Capítulo 1

Resolución de sistemas de ecuaciones

1.1. Introducción

Este tema trata de aproximar la solución de sistemas de ecuaciones

$$A\mathbf{x} = \mathbf{b}$$

mediante:

- métodos directos que, si se utilizara aritmética exacta, producirían la solución exacta
- métodos iterativos
- otros métodos para sistemas de ecuaciones no lineales

1.2. Operador \

La resolución de sistemas de ecuaciones lineales en Matlab se realiza a través del operador \.

Nota 1 Los sistemas lineales también pueden ser resueltos con la instrucción **linsolve**, que permite acelerar la resolución cuando se conoce alguna propiedad especial de la matriz.

Ejercicio 1 *Resolver el sistema*

$$\left.\begin{array}{rcl} 4x_1 + 3x_2 & = & 24 \\ 3x_1 + 4x_2 - x_3 & = & 30 \\ -x_2 + 4x_3 & = & -24 \end{array}\right\}$$

1.3. Almacenamiento de matrices

Antes de comenzar a resolver sistemas de ecuaciones lineales es conveniente conocer la sintaxis utilizada en MatLab para el manejo de sistemas de ecuaciones y, consecuentemente, de matrices y ver que hay dos tipos fundamentales de matrices (Full y Sparse).

Matrices Full

La forma habitual de introducir las matrices de los sistemas mediante corchetes y ;.

Matrices Sparse

Por matrices dispersas o huecas (en inglés, "sparse") se conocen a aquellas con un gran número de elementos nulos. Aparecen frecuentemente en la resolución numérica de ecuaciones en derivadas parciales (método de elementos finitos, diferencias finitas...). El objetivo del tratamiento de estas matrices es poder trabajar sólo con los elementos no nulos sin necesidad de almacenar toda la matriz. Las técnicas de almacenamiento se basan en guardar en vectores los elementos no nulos y sus posiciones relativas en la matriz global. La orden "sparse" para almacenamiento de matrices dispersas.

Función	**Salida**
S= sparse (i,j,c,m,n)	**m :** número de filas de la matriz. **n :** número de columnas de la matriz. **c :** vector de elementos no nulos de la matriz. **i,j :** vectores que indican la fila y columna del elemento del vector **c**.
S= sparse (A)	Convierte una matriz llena **A** en una dispersa **S**
A= full (S)	Convierte una matriz dispersa **S** en una llena **A**

Ejercicio 2 *Sea la matriz*

$$\begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ 0 & -4 & 3 & 0 & 0 \\ 9 & 0 & 7 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \end{pmatrix}$$

Almacenarla de manera llena y dispersa.

Ejercicio 3 *Sea M la matriz tridiagonal 31×31 con el número 2 en la diagonal principal y -1 en las otras dos diagonales y sea b el vector de orden 31×1 con todas sus componentes igual a 1.*

1. *Almacena en la variable **A**, la matriz M llena, en la variable **S**, la matriz M dispersa y en la variable **b** el vector b.*
2. *Calcula el tiempo empleado en resolver el sistema $A\mathbf{x} = \mathbf{b}$ diez mil veces, según se considere la matriz como llena o dispersa.*

1.4. Métodos directos de resolución

Vemos a continuación algunos de los métodos directos de resolución de sistemas entre los que destacaremos el de Gauss, con alguna de sus variantes.

1.4.1. Método de Gauss

Se considera un sistema de ecuaciones lineales de orden $n \times n$ descrito por la matriz $\mathbf{A}$ y el vector $\mathbf{b}$.

$$A\mathbf{x} = \mathbf{b}$$

La eliminación gaussiana consiste en obtener, mediante operaciones elementales, un sistema equivalente al anterior

$$U\mathbf{x} = \mathbf{c}$$

siendo U una matriz triangular superior.

Nota 2 Para la resolución del sistema es necesario por tanto hacer una sustitución regresiva y comenzar por la última incógnita hasta finalizar por la primera.

