RISOLUZIONE NUMERICA DI SISTEMI NON LINEARI

Esempio:

si vuole risolvere numericamente

$$\begin{cases} x_1^2 - x_2^2 = 1 \\ x_1^2 + x_2^2 - 2x_1 = 3. \end{cases} \tag{1}$$

Definisco:

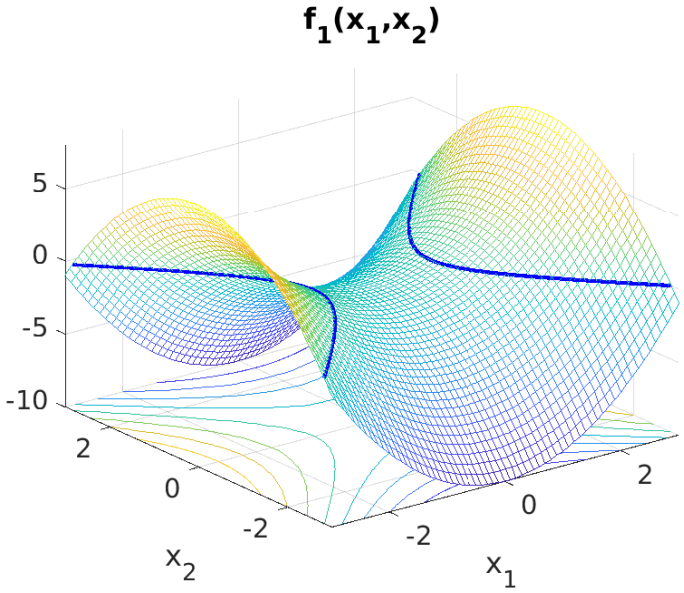
$$f_1(x_1, x_2) = x_1^2 - x_2^2 - 1$$
$$f_2(x_1, x_2) = x_1^2 + x_2^2 - 2x_1 - 3$$

