) de control de deformación de un material $\mathcal{M}$ en un medio continuo $\mathcal{E}$. Utilizando teoría de distribuciones podemos reformular la ecuación (7.1.8) de forma débil obtienendo la expresión,

$$\int_{\mathcal{M}} \sigma(\mathbf{u}) : \varepsilon(\mathbf{v})\, dx + \int_{\mathcal{M}} \mathbf{v} \cdot \mathbf{b}\, dx = 0; \tag{7.2.2}$$

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 (7.2.2) estará dada por la expresión.

$$\int_{\mathcal{M}} \lambda \nabla \cdot \mathbf{u} \nabla \cdot \mathbf{v} + 2\mu\varepsilon(\mathbf{u}) : \varepsilon(\mathbf{v})\, dx + \int_{\mathcal{M}} \mathbf{v} \cdot \mathbf{b}\, dx = 0; \tag{7.2.3}$$

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:

$$\mathbf{K}_{\lambda,\mu} = (\mathcal{A}(\hat{\mathbf{u}}_j, \hat{\mathbf{u}}_k)) \tag{7.2.4}$$

determinado por la forma variacional discretizada

$$\mathcal{A}(\hat{\mathbf{u}}_j, \hat{\mathbf{u}}_k) = \int_{\mathcal{M}_h} \lambda \nabla \cdot \hat{\mathbf{u}}_j \nabla \cdot \hat{\mathbf{u}}_k + 2\mu\varepsilon(\hat{\mathbf{u}}_j) : \varepsilon(\hat{\mathbf{u}}_k)\, dx \tag{7.2.5}$$

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

$$\mathcal{V}_h = \{\mathbf{u}_h |\ \mathbf{u}_h = u_1\hat{\mathbf{u}}_1 + \cdots + u_{M_V}\hat{\mathbf{u}}_{M_V}\} \tag{7.2.6}$$

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 (7.1.9) puede representarse aproximadamente por la expresión.

$$\mathbf{K}_{\lambda,\mu}\mathbf{u}_h + \rho_0\mathbf{b}_h = \rho_0\mathbf{M}\frac{d\mathbf{u}_h}{dt^2} \tag{7.2.7}$$

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

$$\begin{cases} \mathbf{b}_h = (\mathcal{F}(\hat{\mathbf{u}}_j)), \\ \mathbf{M} = (\mathcal{I}(\hat{\mathbf{u}}_j, \hat{\mathbf{u}}_k)) \end{cases} \tag{7.2.8}$$

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

$$\begin{aligned} \mathcal{F}(\hat{\mathbf{u}}_j) &= \int_{\mathcal{M}_h} \hat{\mathbf{u}}_j \cdot \mathbf{b}\, dx \\ \mathcal{I}(\hat{\mathbf{u}}_j, \hat{\mathbf{u}}_k) &= \int_{\mathcal{M}_h} \hat{\mathbf{u}}_j \cdot \hat{\mathbf{u}}_k\, dx \end{aligned} \tag{7.2.9}$$

7.2.1. 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 (7.1.11), 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.

$$\mathbf{N}_{\lambda,\mu}\mathbf{u}_h = -\rho_0 \mathbf{b}_h \tag{7.2.10}$$

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 7.3.

