RISOLUZIONE NUMERICA DI SISTEMI NON LINEARI

Esempio:

si vuole risolvere numericamente

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

$\underline{F}(\underline{x}) = \underline{0}$

$$\begin{bmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Definisco:

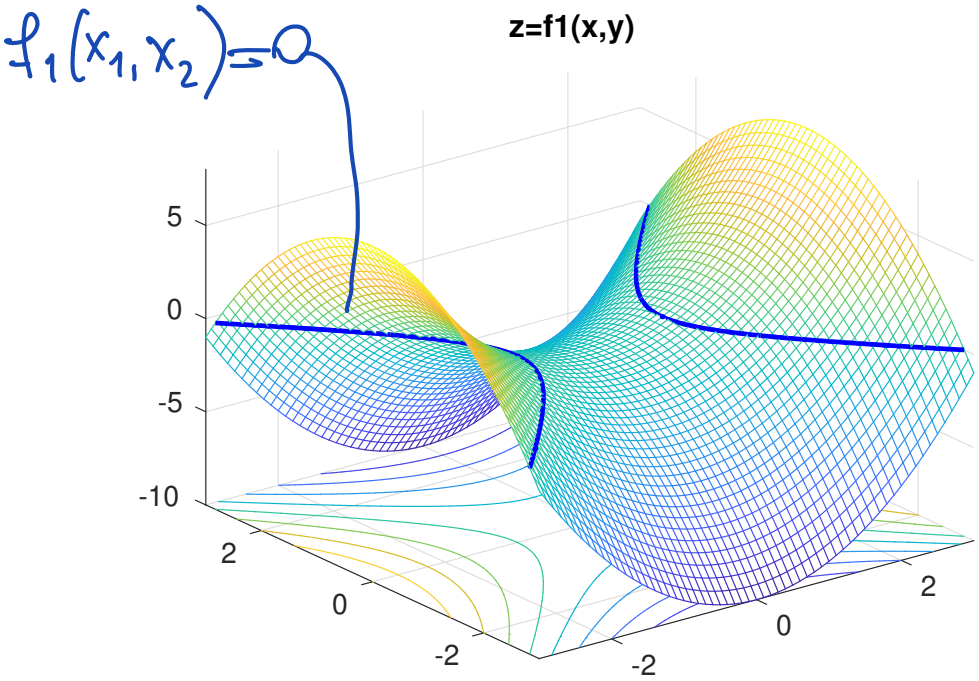
$$f_1(x_1, x_2) = x_1^2 - x_2^2 - 1$$
$$f_2(x_1, x_2) = x_1^2 + x_2^2 - 2x_1 - 3$$