Resolución del sistema triangular superior:

Algoritmo de sustitución hacia atrás

$$x_n = \frac{b_n}{a_{n\,n}}$$

$$x_k = \frac{1}{a_{k\,k}} \left(b_k - \sum_{j=k+1}^{n} a_{k\,j}\ x_j \right) \qquad k = n-1, \cdots, 1$$

Método de Gauss para resolver un sistema $Ax = b$

```
function [x] = gauss(A,b)
[m n] = size (A);
if m ~= n, error( ´ Matriz del sistema NO Cuadrada´ ), end
if m ~= length(b), error( ´ Sistema NO Coherente´), end
% transformacion del sistema en uno triangular
for i=1:n-1
    for k=i+1:n
        l=A(k,i)/A(i,i);
        for j=i+1:n
            A(k,j)=A(k,j)-l*A(i,j);
        end
        b(k)=b(k)-l*b(i);
    end
end
A=triu(A);
x=resolución_triangular(A,b);
```

Ejercicio 4 *Construye la función* `resolución_triangular` *para la resolución de un sistema triangular superior.*

Estrategias de pivoteo

Si todos los cálculos fuesen en aritmética exacta todo iría sobre ruedas pero en situaciones prácticas no se tiene e, incluso en sistemas de pequeño tamaño, debemos considerar los errores debidos a los redondeos.

Para paliar, en parte, este problema se pueden ordenar los cálculos intercambiando filas e, incluso columnas en las matrices que intervienen en el sistema. Lo habitual es el intercambio de filas cuando el elemento pivote es nulo, pero existen otras estrategias de pivoteo en las que, aunque no sea nulo, se hace un intercambio.

Pivoteo parcial Esta es la estrategia mas utilizada en la práctica y consiste en buscar, en cada paso, por debajo del pivote, y en su columna, el elemento de módulo máximo y se procede a la permutación de las filas correspondientes para posteriormente realizar la anulación de los elementos de la columna.

Método de Gauss con pivoteo para resolver un sistema $Ax = b$

```
function [x] = gausspivoteo(A,b)
% Método de Gauss con pivoteo para resolver A x = b
% sistema lineal de ecuaciones cuadrado
[m n] = size (A);
if m ~= n, error( ´ Matriz del sistema NO Cuadrada´ ), end
if m ~= length(b), error( ´ Sistema NO Coherente´), end
for k = 1:n
   [p r]=max(abs(A(k:n,k)));
   r = r+k-1;
   if p < eps, disp('Matriz de coeficientes singular'), return, end
   if k~=r
        A([k r],:)=A([r k],:);
        b([k r])=b([r k]);
   end
   for i=k+1:n
        z=A(i,k);
        A(i,k:n)=A(i,k:n)- (z/A(k,k))*A(k,k:n);
        b(i)=b(i)- (z/A(k,k))*b(k);
   end
end
A=triu(A);
x = resolución_triangular(A,b);
```

Ejercicio 5 *Se considera el sistema* $\left.\begin{array}{rcrcrcl} 10^{-15} & x & + & y & = & 1 \\ & x & + & y & = & 2 \end{array}\right\}$.

1. *Almacenar en la variable* ***s*** *la solución del sistema proporcionada por MATLAB.*
2. *Almacenar en las variables* ***s1*** *y* ***s2*** *las soluciones del sistema obtenidas mediante el método de Gauss y con pivoteo, respectivamente.*
3. *Almacenar en las variables* ***e1*** *y* ***e2*** *el error relativo cometido al aproximar la solución* ***s*** *por* ***s1*** *y* ***s2*** *respectivamente. Razona por qué es menor* ***e2****.*

1.4.2. Método de Cramer

Este es otro método de resolución basado en fórmulas en las que intervienen los determinantes de matrices en las que se cambian las columnas de la matriz asociada por el vector **b**.

La regla de Cramer es de importancia teórica porque da una expresión explícita para la solución del sistema. Sin embargo, para sistemas de ecuaciones lineales de más de tres ecuaciones su aplicación para la resolución del mismo resulta excesivamente costosa: computacionalmente, es ineficiente para grandes matrices y por ello no es usado en aplicaciones prácticas que pueden implicar muchas ecuaciones. Sin embargo, como no es necesario pivotar matrices, es más eficiente que la eliminación gaussiana para matrices pequeña.