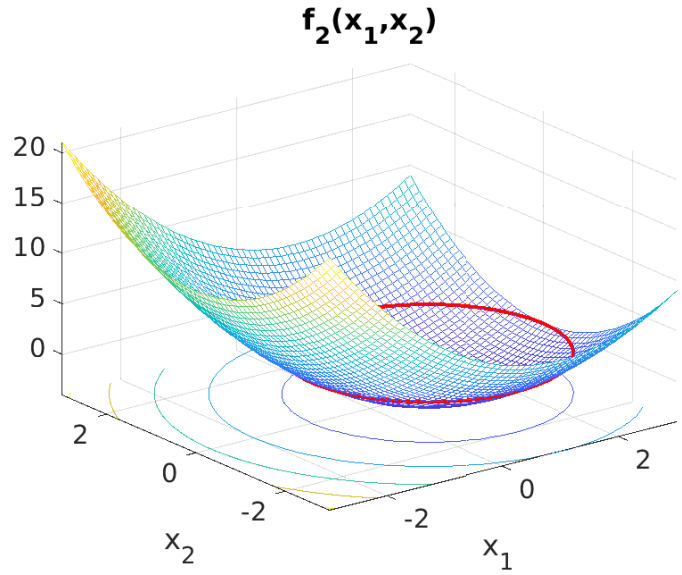
e

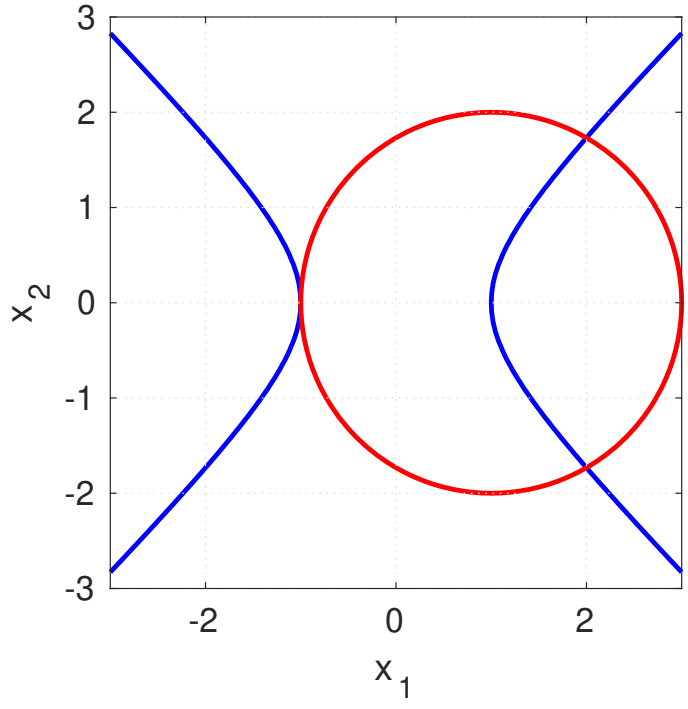
$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{bmatrix}$$

Risolvere (1) vuol dire

$$\text{trovare } \boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} \text{ tale che } \mathbf{F}(\boldsymbol{\alpha}) = \mathbf{0}.$$


$f_1(x_1,x_2)$
5
0
-5
-10
2
0
-2
x_2
-2
0
2
x_1


$f_2(x_1,x_2)$
20
15
10
5
0
2
0
-2
x_2
-2
0
2
x_1


3
2
1
0
-1
-2
-3
x_2
-2
0
2
x_1

Più in generale, per risolvere il sistema

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0 \\ f_2(x_1, x_2, \ldots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n) = 0 \end{cases}$$

si definiscono $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ e la funzione vettoriale

$$\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n : \quad \mathbf{F}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) = f_1(x_1, x_2, \ldots, x_n) \\ f_2(\mathbf{x}) = f_2(x_1, x_2, \ldots, x_n) \\ \vdots \\ f_n(\mathbf{x}) = f_n(x_1, x_2, \ldots, x_n) \end{bmatrix}$$

Si cerca $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_n]^T \in \mathbb{R}^n$: tale che $\mathbf{F}(\boldsymbol{\alpha}) = \mathbf{0}$.

METODO DI NEWTON PER SISTEMI

Ricordo che Newton per equazioni scalari è:

$$\begin{cases} x^{(0)} \text{ dato} \\ x^{(k+1)} = x^{(k)} - \dfrac{f(x^{(k)})}{f'(x^{(k)})}, \quad k = 0, 1, ... \end{cases}$$

Per lavorare con funzioni vettoriali $\mathbf{F}(\mathbf{x})$ devo sostituire: $f'(x^{(k)})$ con la matrice Jacobiana di $\mathbf{F}$ valutata in $\mathbf{x}^{(k)}$

$$J_F(\mathbf{x}^{(k)}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}^{(k)}) & \frac{\partial f_1}{\partial x_2}(\mathbf{x}^{(k)}) & \dots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}^{(k)}) \\ \frac{\partial f_2}{\partial x_1}(\mathbf{x}^{(k)}) & \frac{\partial f_2}{\partial x_2}(\mathbf{x}^{(k)}) & \dots & \frac{\partial f_2}{\partial x_n}(\mathbf{x}^{(k)}) \\ \dots & \dots & \ddots & \dots \\ \frac{\partial f_n}{\partial x_1}(\mathbf{x}^{(k)}) & \frac{\partial f_n}{\partial x_2}(\mathbf{x}^{(k)}) & \dots & \frac{\partial f_n}{\partial x_n}(\mathbf{x}^{(k)}) \end{bmatrix}$$

e reinterpretare la divisione $1/f'(x^{(k)})$ come calcolo della matrice inversa $J_F^{-1}(\mathbf{x}^{(k)})$.

