Primero mostramos algunos ejemplos de códigos y luego se plantean ejercicios computacionales. observar la importacia del coste computacional.
Observación: No usar tildes en los comentarios de los códigos.

Ejemplos:

Observación: Consideramos la secuencia de Fibonacci $n = 2, 3, 5, ..., Ntotal$ para estudiar el comportamiento cuando la talla (dada por los números de Fibonacci) crece. Esto lo hacemos porque es un conjunto de dimensiones que se ajusta bien a una escala loglog y es lo suficientemente fina para mostrar tendencia.

Coste computacional

1. En el código **CosteProductoMatrizVector.m** se contrasta que el producto de una matriz por un vector tiene un coste computacional $O(n^2)$

```
%
% Coste producto de matriz vector.
% Se comprueba usando matrices y vectores con
% entradas aleatorias.
%
clear all;
%
% Talla de la matriz sigue el crecimiento de la
% la secuencia de Fibonacci
%
long= 20; % Numero de valores de la secuencia de Fibonacci
%
n=[2,3,ones(1,long-2)];
for i=3:long
  n(i) = n(i-1)+n(i-2);
end
time=zeros(1,long);
for k=1:long  %Bucle para las muestras
    d=n(k);
    a=rand(d,d);
    b=rand(d,1);
    tic;
    x=a*b;
    time(k)=toc;
    disp(['k= ',num2str(k),' Talla ',num2str(d)]);
end
```

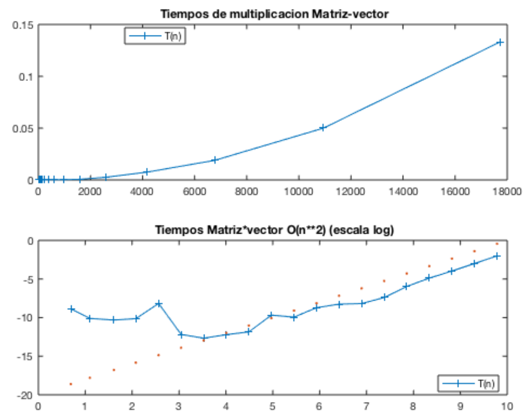

Figura 1: Para cada dimensión $n = 2, 3, 5, ..., F_{20}$ donde F_k es el número de Fibonacci, se calculan los productos matriz-vector y los tiempos

```
figure(1);
subplot(2,1,1)
plot(n,time,'-+');
legend('T(n)','Location','Best');
title(' Tiempos de multiplicacion Matriz-vector');
subplot(2,1,2)
plot(log(n),log(time),'-+',log(n),2.0*log(n)-20,'.');
legend('T(n)','Location','Best');
title(' Tiempos Matriz*vector O(n**2) ');
```

2. En el código **CosteProductoMatrizmatriz.m** se contrasta que el producto de una matriz por una matriz tiene un coste computaciona ligeramente mejor que la estimación $O(n^3)$; se obtiene $O(n^{2.7})$.

```
%
% Coste producto de matrices
%
clear all;
%
% Numero de puntos de la secuencia de Fibonacci
%
long= 18;

n=[2,3,ones(1,long-2)];
for i=3:long
```