Si $\mathbf{Ax} = \mathbf{b}$ es un sistema de ecuaciones. A es la matriz de coeficientes del sistema, $\mathbf{x} = (x_1, \ldots, x_n)$ es el vector columna de las incógnitas y **b** es el vector columna de los términos independientes. Entonces la solución al sistema se presenta así:

$$x_j = \frac{\det(A_j)}{\det(\mathbf{A})}$$

donde A_j es la matriz resultante de reemplazar la j-ésima columna de A por el vector columna **b**. Hágase notar que para que el sistema sea compatible determinado, el determinante de la matriz A ha de ser no nulo.

Ejercicio 6 *Programar el método.*

1.4.3. Factorización matricial

Se pretende hacer una factorización de la matriz **A** en un producto de matrices para una resolución más sencilla del sistema de ecuaciones.

Factorización LU

Con MatLab la tarea de encontrar la matriz permutación del teorema anterior y las matrices de la factorización se realiza con la orden lu.

Función	**Salida**
[L,U,P] = lu (A)	Cálculo de la factorización **PA= LU**

Nota 3 La resolución del sistema usando la factorización $PA = LU$ se realiza transformando el sistema original en $PAx = Pb$ y descomponiendo éste en dos sistemas triangulares:

- $Ly = Pb$ (fase de sustitución hacia delante). Se obtiene y
- $Ux = y$ (fase de sustitución hacia atrás). Se obtiene x

verificándose que

$$PAx = LUx = Ly = Pb$$

Ejercicio 7 *Calcula las matrices L, U y P para* $A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 2 \\ 0 & 1 & 1 \end{pmatrix}$. *Verifica que* $PA = LU$.

Ejercicio 8 *Programación de la resolución de los sistemas triangulares resultantes si no se les ha pedido ya en el ejercicio 4.*

Cholesky

La factorización de Cholesky consiste en descomponer una matriz simétrica definida positiva en el producto de una matriz triangular inferior y la traspuesta de la matriz triangular inferior. El resultado de Cholesky ha sido extendido a matrices con entradas complejas. Es una manera de resolver sistemas de ecuaciones matriciales y se deriva de la factorización LU con una pequeña variación.

Cuando es aplicable, la descomposición de Cholesky es dos veces más eficiente que la descomposición LU.

Ejercicio 9 *Programar el método. Y, si no, programar, si no se ha hecho en los ejercicios 4 ó 8, la resolución de los sistemas triangulares.*

Ejercicio 10 *Resolver el sistema de ecuaciones*

$$\begin{aligned} 4x - y + z &= 4 \\ -x + 4{,}25y + 2{,}75z &= 6 \\ x + 2{,}75y + 3{,}5z &= 7{,}25 \end{aligned}$$

mediante el método de Cholesky.

1.4.4. Número de condición

El número de condición de una matriz nos va a permitir determinar cuánto cambia la solución de un sistema $A\mathbf{x} = \mathbf{b}$ al realizar una pequeña modificación en los coeficientes de A ó $\mathbf{b}$. Esta modificación ocurre por ejemplo cuando los datos provienen de medidas experimentales (contienen errores) o cuando los números son aproximados por números en punto flotante. Para estudiar teóricamente ese problema es necesario introducir el concepto de norma matricial.

Ejemplo 1 La norma vectorial $||(x_1, \cdots, x_n)||_\infty = \sup \{ |x_1|, \cdots, |x_n| \}$ induce la norma matricial:

$$||A||_\infty = \sup_{||\mathbf{x}||_\infty = 1} \{ ||A\mathbf{x}||_\infty \} = \sup_{i=1,\cdots,n} \{ |a_{i\,1}| + \cdots + |a_{i\,n}| \}$$

Ejemplo 2 La norma vectorial euclídea $||(x_1, \cdots, x_n)||_2 = \sqrt{x_1^2 + \cdots + x_n^2}$ induce la norma matricial:

$$||A||_2 = \sup_{||\mathbf{x}||_2 = 1} \{ ||Ax||_2 \} = \sqrt{\rho(A^t A)}$$

con $\rho(B)$ **radio espectral de** B, es decir, el máximo de los módulos de los autovalores de B.