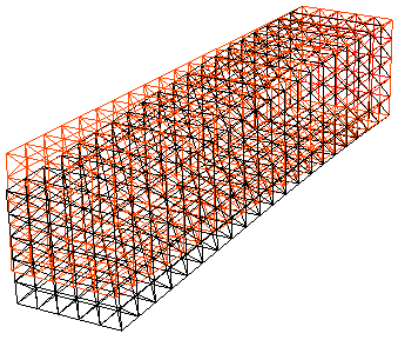


Figura 7.3: Aproximación de la deformación estática de la viga 3D en voladizo

2. Ejecutar `Beam3D.m` con GNU Octave.
   - La salida gráfica principal se muestra en la figura 7.4.

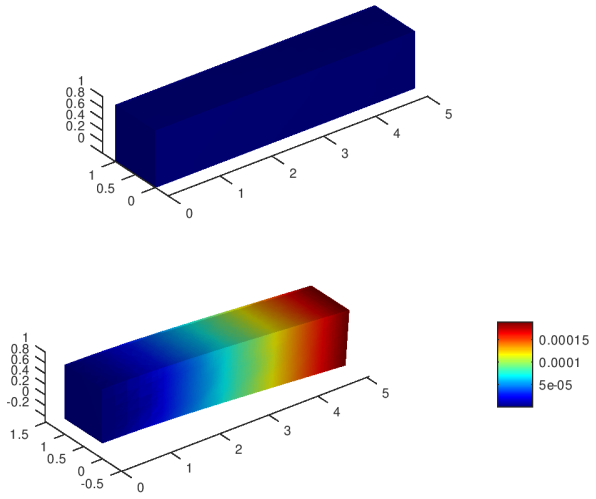


Figura 7.4: Aproximación de la deformación estática de la viga 3D en voladizo

7.2.2. 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 (7.1.11), 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.

$$\mathbf{N}_{\lambda,\mu}\hat{\mathbf{u}}_{k,h} = -\rho_0\omega_k^2\hat{\mathbf{u}}_{k,h} = \alpha_k\hat{\mathbf{u}}_{k,h}, \;\; 1 \leq k \leq N \tag{7.2.11}$$

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 7.5.

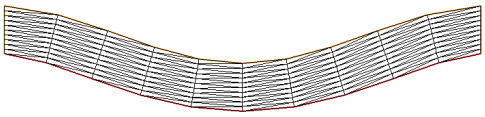


Figura 7.5: Aproximación de orden reducido plano de la deformación estática de la viga 3D doblemente apoyada

2. Ejecutar `Beam2D.m` con GNU Octave.
   - La salida gráfica principal se muestra en la figura 7.6.

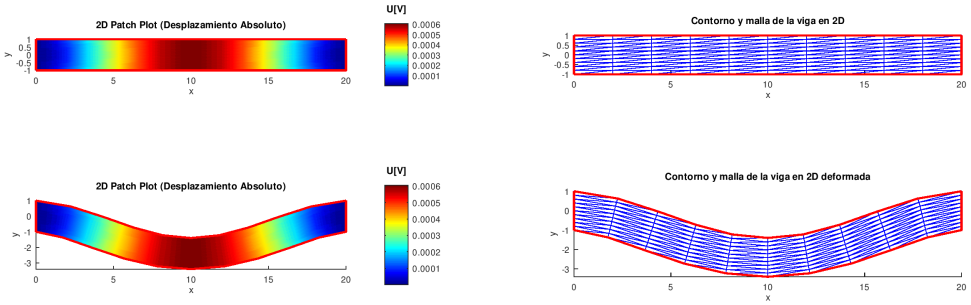


Figura 7.6: Aproximación de orden reducido plano de la deformación estática de la viga 3D doblemente apoyada

Cómputo de Respuetas Mecánicas de una Reduccińn 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 7.7.

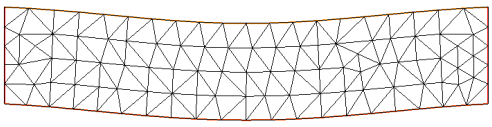


Figura 7.7: Aproximación de orden reducido plano de la respuesta mecánica de más baja frecuencia de la viga 3D doblemente apoyada

2. Ejecutar `Beam2D.m` con GNU Octave.
   - La salida gráfica principal se muestra en la figura 7.8.

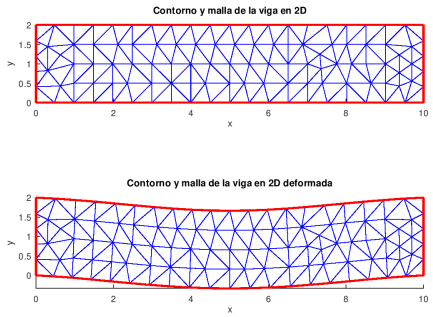


Figura 7.8: Aproximación de orden reducido plano de la respuesta mecánica de más baja frecuencia de la viga 3D doblemente apoyada

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 7.9.

