UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA CIVILE, EDILE E AMBIENTALE
Department Of Civil, Environmental and Architectural Engineering

Corso di Laurea in Ingegneria Civile

TESI DI LAUREA

Sistemi di equazioni non lineari: introduzione alla loro risoluzione numerica mediante i metodi di punto fisso e Newton-Raphson

Relatore:**Ch.ma Prof.ssa Annamaria Mazzia**

Laureanda:**Lucia Burato**

ANNO ACCADEMICO 2022-2023

Indice

Capitolo 1

Introduzione

1.1 Importanza del calcolo numerico

La ragione non ha luogo contro la forza della passione
- Francesco Petrarca.
Nonostante la bellezza di questa citazione del noto poeta Francesco Petrarca, vedremo come possa essere confutata, dimostrando come i fenomeni reali possano essere rappresentati tramite modelli matematici, anche l'amore.
Infatti, Calcolo Numerico è la disciplina che si occupa di risolvere quei modelli matematici per i quali non è possibile trovare una soluzione analitica, attraverso metodi numerici che portano a soluzioni accurate e affidabili. Per risolvere questi problemi matematici Calcolo Numerico sviluppa degli algoritmi che devono poi essere implementati al calcolatore tramite linguaggi di programmazione (MATLAB) in modo da poter essere certi dei risultati ottenuti.
Molti fenomeni reali osservati in diversi ambiti (nell'ingegneria, ma anche in biologia, fisica ed economia) possono essere tradotti da modelli matematici.
Nell'ampio campo del Calcolo Numerico noi ci concentreremo solo su una sua parte, quella che riguarda la risoluzione di sistemi di equazioni non lineari. Questo argomento rappresenta un problema molto rilevante nel mondo delle scienze matematiche applicate, in quanto i sistemi di equazioni non lineari sono essenziali nella modellazione di fenomeni complessi, poichè molte situazioni del mondo reale richiedono la risoluzione di tali sistemi per comprenderle e prevederle.

1.2 Metodi iterativi per la risoluzione di modelli matematici

I metodi iterativi sono algoritmi affidabili e ampiamente utilizzati per risolvere problemi matematici quando non è possibile trovare soluzioni esatte o quando le soluzioni esatte sono troppo complesse da calcolare. Questi metodi forniscono soluzioni

approssimate, ma precise a sufficienza per molte applicazioni. Le iterazioni successive si basano sulle precedenti, richiedendo una stima iniziale. Il metodo continua finché l'approssimazione non soddisfa una certa tolleranza prefissata. Questi metodi sono particolarmente utili per risolvere equazioni e sistemi non lineari, per i quali mancano soluzioni analitiche. Tra i principali metodi iterativi troviamo il metodo del punto fisso e il metodo di Newton-Raphson, che differiscono nella loro costruzione e velocità di convergenza.

1.3 Applicazioni reali di sistemi non lineari

Abbiamo vari esempi di applicazioni reali schematizzabili tramite sistemi non lineari risolvibili con metodi iterativi e non, ad esempio:

- GPS, noto a tutti a livello generale ma forse non nel suo funzionamento specifico. Il GPS, abbreviazione di Sistema di Posizionamento Globale, determina la posizione di un dispositivo basandosi sulle distanze dai satelliti. I satelliti trasmettono segnali verso la Terra, e un ricevitore GPS misura quanto tempo impiega un segnale per raggiungerlo. Questo intervallo di tempo è convertito in una distanza, considerando che il segnale si propaga alla velocità della luce. Con almeno tre satelliti, il ricevitore può calcolare la sua posizione intersecando le sfere descritte dalle distanze misurate dai satelliti. In questo modo, si determina con precisione la posizione del ricevitore. Quindi, il problema di determinare la posizione del ricevitore nel contesto del GPS può essere visto come un problema numerico, in particolare come un problema di approssimazione ai minimi quadrati non lineari.
  In altre parole, si tratta di trovare la posizione del ricevitore che minimizza la somma dei quadrati degli errori tra le distanze misurate dai satelliti e le distanze calcolate sulla base della posizione stimata. Questo è un esempio di problema di ottimizzazione numerica in cui si cerca la soluzione migliore tra un insieme di possibili posizioni del ricevitore. L'applicazione dei metodi di punto fisso e Newton-Raphson è uno dei modi per affrontare questa approssimazione ai minimi quadrati non lineari. Questi metodi consentono di iterare per migliorare la stima della posizione del ricevitore fino a raggiungere una soluzione approssimata che minimizzi l'errore quadratico.