MatLab tiene órdenes para obtener algunas normas matriciales:

Función	Salida
norm (A)	Calcula la 2-norma de **A**
norm (A, inf)	Calcula la ∞-norma de **A**

Ejercicio 11 *Obtener las normas de las matrices* $A = \begin{pmatrix} 0{,}832 & 0{,}448 \\ 0{,}784 & 0{,}421 \end{pmatrix}$ *y*

$B = \begin{pmatrix} 0{,}832 & 0{,}448 \\ 0{,}784 & 0{,}422 \end{pmatrix}$

Nota 4

- El número de condición es una cota de la amplificación que se puede producir por errores en los datos. Si $cond(A) >> 1$ significa que es posible que la solución del problema perturbado sea muy mala.
- Se cumple que $cond(A) \geq 1$ (pues $1 = ||I|| = ||A.A^{-1}||.||A||||A^{-1}||$) y $cond(A) = cond(A^{-1})$.

El número de condición, con MatLab, se puede estimar mediante las órdenes:

Función	Salida
cond (A)	Calcula el número de condición en la 2-norma de **A**
cond (A, inf)	Calcula el número de condición en la ∞-norma de **A**
condest (A)	Estimación del número de condición en la 1-norma de **A**
normest (A)	Estimación de la 2-norma de **A**
eig (A)	Calcula los autovalores de **A**

Ejercicio 12 *Obtener los números de condición de las matrices* $A = \begin{pmatrix} 0{,}832 & 0{,}448 \\ 0{,}784 & 0{,}421 \end{pmatrix}$ *y*

$B = \begin{pmatrix} 0{,}832 & 0{,}448 \\ 0{,}784 & 0{,}422 \end{pmatrix}$

Ejercicio 13 *Sea* $A = \begin{pmatrix} 0{,}832 & 0{,}448 \\ 0{,}784 & 0{,}421 \end{pmatrix}$ *y* $\mathbf{b} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$*:*

1. *Resuelve el sistema* $A\mathbf{x} = \mathbf{b}$ *y llama* $\mathbf{x}_e$ *a su solución.*
2. *Aumenta en* 0,001 *el coeficiente* a_{22} *de* A *y llama* A_m *a la matriz modificada.*
3. *Resuelve el sistema* $A_m\mathbf{x} = \mathbf{b}$ *y llama* $\mathbf{x}_m$ *a su solución.*
4. *Calcula el error relativo (en la 2-norma) cometido al aproximar* $\mathbf{x}_e$ *por* $\mathbf{x}_m$*.*
5. *Dar una estimación del número de condición de la matriz A.*

6. *Calcula* ***cond(A)*** *y verifica que es mayor que el cociente entre el máximo y el mínimo del valor absoluto de los autovalores.*

Ejercicio 14 *Resuelve el sistema*

$$\begin{aligned} x + \frac{1}{2}y &= \frac{11}{24} \\ \frac{1}{2}x + \frac{1}{3}y &= \frac{1}{4} \end{aligned}$$

1.5. Métodos iterativos de resolución

Consisten en expresar la solución del sistema $A\mathbf{x} = \mathbf{b}$ como el punto fijo de alguna función de iteración lineal:

$$\mathbf{x} = \mathbf{g}(\mathbf{x}) = \mathbf{B}\mathbf{x} = \mathbf{c}$$

La solución se obtiene aplicando el método de búsqueda de punto fijo:

$$x^{(0)} \in \mathbb{R}^n \text{ dado}$$
$$x^{(k+1)} = Bx^{(k)} + c$$

La matriz y el vector son los que van a caracterizar los métodos que vamos a explicar.

Nota 5 Vamos a utilizar la siguiente notación:

- $A = (a_{ij})$ matriz inicial del sistema.
- $b = (b_i)$ vector de segundos miembros.
- $D = (d_{ij})$ diagonal de A, es decir, $d_{ii} = a_{ii}$ y $d_{ij} = 0$ si $i \neq j$.
- $L = (l_{ij})$ triangular inferior de A, es decir, $l_{ij} = a_{ij}$ si $i > j$ y $l_{ij} = 0$ si $i \leq j$
- $U = (u_{ij})$ triangular superior de A, es decir, $u_{ij} = a_{ij}$ si $i < j$ y $u_{ij} = 0$ si $i \geq j$

con lo que se podrá descomponer la matriz A en la forma

$$A = D + L + U$$

1.5.1. Método de Jacobi

La elección de la matriz B y del vector c es de la siguiente forma

$$x^{(k+1)} = \underbrace{-D^{-1}(L+U)}_{B_J} x^{(k)} + \underbrace{D^{-1}b}_{c_J}$$