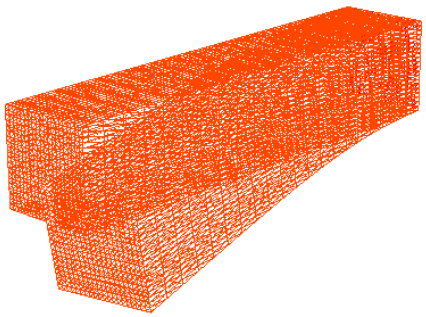


Figura 7.9: Respuesta mecánica aproximada de más baja frecuencia de la viga 3D en voladizo.

- 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 7.10.

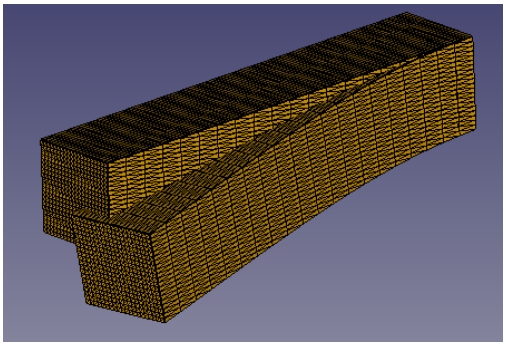


Figura 7.10: Respuesta mecánica aproximada de más baja frecuencia de la viga 3D en voladizo.

7.3. 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.

7.3.1. 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 (7.1.9) 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 7.11.

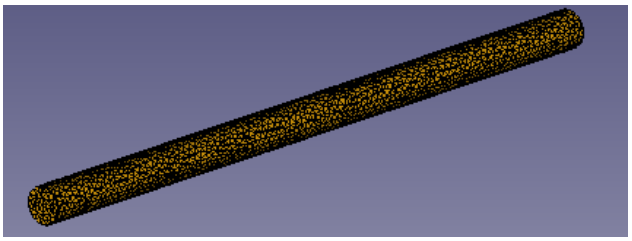


Figura 7.11: Representación de malla 3D aproximada $\mathcal{M}_h$ del material cilíndrico $\mathcal{M}$ en formato *.med.

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 7.12.

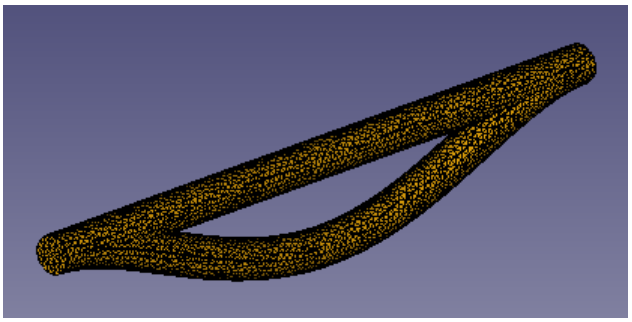


Figura 7.12: Representación 3D aproximada de la deformación $\mathcal{DM}_h$ del material cilíndrico $\mathcal{M}$ en formato *.med.

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 7.13.

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 7.14.

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 7.13.

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.