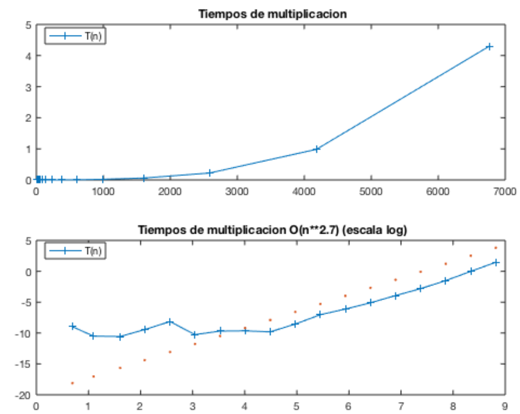

Figura 2: Para cada dimensión $n = 2, 3, 5, ..., F_{20}$ donde F_k es el número de Fibonacci, se calculan los productos matriz-matriz y los tiempos

```
    n(i) = n(i-1)+n(i-2);
  end
  time=zeros(1,long);
  for k=1:long  %Bucle para las muestras
      d=n(k);
      a=rand(d,d);
      b=rand(d,d);
      tic;
      x=a*b;
      time(k)=toc;
      disp(['k= ',num2str(k),' Talla ',num2str(d)]);
  end
    figure(1);
    subplot(2,1,1)
    plot(n,time,'-+');
    legend('T(n)','Location','Best');
    title(' Tiempos de multiplicacion');
    subplot(2,1,2)
    plot(log(n),log(time),'-+',log(n),2.7*log(n)-20,'.');
    legend('T(n)','Location','Best');
    title(' Tiempos de multiplicacion O(n**2.7) (escala log)');
```

Uso de Cramer

El siguiente codigo **determinante.m** computa el determinante de una matriz A de forma recursiva y usando el desarrollo por la primera fila de A

```
%
% Se obtiene el determinante de A desarrollando
% por la fila primera de A.
%
function det=determinante(A)
  N=size(A);
  if (N(1)~=N(2))
    error('Matriz A no cuadrada');
  end
  if (N(1)==1)
    det=A(1,1);
  else
      % N(1)>1
    det=0;
    B=A;
    B(1,:)=[];% quitamos la fila 1 de A
    for j=1:N(1)
     C=B;
     C(:,j)=[];% quitamos la columna j de C
     det=det + A(1,j)* (-1)^(1+j) * determinante(C);
    end
  end
end
```

Usando la regla de Cramer, aqui resolvemos formalmente un sistema lineal

```
function s=solveCramer(A,b)
  N=size(A);
  if (N(1)~=N(2))
    error('Matriz A no cuadrada');
  elseif (N(1)~=length(b))
    error('dimensiones incompatibles');
  end
  s=zeros(1,N(1));
  denom=determinante(A);
  if (abs(denom)< 1e-12)
    error("Matriz singular");
  end
  for j=1:N(1)
```

```
    B=A;
    for i=1:N(1)
      B(i,j)=b(i);
    end
    % Mas eficiente hacer B(:,j)=b;
    s(j)=determinante(B)/denom;
  end
end
```

En **ejemploCramer1.m** resolvemos varios sistemas mediante la regla de Cramer y usando también el comando Matlab/Octave. Medimos tiempos y exploramos al sensibilidad de los sistemas lineales. Usamos el ejemplo clasico de R.S.Wilson (ver Infante-Rey pag 37 por ejemplo).

```
%
% Ejemplo de uso de la regla de Cramer
% Usamos el ejemplo clasico de R.S.Wilson
% Ver Infante-Rey pag 37 por ejemplo.
%
clear all;
b=[32,23,33,31];
b
A=[10,7,8,7;
   7,5,6,5;
   8,6,10,9;
   7,5,9,10];
A
fprintf('\n Solucion Cramer \n \n');
tic;
x=solveCramer(A,b);
toc;
fprintf('solucion x');
x
res=b'- A*x';
fprintf('norma residual || b-Ax|| = %d',norm(res));
fprintf(' \n');
fprintf('\n Solucion Octave/Matlab \n \n');
tic;
xC= A\b';
toc;
x=xC';
fprintf('solucion x');
x
res=b'- A*x';
```

```
fprintf('norma residual || b-A*x|| = %d',norm(res));
fprintf(' \n');
 % Perturbacion en termino independiente
 fprintf('\n *** Perturbacion en termino independiente b1(i)= b(i)+/- epsilon \n ');
 epsilon=0.1;
 b1=[32+epsilon,22-epsilon,33+epsilon,30-epsilon];
 fprintf('b1');
 b1
 fprintf('\n Solucion Cramer \n \n');
 tic;
 x1=solveCramer(A,b1);
 toc;
 fprintf('solucion x1');
 x1
 res1=b1'- A*x1';
 fprintf('norma residual || b1-A*x1|| = %d',norm(res1));
 fprintf(' \n');
 fprintf('\n Solucion Octave/Matlab \n \n');
 tic;
 x1C= A\b1';
 toc;
 x1=x1C';
 fprintf('solucion x1');
 x1
 res1=b1'- A*x1';
 fprintf('norma residual || b1-A*x1|| = %d',norm(res1));
 fprintf(' \n');
  % Perturbacion en matriz
 fprintf('\n *** Perturbacion en matriz A, A1=A+DeltaA \n');
  DeltaA=[0,0,0.1,0.2;0.08,0.04,0,0;0,-0.02,-0.11,0;-0.01,-0.01,0,-0.02];
  A1=A+DeltaA;
  fprintf('A1');
  A1
  fprintf('\n Solucion Cramer \n \n');
  tic;
  x2=solveCramer(A1,b);
  toc;
  fprintf('solucion x2');
  x2
  res2=b'- A1*x2';

  fprintf('norma residual || b-A1 x2|| = %d',norm(res2));
  fprintf(' \n');
```

```
  fprintf('\n Solucion Octave/Matlab \n \n');
  tic;
  x2C= A1\b';
  toc;
  x2=x2C';
  fprintf('solucion x2');
  x2
  res=b'- A1*x2';
  fprintf('norma residual || b-A1*x2|| = %d',norm(res));
  fprintf(' \n');
```