Allora il metodo di Newton per equazioni scalari

$$\begin{cases} x^{(0)} \text{ dato} \\ x^{(k+1)} = x^{(k)} - \dfrac{f(x^{(k)})}{f'(x^{(k)})}, \quad k \geq 0 \end{cases}$$

diventa, per equazioni vettoriali:

$$\begin{cases} \mathbf{x}^{(0)} \text{dato} \\ \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - J_F^{-1}(\mathbf{x}^{(k)})\mathbf{F}(\mathbf{x}^{(k)}) \quad k \geq 0 \end{cases} \tag{2}$$

Il problema cruciale è: come calcolare il vettore
$\mathbf{z} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} = -J_F^{-1}(\mathbf{x}^{(k)})\mathbf{F}(\mathbf{x}^{(k)})$

Come calcolare $\mathbf{z} = -J_F^{-1}(\mathbf{x}^{(k)})\mathbf{F}(\mathbf{x}^{(k)})$

Ad ogni iterazione k:
- valuto $J_F(\mathbf{x}^{(k)})$ e la memorizzo in una matrice A
- valuto $-\mathbf{F}(\mathbf{x}^{(k)})$ e lo memorizzo in un vettore $\mathbf{b}$
- calcolo $\mathbf{z} = A\backslash\mathbf{b}$

Attenzione: Il comando $\backslash$ non calcola in maniera esplicita A^{-1}, ma calcola $\mathbf{z}$ risolvendo il sistema lineare $A\mathbf{z} = \mathbf{b}$

Se A è non singolare $\quad A\mathbf{z} = \mathbf{b} \Leftrightarrow \mathbf{z} = A^{-1}\mathbf{b}$

Partendo dalla function `newton.m` scrivere NEWTON per SISTEMI

`newtonsys.m`

Input: `f, Jf, x0, tol, kmax`

`k=0, err=tol+1`
mentre `k` $<$ `kmax` e `err` $>$ `tol`
valuto $\mathbf{b} = -\mathbf{F}(\mathbf{x}^{(k)})$
valuto $A = J_F(\mathbf{x}^{(k)})$
risolvo $A\mathbf{z} = \mathbf{b}$
aggiorno la soluzione $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{z}$
calcolo l'errore `err`$=\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| = \|\mathbf{z}\|$
aggiorno `k`: `k=k+1`
aggiorno `x`

Output: `zero, f(zero), n_iterazioni, [err]`.

ESERCIZIO

(essisnl)
Calcolare numericamente le 4 radici del sistema non lineare

$$\begin{cases} x^2 - y^2 = 1 \\ x^2 + y^2 - 2x = 3 \end{cases} \Leftrightarrow \begin{cases} x_1^2 - x_2^2 = 1 \\ x_1^2 + x_2^2 - 2x_1 = 3 \end{cases} \quad (3)$$

con il metodo di Newton. Rappresentare graficamente su un solo grafico le storie di convergenza (ovvero gli errori $\|\mathbf{x}_{k+1} - \mathbf{x}_k\|$ al variare di $k = 0, ..., k_{max}$) e commentare i risultati ottenuti.
Scegliere questi tre dati inziali `x0=[1;1]`, `x0=[2;-1]`, `x0=[-2;-2]`.

Svolgimento

1. Rappresentazione grafica per la localizzazione delle radici
2. Risoluzione numerica

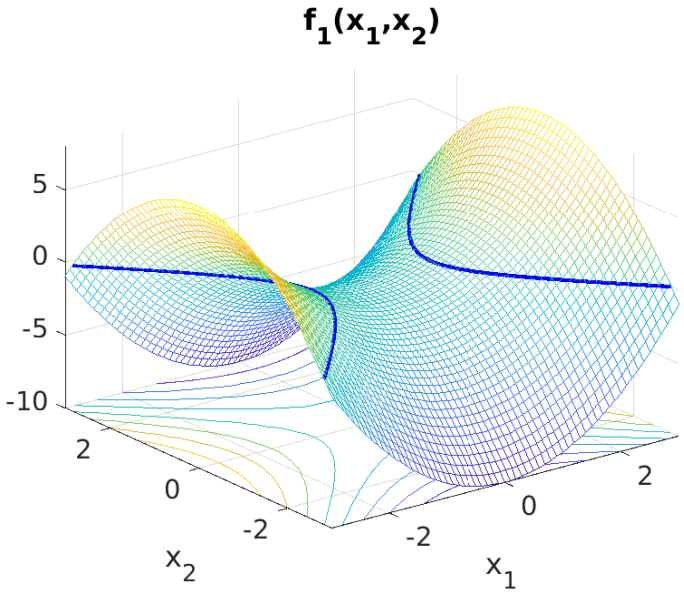
1. Rappresentazione grafica per la localizzazione delle radici