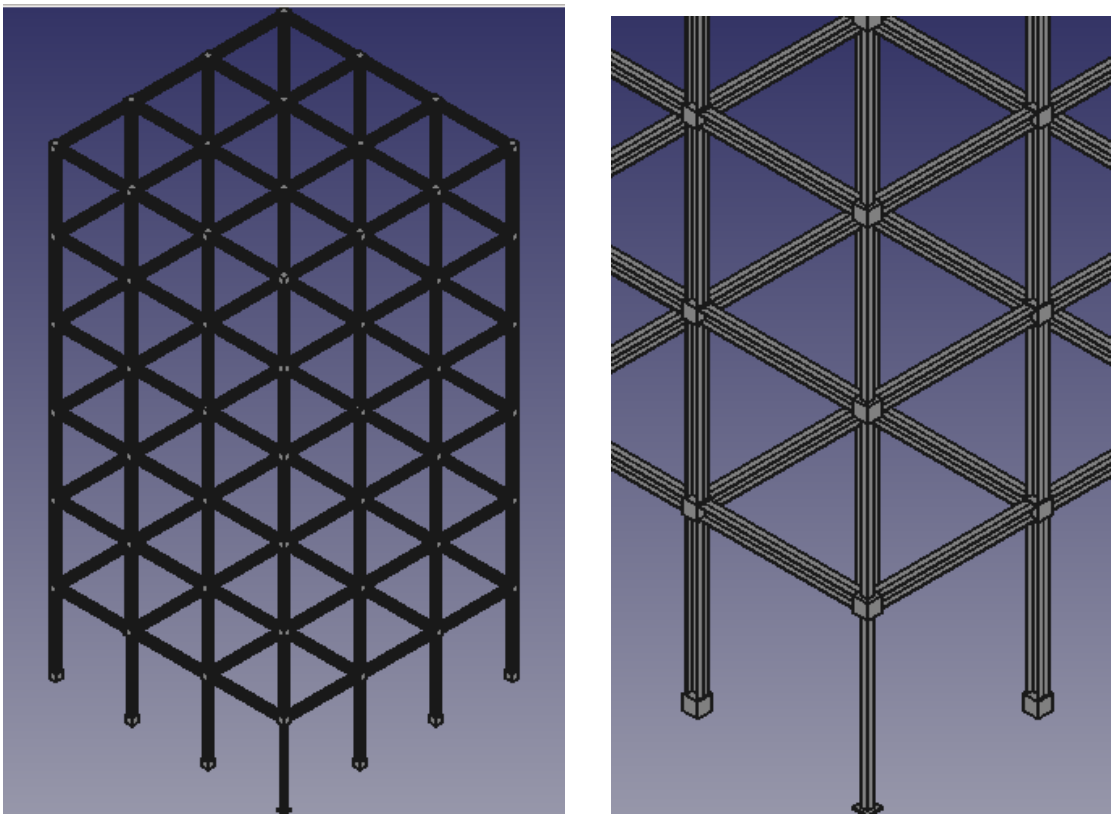


Figura 7.13: Esqueleto de acero de seis niveles $\mathcal{M}$: vista global (izquierda) y vista local (derecha).

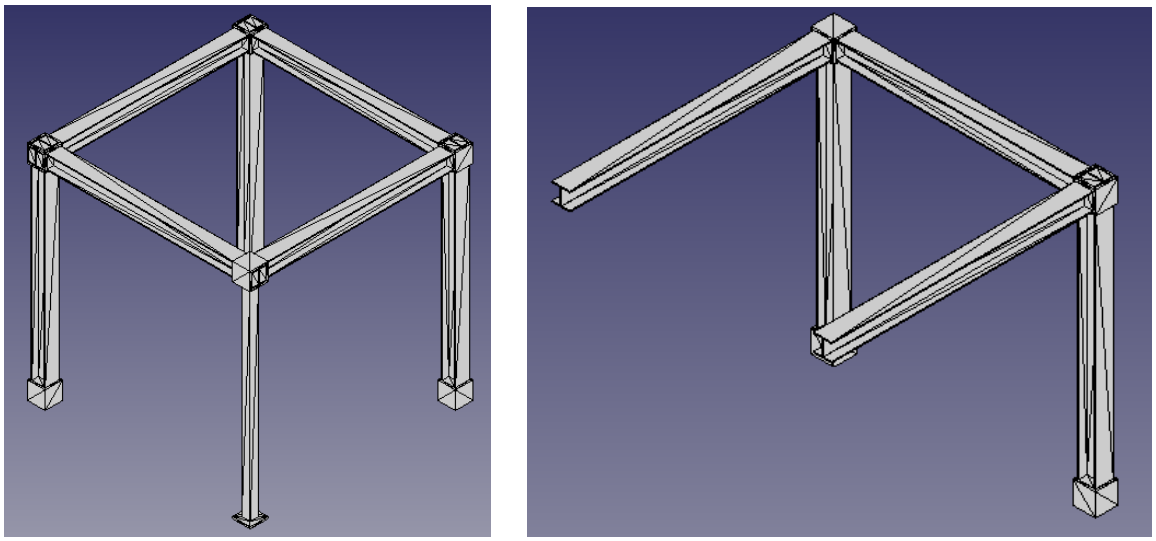


Figura 7.14: Componentes geométricas elementales: `ElementoA.stl` (izquierda) y `ElementoB.stl` (derecha).

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 7.15.