Para asegurar que existe D^{-1} es necesario que a_{ii} sea no nulo para todo i. Si se desarrolla componente a componente esta etapa, se obtiene la relación

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\Big(\sum_{j=1}^{i-1} -a_{ij}x_j^{(k)} + \sum_{j=i+1}^{n} -a_{ij}x_j^{(k)} + b_i\Big)$$

ALGORITMO

Datos iniciales A, **b**, $x^{(0)}$, tol, $Maxiter$

Iteración k Dado $x^{(k)}$ se calcula $x^{(k+1)}$

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\Big(\sum_{j=1}^{i-1} -a_{ij}x_j^{(k)} + \sum_{j=i+1}^{n} -a_{ij}x_j^{(k)} + b_i\Big)\ i = 1, \ldots, n$$

Test de parada

$$||x^{(k+1)} - x^{(k)}|| \begin{cases} < tol & \text{Parar. Solución } x^{(k+1)} \\ \geq tol \wedge k \leq Maxiter & \text{Siguiente iteración} \\ \geq tol \wedge k > Maxiter & \text{Parar. No converge} \end{cases}$$

Método de Jacobi para resolver un sistema $Ax = b$

```
function x=jacobi(A,b,x0,tol,maxiter)
%     Método de Jacobi para resolver
%                     A x = b
%      sistema lineal de ecuaciones cuadrado
%     x0 .... valor inicial
%   maxiter.... número máximo de iteraciones
%     tol ....  tolerancia permitida

[m n] = size (A);
if m ~= n ,error( ' Matriz del sistema NO Cuadrada' ), end
if m ~= length(b), error( ' Sistema NO Coherente'), end
x=zeros(size(b));
switch  nargin
   case  2, x0=zeros(size(b)); maxiter=100; tol=1.e-5;
   case  3, maxiter=100; tol=1.e-5;
   case  4, maxiter=100;
end

if any(abs(diag(A))<eps)
   error('Método no válido. Elemento diagonal nulo')
end
```

```
for k = 1:maxiter
   for i=1:n
 x(i)=(b(i)-sum(A(i,[1:i-1 i+1:n]).*x0([1:i-1 i+1:n])'))/A(i,i);
   end
   if norm(x-x0) < tol
      fprintf('\n Jacobi CONVERGE en %d iteraciones \n', k),
      return
   end
   x0=x;
end
fprintf('\n Jacobi NO CONVERGE en %d iteraciones \n',maxiter)
```

Ejercicio 15 *Comprueba que el método de Jacobi aplicado al sistema*

$$\left.\begin{array}{rcl} 4x - y + z & = & 7 \\ 4x - 8y + z & = & -21 \\ -2x + y + 5z & = & 15 \end{array}\right\}$$

converge. Comprueba también que si se reordenan las ecuaciones

$$\left.\begin{array}{rcl} -2x + y + 5z & = & 15 \\ 4x - 8y + z & = & -21 \\ 4x - y + z & = & 7 \end{array}\right\}$$

el método no converge. ¿Cómo se puede justificar este comportamiento?

1.5.2. Método de Gauss-Seidel

La elección de la matriz B y del vector c es de la siguiente forma

$$x^{(k+1)} = \underbrace{-(D+L)^{-1}U}_{B_{GS}} x^{(k)} + \underbrace{(D+L)^{-1}b}_{c_{GS}}$$

Puede plantearse como una modificación al método de Jacobi en el sentido de que en el cálculo de $x_i^{(k+1)}$ se utilizan $x_1^{(k+1)}, \ldots, x_{i-1}^{(k+1)}$, (ya calculados en esta iteraci´on) y $x_{i+1}^{(k)}, \ldots, x_n^{(k)}$ (datos de la iteración anterior).