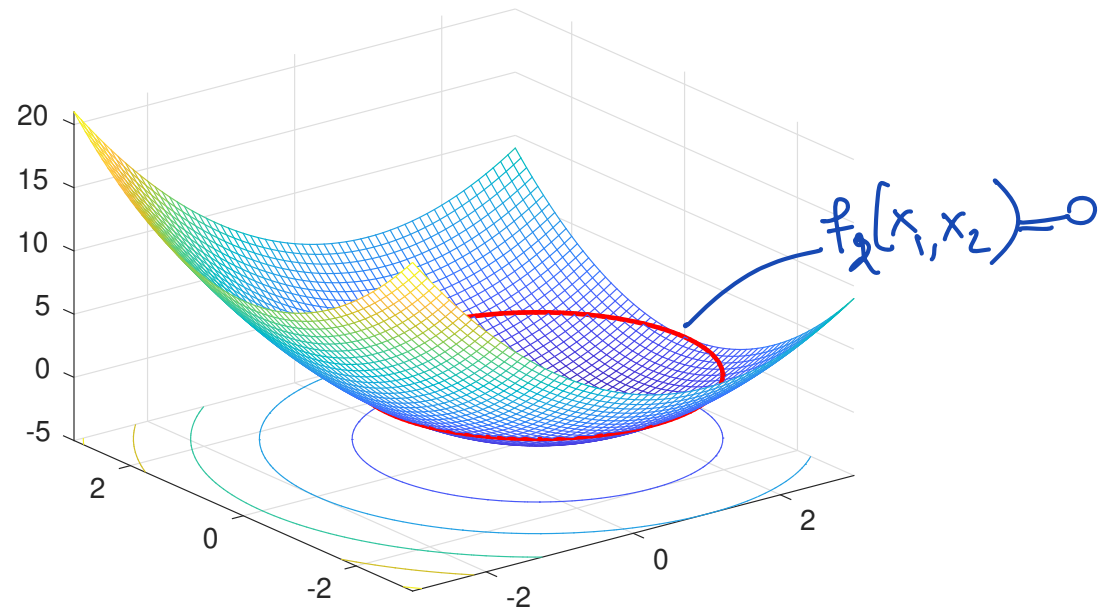
e

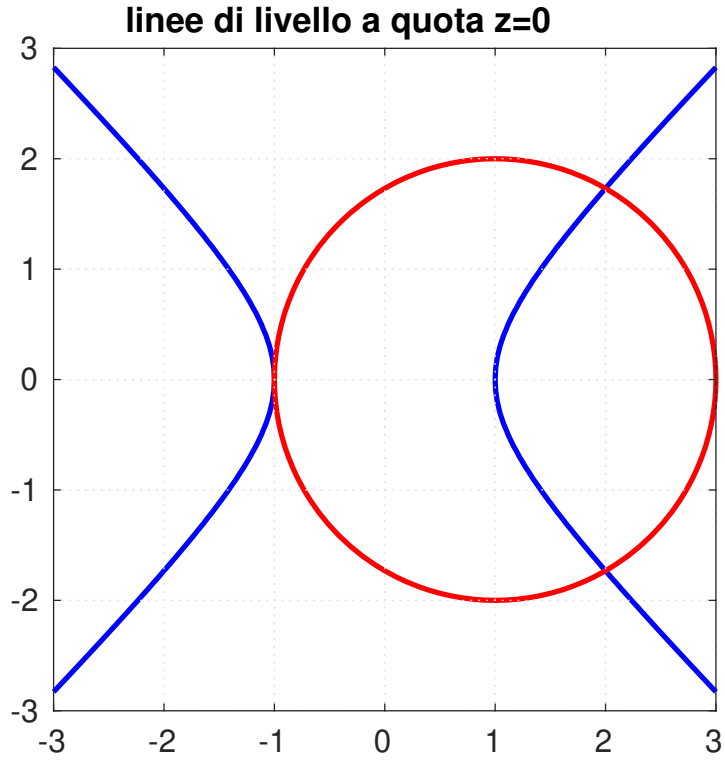
$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{bmatrix}$$

Risolvere (1) vuol dire

$$\text{trovare } \boldsymbol{\alpha} = [\alpha_1, \alpha_2]^T \in \mathbb{R}^2 : \text{tale che } \mathbf{F}(\boldsymbol{\alpha}) = \mathbf{0}.$$


z=f1(x,y)
5
0
-5
-10
2
0
-2
2
0
-2


z=f2(x,y)
20
15
10
5
0
-5
2
0
-2
2
0
-2


linee di livello a quota z=0
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

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:

$$\frac{1}{f'(x^{(k)})} = \left(f'(x^{(k)})\right)^{-1}$$

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

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 $J_F^{-1}(\mathbf{x}^{(k)})$.

$$\underline{F} : \underline{x} \to \underline{F}(\underline{x})$$
$$\mathbb{R}^n \to \mathbb{R}^n$$

matrice Jacobiana (Jacobiano)

$$J_F(\underline{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\underline{x}) & \frac{\partial f_1}{\partial x_2}(\underline{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\underline{x}) \\ \frac{\partial f_2}{\partial x_1}(\underline{x}) & \frac{\partial f_2}{\partial x_2}(\underline{x}) & \cdots & \frac{\partial f_2}{\partial x_n}(\underline{x}) \\ \vdots & & & \\ \frac{\partial f_n}{\partial x_1}(\underline{x}) & \frac{\partial f_n}{\partial x_2}(\underline{x}) & \cdots & \frac{\partial f_n}{\partial x_n}(\underline{x}) \end{bmatrix}$$