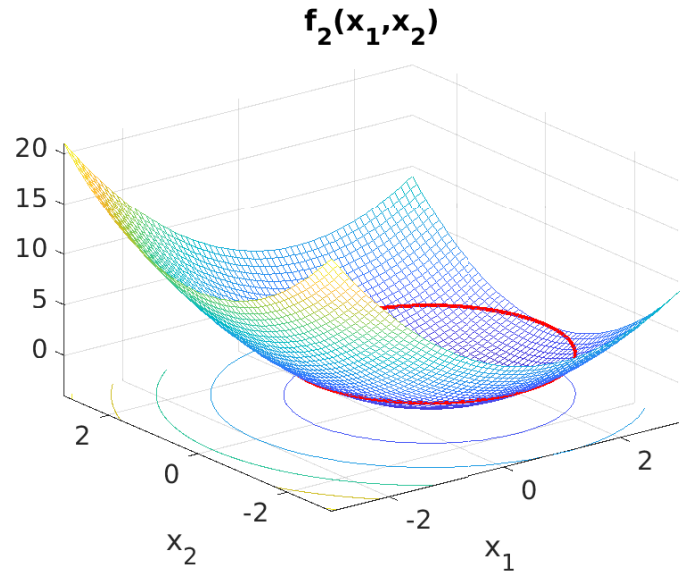
$\mathbf{x} = [x_1, x_2]^t$, $f_1(\mathbf{x}) = x_1^2 - x_2^2 - 1$, $f_2(\mathbf{x}) = x_1^2 + x_2^2 - 2x_1 - 3$.
Per localizzare le radici del sistema: disegno le superfici $z = f_1(\mathbf{x})$ e $z = f_2(\mathbf{x})$ e le contour $f_1(\mathbf{x}) = 0$ e $f_2(\mathbf{x}) = 0$.

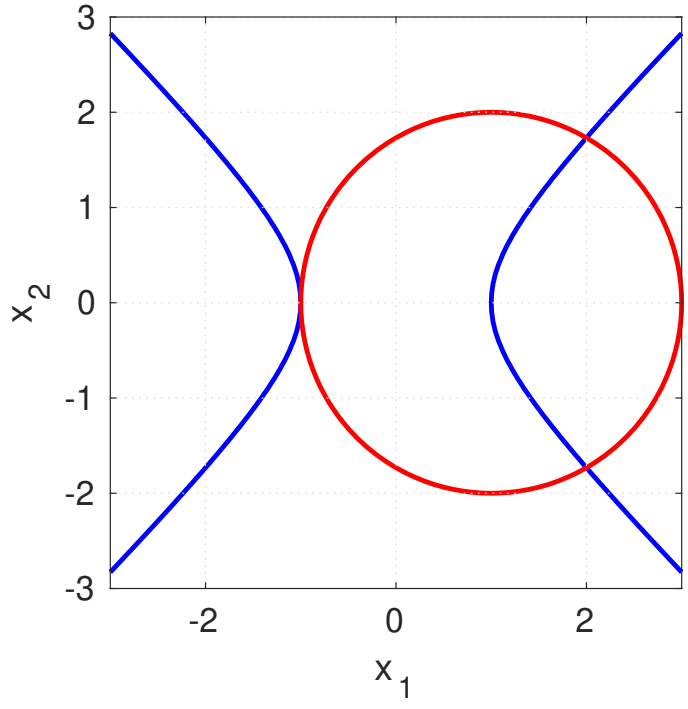
```
f1=@(x1,x2)x1.^2-x2.^2-1;
f2=@(x1,x2)x1.^2+x2.^2-2*x1-3;
[x1,x2]=meshgrid(-3:.1:3);
z_f1=f1(x1,x2);
z_f2=f2(x1,x2);

figure(1);clf
meshc(x1,x2,z_f1); title('z=f1(x_1,x_2)')
xlabel('x_1'); ylabel('x_2')
hold on
contour(x1,x2,z1,[0 0],'g','linewidth',2);
```