   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
   ```

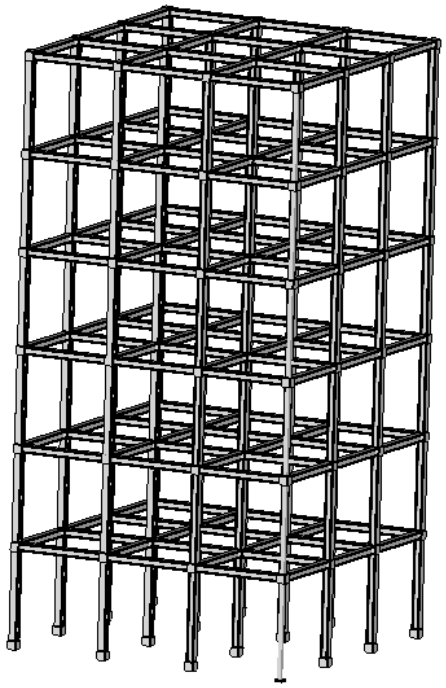


Figura 7.15: Salida gráfica del pre-procesador `cgx` de Calculix.

```
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 7.16.

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.

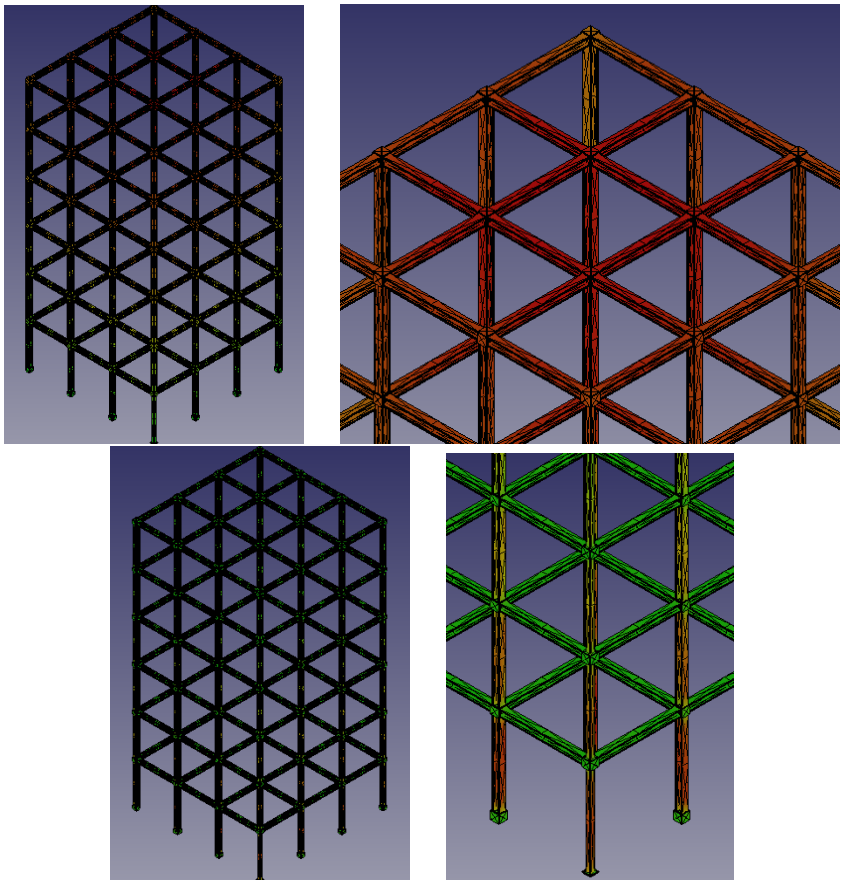


Figura 7.16: Salida gráfica de FreeCAD correspondiente al archivo post-procesado `EsqueletoDeAcero6.frd`: Desplazamiento absoluto (arriba) y Esfuerzo cortante (abajo)

```
** 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, 10235
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 7.17.

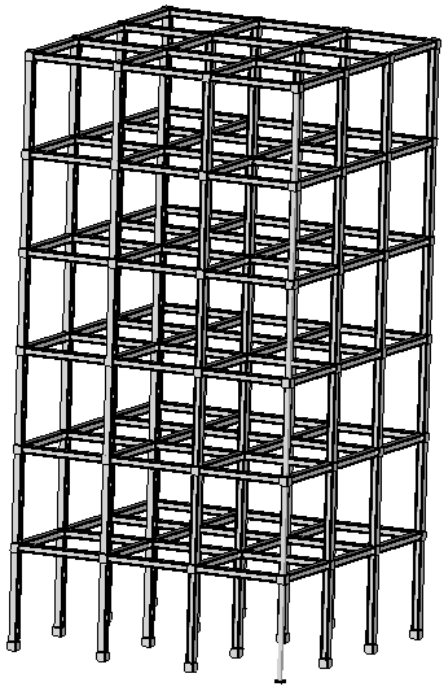


Figura 7.17: Salida gráfica del pre-procesador `cgx` de Calculix.

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 7.18.

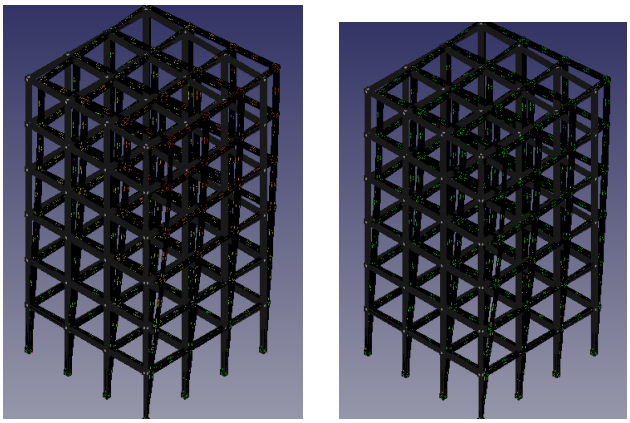


Figura 7.18: Salida gráfica de FreeCAD correspondiente al archivo post-procesado `EsqueletoDeAceroRM6.frd`: Desplazamiento absoluto (izquierda) y Esfuerzo cortante (derecha)

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 8

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.

8.1. 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.

$$\begin{aligned} (\mathbf{M} + \theta\tau\mathbf{A})\mathbf{u}^{n+1} &= \{\mathbf{M} - (1-\theta)\tau\mathbf{A}\}u^n + \tau\left\{\theta\mathbf{f}^{n+1} + (1-\theta)\mathbf{f}^n\right\} \\ \mathbf{M} = (m_{ij}), \quad m_{ij} &= \mathcal{I}(\hat{\mathbf{u}}_i, \hat{\mathbf{u}}_j), \qquad \mathbf{A} = (a_{ij}), \quad a_{ij} = \mathcal{A}(\hat{\mathbf{u}}_i, \hat{\mathbf{u}}_j) \end{aligned} \tag{8.1.1}$$

Los modelos de la forma (8.1.1) son denominados modelos de deformación matricial dinámicos en este documento.

8.2. 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 (7.1.11), 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.

$$\begin{cases} \mathbf{u}_h'(t) = \mathbf{v}_h(t) \\ \mathbf{v}_h'(t) = \frac{1}{\rho_0}\mathbf{N}_{\lambda,\mu}\mathbf{u}_h(t) \\ \mathbf{u}_h(0) = \mathbf{u}_0 \\ \mathbf{v}_h(0) = \mathbf{u}_1 \end{cases} \tag{8.2.1}$$