è una matrice di funzioni

ES
$$\underline{F}(\underline{x}) = \begin{bmatrix} f_1(x_1, x_2) = x_1^2 - x_2^2 - 1 \\ f_2(x_1, x_2) = x_1^2 + x_2^2 - 2x_1 - 3 \end{bmatrix}$$

$$J_F(\underline{x}) = \begin{bmatrix} 2x_1 & -2x_2 \\ 2x_1 - 2 & 2x_2 \end{bmatrix}$$

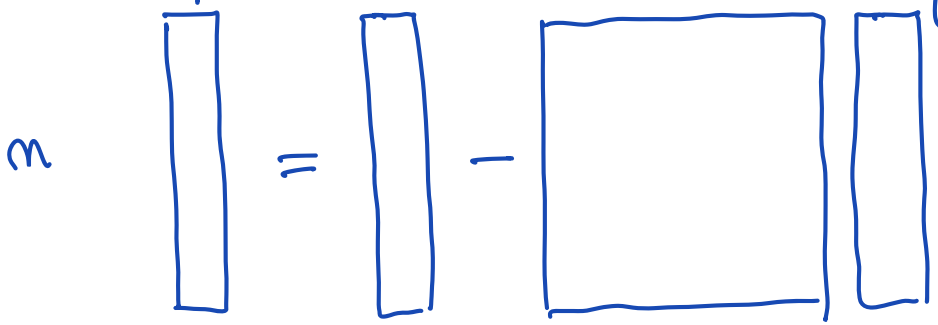
Allora il metodo di Newton per equazioni scalari

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

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



$$x = A^{-1} b$$

$$Ax = \underbrace{A \cdot A^{-1}}_{I} b$$

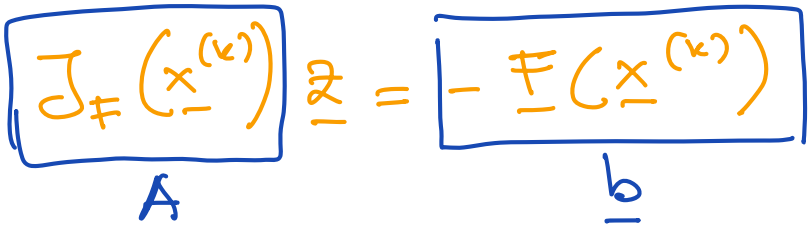
ricondurre il calcolo $x = A^{-1}b$ alla risoluzione di un sistema lineare $Ax = b$ ← t. noto

matrice del sistema ↑ ↖ incognita

per risolvere $Ax = b$ in matlab: $x = A \backslash b$

↑ back slash

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 $\quad A\mathbf{z} = \mathbf{b} \Leftrightarrow \mathbf{z} = A^{-1}\mathbf{b}$

Algoritmo Newton per sistemi

Input: $\underline{F}$, J_F, $\underline{x}^{(0)}$ (↑ vettore), tol, kmax

Output: sol (↑ vett soluzione), k, res (↑ vettore dei residui)

```
k = 0
err = tol + 1
while k < kmax & err ≥ tol
    in A salvo la valutazione di J_F nel punto x(0)
    in b salvo - la valut. di F(x(0))
    risolvo il sistema A z = b
    x_new = x(0) + z
    err = ||x_new - x(0)||_2   ← norma euclidea
    k = k+1
    x(0) = x_new
end
sol = x_new
res = F(sol)
```

($\underline{z} = -J_F^{-1}(\underline{x}^{(k)})\,\underline{F}(\underline{x}^{(k)})$, $\underline{x}^{(k+1)} = \underline{x}^{(k)} + \underline{z}$)

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].`