En el siguiente ejemplo **ejemploCramer2.m** se realiza una comparativa de tiempos y metodos de resolucion usando las matrices de Hilbert

```
clear all;
n=8;
fprintf(' \t n = %d',n);
fprintf('\n');

% Consulta en la documentacion de Octave/Matlab las definiciones
% de las matrices de Hilbert  hilb(n) y sus inversas invhilb(n)
H=hilb(n);
Hinv=invhilb(n);
fprintf('H(n)* H^(-1)(n) = \n');
H*Hinv
%
b=zeros(1,n);
for i=1:n
  for j=1:n
    b(i)=b(i)+H(i,j);
  end
end
fprintf('\n Solucion Cramer \n \n');
tic;
x=solveCramer(H,b);
toc;
fprintf('solucion x');
x
res=b'- H*x';
fprintf('norma residual || b-Hx|| = %d',norm(res));
fprintf(' \n');
fprintf('\n Solucion Octave/Matlab \n \n');
tic;
xC= H\b';
```

```
toc;
x=xC';
fprintf('solucion x');
x
res=b'- H*x';
fprintf('norma residual || b-Hx|| = %d',norm(res));
fprintf(' \n');
% Perturbacion en termino independiente
fprintf("\n *** Perturbacion en termino independiente b1(i)= b(i)+/- epsilon \n ");
epsilon=0.5e-5;
b1=zeros(1,n);
signo=1;
for i=1:n
    b1(i)=b(i)+epsilon *signo;
    signo=-signo;
end

fprintf('\n Solucion Cramer \n \n');
tic;
x1=solveCramer(H,b1);
toc;
fprintf('solucion x1');
x1
res1=b1'- H*x1';
fprintf('norma residual || b1-H x1|| = %d',norm(res1));
fprintf(' \n');
fprintf('\n Solucion Octave/Matlab \n \n');
tic;
x1C= H\b1';
toc;
x1=x1C';
fprintf('solucion x1');
x1
res1=b1'- H*x1';
fprintf('norma residual || b1-Hx1|| = %d',norm(res1));
fprintf(' \n');
```

Ejercicios prácticos:

1. (AllaireKaber2.1) Fijamos la dimension $n \geq 2$

   a) Indicar qué es el vector u en términos de la matriz a definida por las instrucciones $a = eye(n, n); u = a(:, i)$ para un entero i tal que $1 \leq i \leq n$

   b) La orden $rand(n, m)$ devuelve una matriz de talla $n \times m$ cuyas entradas son números reales en $[0, 1]$. Para n fijo, definir dos vectores $u = rand(n, 1)$ y $v = rand(n, 1)$. Calcular el vector

   $$w = v - \frac{< v, u >}{\|u\|_2^2} u$$

   y el producto escalar $< w, u >$ (el producto escalar $< w, u >$ en Matlab se obtiene usando $u' * v$ o bien usando la función dot).

   c) Sea A una matriz real cuadrada inicializada por $rand(n, n)$; definir dos matrices B y C dadas por $B = 0.5 * (A + A')$ y $C = 0.5 * (A - A')$.

      1) Calcular el producto escalar $< Cx, x >$ para varios vectores x y justificar el resultado.