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 (8.2.1) aproximadamente utilizando un esquema de Crank-Nicolson de la forma:

$$(2\mathbf{1} - h_t\mathbf{M}_{\lambda,\mu})\mathbf{w}_h(t + h_t) = (2\mathbf{1} + h_t\mathbf{M}_{\lambda,\mu})\mathbf{w}_h(t) \tag{8.2.2}$$

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,

$$\begin{cases} \mathbf{u}_h''(t) = \frac{1}{\rho_0}\mathbf{N}_{\lambda,\mu}\mathbf{u}_h(t) \\ \mathbf{u}_h(0) = \mathbf{u}_0 \\ \mathbf{u}_h'(0) = \mathbf{u}_1 \end{cases} \tag{8.2.3}$$

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

$$\begin{cases} \mathbf{u}_h(t+1) - 2\mathbf{u}_h(t) + \mathbf{u}_h(t-1) = \frac{h_t^2}{2\rho_0}\mathbf{N}_{\lambda,\mu}\mathbf{u}_h(t+1) + \frac{1}{2\rho_0}\mathbf{N}_{\lambda,\mu}\mathbf{u}_h(t-1) \\ \mathbf{u}_h(0) = \mathbf{u}_0 \\ \mathbf{u}_h(1) = \mathbf{u}_0 + h_t\mathbf{u}_1 \end{cases} \tag{8.2.4}$$

para $t \geq 1$ (entero).

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

Consideremos un modelo Navier de la forma (7.1.9) para una reducción de orden unidimensional de un material $\mathcal{M}$ libre de divergencia, cuya deformación dinámica estará controlada por la siguiente ecuación.

$$\begin{cases} \mu\nabla^2 u_x + \rho_0\delta\partial_t u_x = \rho_0\partial_t^2 u_x \\ \partial_\eta u(x,t) = u^*(x,t), x \in \partial\mathcal{M} \\ u(x,0) = u_0(x) \\ \partial_t u(x,t) = u_1(x) \end{cases} \tag{8.2.5}$$

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.

8.2.1. Cálculo de ondas materiales en 2D

Consideremos el sistema dinámico determinado por la ecuación de Navier (7.1.8) y la restricción (8.2.5) 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',...%'
    'ColorMap',hsv,'Boundary','off','Colorbar','off','CBTitle','w','Title',['Wave F
    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): epsi
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;
```