ALGORITMO

Datos iniciales A, **b**, $x^{(0)}$, tol, $Max - iteraciones$

Iteración k Dado $x^{(k)}$ se calcula $x^{(k+1)}$

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\Big(\sum_{j=1}^{i-1} -a_{ij}x_j^{(k+1)} + \sum_{j=i+1}^{n} -a_{ij}x_j^{(k)} + b_i\Big) \; i = 1, \ldots, n$$

Test de parada

$$||x^{(k+1)} - x^{(k)}|| \begin{cases} < tol & Parar. \text{Solución } x^{(k+1)} \\ \geq tol \wedge k \leq Max - iteraciones & \text{Siguiente iteración} \\ \geq tol \wedge k > Max - iteraciones & \text{Parar. No converge} \end{cases}$$

Ejercicio 16 *Programar el método de Gauss-Seidel a partir del de Jacobi.*

Ejercicio 17 *Repetir el ejercicio 15 utilizando el método de Gauss-Seidel.*

1.6. Sistemas de ecuaciones no lineales

Ahora consideraremos un sistema de n ecuaciones y n incógnitas

$$\begin{array}{rcl} f_1(x_1, \ldots, x_n) & = & 0 \\ \vdots & \vdots & \vdots \\ f_n(x_1, \ldots, x_n) & = & 0 \end{array}$$

denotando $\mathbf{f} = (f_1, \ldots, f_n)$ y $\mathbf{x} = (x_1, \ldots, x_n)$, el sistema puede representarse por:

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}$$

Se considera entonces el problema de hallar $\mathbf{r} \in \mathbb{R}^{\mathbf{n}}$ tal que $\mathbf{f}(\mathbf{r}) = \mathbf{0}$.

1.6.1. Método de punto fijo

Al igual que en una variable, el método de punto fijo consiste en hallar una función continua $\mathbf{g}$ de forma que si $\mathbf{g}(\mathbf{r}) = \mathbf{r}$ entonces $\mathbf{f}(\mathbf{r}) = \mathbf{0}$.

A partir de la función de iteración $\mathbf{g}$ se plantea el procedimiento iterativo:

$$\begin{cases} \text{Para } \mathbf{x}^{(0)} \text{ dado} \\ \mathbf{x}^{(m+1)} = \mathbf{g}(\mathbf{x}^{(m)}) \text{ con } m \in \mathbb{N} \end{cases}$$

Programación del método:

```
function [x,iter]= ptofijoN(fun,x0,tol,maxiter)
%         function [x,iter]= ptofijoN(fun,x0,tol,maxiter)
% Método de punto fijo para resolver el
%              sistema n x n : fun(x) = x
%   x  ......  Solución
%   iter.....  Número de iteraciones empleadas
%   x0 ......  Estimación inicial (DIMENSION n x 1)
%   fun .....  Función vectorial (DIMENSION n x 1)
%   tol .....  Tolerancia relativa permitida
%   maxiter .  Numero Máximo de Iteraciones
if nargin == 2 ,  maxiter=500; tol=5e-6; end
if nargin == 3 ,  maxiter=500; end
```

```
for iter=1:maxiter
     x=fun(x0);
     % Test del error:
     if norm(x-x0)<= tol*norm(x0)
        return
     end
     x0=x;
end
fprintf('\n NO converge en %d iteraciones \n',maxiter)
```

Ejercicio 18 *Sea* $\mathbf{g}(x_1, x_2) = ((x_1^2 + x_2^2)/5\ , (2x_1^4 + x_2^4)/9)$. *Comprueba que* $(0,0)$ *y* $(1,2)$ *son puntos fijos de* $\mathbf{g}$ *y aplica el método de punto fijo tomando* $x^0 = y^0 = 1/2$.

Ejercicio 19 *Calcula, mediante el método del punto fijo, un punto de corte entre la parábola* $x^2 - 2x - y = 0$ *y la elipse* $x^2 + 4y^2 - 4 = 0$ *(Sugerencia: Utiliza*

$$\mathbf{g}(x_1, x_2) = \Big((x_1^2 - x_2)/2\ ,\ (-x_1^2 - 4x_2^2 + 8x_2 + 4)/8\Big)$$

como función de iteración en el método de punto fijo).