Disegnare f_2 su una seconda figura e le due contour (in $z = 0$) su una terza figura.


$f_1(x_1,x_2)$
5
0
-5
-10
2
0
-2
x_2
-2
0
2
x_1


$f_2(x_1,x_2)$
20
15
10
5
0
2
0
-2
x_2
-2
0
2
x_1


3
2
1
0
-1
-2
-3
x_2
-2
0
2
x_1

`f` e `Jf` devono essere function handle che dipendono dal vettore `x`.
Prima per disegnare le superifici ho utilizzato gli array `x1` e `x2`.
Ora per risolvere il sistema, `x(1)` e `x(2)` sono le componenti di un unico array `x`.

`f` è un function handle vettore colonna di 2 componenti

```
f=@(x)[f1(x(1),x(2));     % <-- f_1(x_1,x_2)
       f2(x(1),x(2))]   % <-- f_2(x_1,x_2)
```

`Jf` è un function handle matrice 2×2

```
Jf=@(x)[2*x(1),    -2*x(2);   % df_1/dx_1 , df_1/dx_2
        2*x(1)-2,  2*x(2)]    % df_2/dx_1 , df_2/dx_2
```

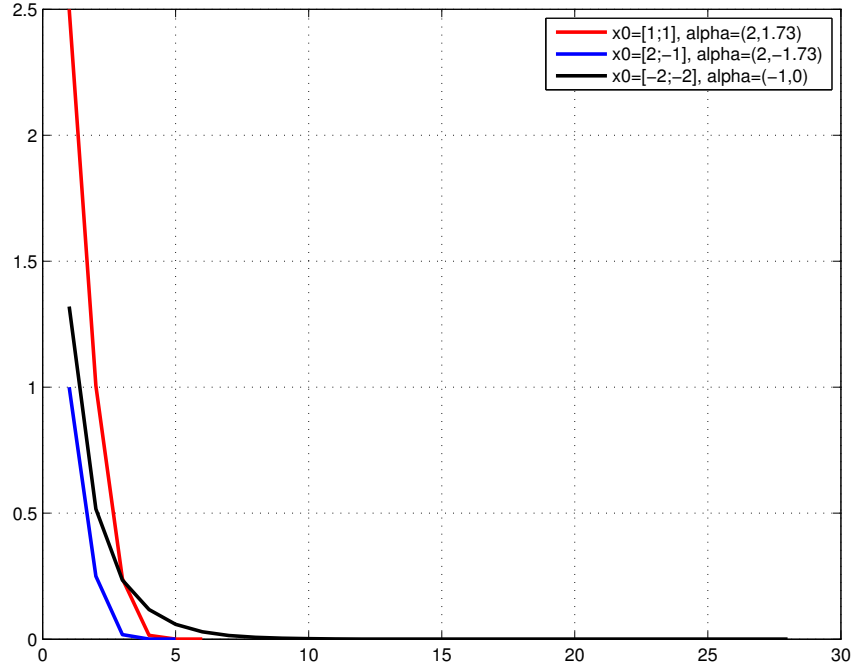
Definire gli altri input per `newtonsys.m` e chiamare `newtonsys.m`

Storie di convergenza.