8.2.2. Cálculo de ondas materiales en 3D

Consideremos el sistema dinámico determinado por la ecuación de Navier (7.1.8) 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)+

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++;
}
```

8.3. Cálculo de Deformación Dinámica en Mecánica de Fluidos

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

$$\frac{\partial u}{\partial t} + u \cdot \nabla u - \nu \Delta u + \nabla p = 0, \nabla \cdot u = 0 \tag{8.3.1}$$

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

$$\begin{array}{cc} \frac{1}{\tau}(u^{n+1} - u^n \circ X^n) - \nu \Delta u^{n+1} + \nabla p^{n+1} & = 0, \\ \nabla \cdot u^{n+1} & = 0 \end{array} \tag{8.3.2}$$

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 describe 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=e
    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 8.1: 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 describe 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.
 *
```

```
 * 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*
   - 0.212836*(4*(t-.2))^2 + 0.17363*(4*(t-.2))^3 - 0.06254*(4*(t-.2))^4); labe
  border Sminus(t=.45, .2){x=t-.02; y=.25*(2*.8-.3*(4*(.2-t))^2-(0.17735*sqrt(4
     -0.212836*(4*(t-.2))^2 + 0.17363*(4*(t-.2))^3 - 0.06254*(4*(t-.2))^4)); la

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

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

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 Discretiza
             + dt*nu*(dx(u1)*dx(v1) + dy(u1)*dy(v1) //Viscocity term Discretiza
             +     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 8.2. 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
```

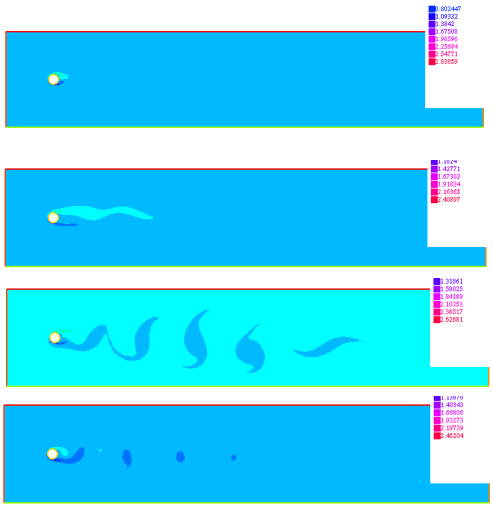


Figura 8.2: Cuatro perfiles de flujo de calle de vórtices de von Kármán alrededor de un obstáculo cilíndrico.

- Esto produce una salida gráfica como la mostrada en la figura 8.3.