1.6.2. Método de Newton

Sean $U \subset \mathbb{R}^n$ y $f : U \to \mathbb{R}^n$. Se plantea el problema de hallar $\mathbf{r} \in U$ tal que $\mathbf{f}(\mathbf{r}) = \mathbf{0}$.

Conocida una estimación inicial $\mathbf{x}^{(0)}$ de la solución, el sistema original $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ es sustituido por el sistema lineal:

$$\mathbf{f}(\mathbf{x}^{(0)}) + d\mathbf{f}(\mathbf{x}^{(0)})(\mathbf{x} - \mathbf{x}^{(0)}) = 0$$

dando lugar al siguiente proceso iterativo:

$$\begin{cases} \text{Para } \mathbf{x}^{(0)} \text{ dado} \\ \mathbf{x}^{(m+1)} = \mathbf{x}^{(m)} + [d\mathbf{f}(\mathbf{x}^{(m)})]^{-1}(-\mathbf{f}(\mathbf{x}^{(m)})) \text{ con } m \in \mathbb{N} \end{cases}$$

donde se ha utilizado la notación:

$$d\mathbf{f}(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix} (\mathbf{x})$$

Nota 6 El método tiene convergencia local cuadrática.

Para que el método pueda implementarse, es necesario que para cada $m \in \mathbb{N}$ la matriz $d\mathbf{f}(\mathbf{x}^{(m)})$ sea invertible.

Cada iteración del método puede resultar muy costosa, pues se deben realizar las siguientes tareas:

- Evaluar las n^2 funciones $\frac{\partial f_i}{\partial x_j}$ en $\mathbf{x}^{(m)}$
- Resolver un sistema lineal de orden n.

Desde el punto de vista de la programación, el algoritmo se puede descomponer en dos etapas:

- Cálculo de $\Delta\mathbf{x}^{(m)}$ como la solución del sistema lineal

$$[d\mathbf{f}(\mathbf{x}^{(m)})](\Delta\mathbf{x}^{(m)}) = -\mathbf{f}(\mathbf{x}^{(m)})$$

- Obtención de la nueva estimación $\mathbf{x}^{(m+1)}$ como $\mathbf{x}^{(m+1)} = \mathbf{x}^{(m)} + \Delta\mathbf{x}^{(m)}$.

Con la organización anterior, el vector $\Delta\mathbf{x}^{(m)}$ se interpreta como la corrección que se debe realizar sobre $\mathbf{x}^{(m)}$ para aproximar mejor la solución. Además, se realiza el test de convergencia sobre la norma de $\Delta\mathbf{x}^{(m)}$.

Programación del método:

```
function [x,iter]= newtonN(f,df,x0,tol,maxiter)
%     function [x,fx,iter]= newtonN(f,df,x0,tol,maxiter)
% Método de Newton-Raphson para resolver el
%                   sistema n x n :  fun(x) = 0
%    x  ........    Solución
%    iter.......    Número de iteraciones empleadas
%    f  ........    Función vectorial (DIMENSION n x 1)
%   df  ........    Función Diferencial de f
%    x0 ........    Estimación inicial (DIMENSION n x 1)
%   tol ........    Tolerancia del test de parada
%   maxiter ......  Numero Máximo de Iteraciones
if nargin == 3 , maxiter=500; tol=5e-6; end
if nargin == 4 , maxiter=500; end
for  iter=1:maxiter
     fx0 = f(x0);
     dfx0 = df(x0);
     Delta_x0=dfx0\(-fx0);
     x = x0+Delta_x0;
     if norm(Delta_x0) <= norm(x0) * tol
        return
     else,
        x0=x;
     end
end
fprintf('\n NO converge en %d iteraciones\n',maxiter)
```

Ejercicio 20 *Aplica el método de Newton para resolver el sistema*

$$\left.\begin{array}{rcl} x^2+y^2 & = & 4 \\ xy & = & 1 \end{array}\right\}$$

tomando como estimación inicial el punto $(1/2, 2)$.

Se presenta a continuación un programa del método de Newton en el que solo es necesario proporcionar la función. La diferencial de la misma se obtiene utilizando herramientas de cálculo simbólico.