Con i comandi

```
plot(Err1,'r'); hold on
plot(Err2,'b'); plot(Err3,'k')
legend('x0=[1;1], alpha=(2,1.73)',...
       'x0=[2;-1], alpha=(2,-1.73)',...
       'x0=[-2;-2], alpha=(-1,0)')
```

si ottengono le seguenti storie di convergenza:



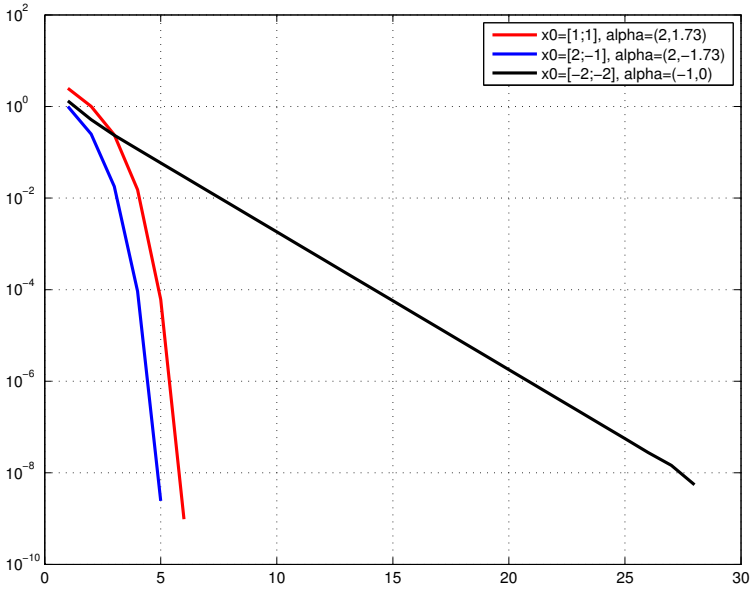
La rappresentazione non è buona.

Comando semilogy

Sostituendo il comando `plot` con `semilogy`

```
semilogy(Err1,'r'); hold on
semilogy(Err2,'b'); semilogy(Err3,'k')
legend('x0=[1;1], alpha=(2,1.73)',...
'x0=[2;-1], alpha=(2,-1.73)',...
'x0=[-2;-2], alpha=(-1,0)')
```

si ottengono le seguenti storie di convergenza:



La curva nera mostra una convergenza lineare. Le altre due curve convergenza quadratica. Infatti `alpha=(-1,0)` è radice multipla, le altre due sono semplici.

Il metodo di Broyden

E‘ una possibile generalizzazione del metodo delle secanti per risolvere sistemi di equazioni non lineari.
Non richiede il calcolo della matrice Jacobiana, ma costruisce ad ogni k una matrice B_k che approssima in maniera opportuna $J_F(\mathbf{x}^{(k)})$.
Richiede una matrice iniziale B_0, una scelta comune consiste nel prendere $B_0 = I$ (matrice identità).
Scaricare `broyden.m` dal direttorio delle function matlab `paola-gervasio.unibs.it/CS/matlab`

La sintassi di chiamata è:
`[zero,res,niter]=broyden(fun,B0,x0,tol,nmax,pflag)`
Risolvere il problema precedente con il metodo di Broyden.

Altri esempi di sistemi non lineari

$$\begin{cases} 4x_1^2 + x_2^2 = 4 \\ x_1 + x_2 = sin(x_1 - x_2). \end{cases}$$

$$\begin{cases} -\frac{1}{81}\cos(x_1) + \frac{1}{9}x_2^2 + \frac{1}{3}\sin(x_3) = x_1 \\ \frac{1}{3}\sin(x_1) + \frac{1}{3}\cos(x_3) = x_2 \\ -\frac{1}{9}\cos(x_1) + \frac{1}{3}x_2 + \frac{1}{6}\sin(x_3) = x_3. \end{cases}$$

$$\begin{cases} x_1^2 + x_2^2 - 2x_1 - 2x_2 + 1 = 0 \\ x_1 + x_2 - 2x_1x_2 = 0 \end{cases}$$