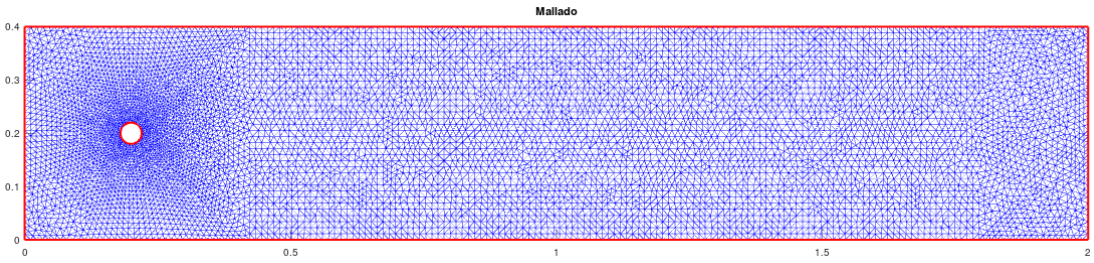


Figura 8.3: Malla de la región de cómputo.

- 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
% *
% * 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',
    'ColorMap',winter,'Boundary','off','CBTitle','w','Title',['Vorticity: \Omeg
    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 8.4. Además los diagramas espectrales de predicción se muestran en la figura 8.5.

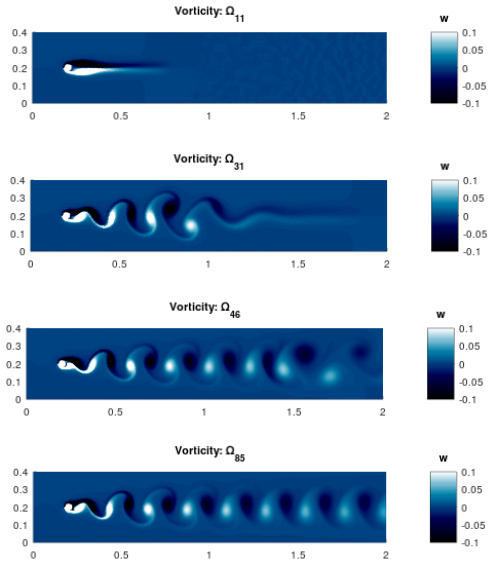


Figura 8.4: Cuatro perfiles de flujo de calle de vórtices de von Kármán alrededor de un obstáculo cilíndrico.

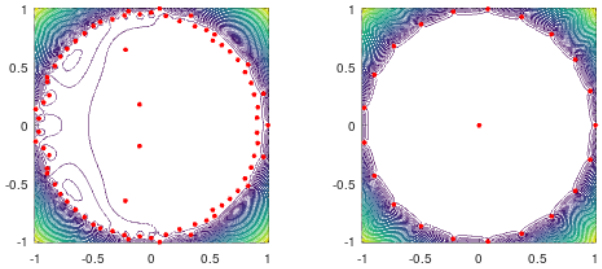


Figura 8.5: Diagramas espectrales de predicción del comportamiento del modelo: Predicción DMD (izquierda) y predicción CCEF (derecha).

Bibliografía

[1] John W. Eaton, David Bateman, Søren Hauberg, Rik Wehbring (2020). GNU Octave version 6.1.0 manual: a high-level interactive language for numerical computations. `https://www.gnu.org/software/octave/doc/v6.1.0/`

[2] I. Markovsky, S. Van Huffel, J. C. Willems, B. De Moor (2005). Exact and Approximate Modeling of Linear Systems: A Behavioral Approach. SIAM.

[3] Quarteroni A., Saleri F., Gervasio P. (2014). Scientific computing with MATLAB and Octave.

[4] S. Boyd, L. Vandenberghe. (2018). Introduction to Applied Linear Algebra Vectors, Matrices, and Least Squares. Cambridge University Press.

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

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

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

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

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

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

[11] 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.

[12] A. S. Householder (1964). The Theory of Matrices in Numerical Analysis. Dover Publications, Inc.

[13] Y. Saad (2003). Iterative Methods for Sparse Linear Systems. 2a Ed. SIAM.

[14] L. N. Trefethen, M. Embree (2005). Spectra and Pseudospectra: The Behavior of Nonnormal Matrices and Operators. Princeton University Press.

[15] R. L. Burden, D. J. Faires, A. M. Burden. (2017). Análisis Numérico. 10a Ed. Cengage Learning Editores.

[16] Golub, G. H., Van Loan C. F (1996). Matrix Computations (3aEd.). The Johns Hopkins University Press.

[17] D. G. Luenberger, Y. Ye. (2016). Linear and Nonlinear Programming. 4a Ed. Springer International Publishing Switzerland.