      2) Calcular el producto escalar $< Bx, x >$ para varios vectores x y justificar el resultado. Comprobar que coincide con $< Ax, x >$ y explicar porqué.

2. Usamos el ejemplo clasico de R.S.Wilson (ver Infante-Rey pag 37 por ejemplo). Consideramos

   $$A = \begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix}, \quad b = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix}.$$

   La solución de $Ax = b$ es $x = (1, 1, 1, 1)^t$. Usando las perturbaciones

   $$\tilde{A} = \begin{pmatrix} 10 & 7 & 8.1 & 7.2 \\ 7.08 & 5.04 & 6 & 5 \\ 8 & 5.98 & 9.89 & 9 \\ 6.99 & 4.99 & 9 & 9.98 \end{pmatrix}, \quad \tilde{b} = \begin{pmatrix} 32.1 \\ 22.9 \\ 33.1 \\ 30.9 \end{pmatrix}.$$

   Calcular las soluciones de los sistemas lineales $\tilde{A}y = b$ y $Az = \tilde{b}$ y observar numéricamente la estabilidad del problema.

3. (AK2.2) Definir las siguientes funciones que devuelven matrices con propiedades especiales. Estas funciones se usarán más adelante. Todas las variables hay que inicializarlas mediante la función $rand$.

   a) Escribir una función de nombre **SymmetricMat(n)** que devuelva una matriz real simétrica de talla $n \times n$. Puedes usar la función de octave rand(n,n) que devuelve una matriz aleatoria, luego quedarte con la parte triangular inferior con tril() y acaba sumando esta con su traspuesta.

b) Escribir una función de nombre **NonsingularMat(n)** que devuelva una matriz real no singular de talla $n \times n$.

c) Escribir una función de nombre **LowNonsingularMat(n)** que devuelva una matriz real no singular de talla $n \times n$ y triangular inferior.

d) Escribir una función de nombre **UpNonsingularMat(n)** que devuelva una matriz real no singular de talla $n \times n$ y triangular superior.

e) Escribir una función de nombre **ChanceMat(m,n,p)** que devuelva una matriz real de talla $m \times n$ con entradas aleatorias en el intervalo $[-p, p]$.

4. (AK2.3) Definir una matriz A por las instrucciones

$$p = NonsingularMat(n); A = p * diag([ones(n-1,1); r]) * inv(p)$$

donde r es un número real cualquiera. ¿Cual es el determinante de A? (No usar Matlab para responder). Si tomamos $r = 10^{-20}, n = 5$ y calculamos con Matlab el determinante de A, ¿qué se observa?

5. (AK2.6) Fijando $n = 5$

a) para cualquier entero r tal que $1 \leq r \leq 5$ inicializar con $rand$ r vectores u_i y definir la matriz $A = \sum_{i=1}^{r} u_i u_i^t$. Comparar el rango de A con r.

b) Justificar los resultados.

6. (AK2.22) Para varios valores de n calcular el rango de $A = rand(n,1) * rand(1,n)$. Que se observa? Vamos a probar el resultado:

a) Sean $u, v \in \mathbb{R}^n$ dos vectores no nulos. ¿Cual es el rango de $A = vu'$?

b) Sea $A \in \mathbb{R}^{n \times n}$ de rango 1. Comprobar que existen dos vectores $u, v \in \mathbb{R}^n$ tales que $A = vu'$.

7. (AK2.24) Para cada una de las matrices

$$A_1 = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 4 & 2 & 1 \end{pmatrix}, \ A_2 = \begin{pmatrix} 0.75 & 0 & 0.25 \\ 0 & 1 & 0 \\ 0.25 & 0 & 0.75 \end{pmatrix}$$

$$A_3 = \begin{pmatrix} 0.375 & 0 & -0.125 \\ 0 & 0.5 & 0 \\ -0.125 & 0 & 0.375 \end{pmatrix}, \ A_4 = \begin{pmatrix} -0.25 & 0 & -0.75 \\ 0 & 1 & 0 \\ -0.75 & 0 & -0.25 \end{pmatrix}$$

calcular A_i^n para $n = 1, 2, 3, ...$ ¿Cual parece ser el límite de A_i^n? Justificar la respuesta.