Programación del método:

```
function [r,iter]= newtonNsym(f,X,x0,tol,maxiter)
%     function [r,iter]= newtonNsym(f,X,x0,tol,maxiter)
% Método de Newton-Raphson para resolver el sistema simbólico:
%                                    fun(X) = 0
%    r  ........ Solución
%    iter....... Número de iteraciones empleadas
%    f  ........ Función vectorial (DIMENSION n x 1)
%                  almacenada como objeto simbólico
%    X  ........ Celda con los nombres de las variables
%                 simbólicas de f. Por ejemplo X={x,y,z}
%    x0 ........ Estimación inicial (DIMENSION n x 1)
%   tol ........ Tolerancia del test de parada
%   maxiter .... Numero Máximo de Iteraciones
if nargin == 3 , maxiter=100; tol=5e-6; end
if nargin == 4 , maxiter=100; end
df=jacobian(f);
for  iter=1:maxiter
     fx0=double(subs(f,X,x0));
     dfx0 =double( subs(df,X,x0));
     r = (x0+dfx0\(-fx0));
     % Test de error
     if norm(r -x0) <= norm(x0) * tol
        return
     else,
        x0=r;
     end
end
fprintf('\n NO converge en %d iteraciones  \n',maxiter)
```

Ejercicio 21 *Repite el ejercicio 20 mediante el método de Newton simbólico.*

Ejercicio 22 *La presión* $\mathbf{p}$ *necesaria para enterrar un plato circular de radio* $\mathbf{r}$ *a una distancia* $\mathbf{d}$ *en un suelo blando y homogéneo puede aproximarse por la ecuación:*

$$p = k_1 e^{k_2 r} + k_3 r$$

siendo $\mathbf{k}_1, \mathbf{k}_2, \mathbf{k}_3$ *constantes, con* $\mathbf{k}_2 > 0$*, que dependen de la distancia* $\mathbf{d}$ *pero no del radio del plato.*

1. *Determina los valores de* $\mathbf{k}_1, \mathbf{k}_2, \mathbf{k}_3$ *sabiendo que para enterrar en fango, a una distancia de* $20cm$ *tres platos de radios* 1, 2 *y* $3cm$ *se requieren, respectivamente, presiones de* $10N/cm^2$, $12N/cm^2$ *y* $15N/cm^2$

2. *Usa los resultados del apartado anterior para calcular el tamaño mínimo de un plato circular que sea capaz de sostener una carga de* $50Kg$ *sobre ese terreno sin hundirse mas de* $20cm$ *(Nota: Tenga en cuenta que la presión ejercida será* $(50 * 9{,}8)/(\pi r^2)$*).*

Variaciones

En el método de Newton se pueden introducir algunas modificaciones con el objeto de evitar o reducir el cálculo de $d\mathbf{f}(\mathbf{x}^{(m)})$, a cambio de perder velocidad de convergencia. Las principales opciones en este sentido son las siguientes:

- Cálculo aproximado de las derivadas parciales

$$\frac{\partial f_i}{\partial x_j}(\mathbf{x}^{(m)}) \approx \frac{f_i(\mathbf{x}^{(m)} + h\mathbf{e}_j) - f_i(\mathbf{x}^{(m)})}{h}$$

  siendo e_j el j-ésimo vector de la base canónica y h un parámetro a elegir.

- Extensión del método de la secante.

$$\frac{\partial f_i}{\partial x_j}(\mathbf{x}^{(m)}) \approx \frac{f_i(\mathbf{x}^{(m)}) - f_i(x_1^{(m)}, \ldots, x_j^{(m-1)}, \ldots, x_n^{(m)})}{x_j^{(m)} - x_j^{(m-1)}}$$

- Métodos Cuasi-Newton. La diferencial $d\mathbf{f}$ sólo se actualiza cada cierto número de iteraciones.

Resolución con MatLab

En MATLAB se pueden resolver sistemas de ecuaciones no lineales a través de la orden fsolve con sus diferentes opciones que se pueden consultar con la ayuda.

Función	**Salida**
X=fsolve(fun,X0)	Resuelve el sistema no lineal **fun(X)=0**, siendo **X0** un punto inicial.

La función **fun** se construye en formato de vector columna igual que las descritas anteriormente.

Ejercicio 23 *Resolver el ejercicio 20 con esta orden fsolve.*