- Analisi di circuiti elettrici complessi, in cui sono presenti bipoli non lineari, quindi descritti da una caratteristica esterna rappresentabile mediante un'equazione non lineare. In questi bipoli le variabili di corrente e tensione non so-

no legate tra loro da relazioni lineari. Ci troviamo quindi con equazioni non lineari che descrivono il flusso di corrente e tensione attraverso i vari componenti/bipoli e i metodi iterativi di Newton o di punto fisso ci aiutano a trovare il punto di funzionamento del circuito, ovvero i valori incogniti di tensioni e correnti di ogni componente in modo che il circuito sia in regime stabile.

Abbiamo altri esempi in svariati campi di applicazione, come ad esempio nella dinamica dei fluidi, ma un caso molto particolare può essere osservato anche in campo letterario, come già visto nelle prime righe della nostra trattazione, a dismostrare che anche fenomeni apparentemente imprevedibili, come l'amore, possono essere schematizzati da metodi numerici.

1.4 Laura e Petrarca: un amore razionale

Entrando più nel dettaglio, infatti, F.J. Jones, scienziato dell'opera letteraria, riconobbe un comportamento oscillante tra gli stati di amore e disperazione presenti nei sonetti di Petrarca dedicati alla sua amata Laura; questo comportamento venne chiamato "ciclo emozionale di Petrarca". Successivamente, il matematico S. Rinaldi schematizzò questo ciclo tramite un modello matematico basato su due ODE non lineari accoppiate, che riflettono le emozioni di Laura e Petrarca l'uno per l'altra.
Nei suoi studi Jones ha analizzato i 23 sonetti di Petrarca dedicati a Laura con una datazione sicura, in modo da essere certi del loro ordine cronologico. Ha poi assegnato dei voti alle poesie, da -1 a +1, determinando il ciclo emotivo di Petrarca: il voto massimo (+1) indica l'amore estatico, mentre i voti molto negativi corrispondono a una profonda disperazione. Rinaldi ha cercato di modellizzare il ciclo emozionale di Petrarca, accertato sperimentalmente. Ha sviluppato un modello matematico che si fonda su una dinamica del tipo predatore-preda tra le variabili L(t) e P(t).
L(t) rappresenta l'amore di Laura per il poeta al tempo t, mentre P(t) l'amore di Petrarca per l'amata. Valori positivi e elevati di L assumono il significato di affetto profondo, mentre valori negativi sono interpretati come antagonismo. Invece, elevati valori di P significano amore trascendente e valori negativi disperazione. In sintesi, il modello generale Laura-Petrarca è dato dalle seguenti equazioni differenziali:

$$\frac{d}{dt}L(t) = -\alpha_L L(t) + R_L P(t) + \beta_L A_P$$

$$\frac{d}{dt}P(t) = -\alpha_P P(t) + R_P L(t) + \beta_P \frac{A_L}{1 + \delta_P I_P(t)}$$

$$\frac{d}{dt}I_P(t) = -\alpha_{\text{IP}} P(t) + \beta_{\text{IP}} P(t)$$

dove:

- $I_P(t)$ è una variabile utilizzata per descrivere l'ispirazione poetica di Petrarca.
- A_P e A_L rappresentano l'attrazione fisica, ma anche mentale che provano Laura e Petrarca l'uno nei confronti dell'altra.
- $R_L(P)$ è la funzione di reazione di Laura nei confronti dell'amore di Petrarca, mentre $R_P(L)$ è la funzione di reazione di Petrarca all'amore di Laura.

La variazione rispetto al tempo dell'amore di Laura è somma di tre termini: il primo descrive il processo di dimenticanza che caratterizza ogni individuo, il secondo è la reazione di Laura all'amore di Petrarca e il terzo è la risposta di lei all'appello del poeta. La variazione rispetto al tempo dell'amore di Petrarca verso Laura ha una struttura simile alla precedente ma con un'estensione, infatti la risposta di Petrarca all'appello di Laura dipende anche dalla sua ispirazione poetica.
Infine, l'equazione per l'ispirazione afferma che l'amore di Petrarca sostenga la sua ispirazione che, altrimenti, decadrebbe esponenzialmente.
Possiamo affermare quasi con certezza che $R_P(L)$ sia lineare, dal momento che le reazioni di Petrarca sono evidenti nelle sue poesie: quando Laura mostra segni di apprezzamento, lui reagisce con amore, mentre a segni di antagonismo da parte dell'amata risponde con disperazione.
Non possiamo invece considerare lineare la reazione di Laura all'amore manifestato dal poeta. Infatti, il comportamento di Laura supponiamo essere il seguente: quando il poeta non dimostra eccessivo interesse, ella risponde con neutralità, quando il poeta invece dimostra il proprio amore, tende a respingerlo, ma quando invece Petrarca mostra sconforto, Laura sembra provare dispiacere per lui. Quindi, traducendo questo comportamento in termini matematici, per $P > 0$, $R_L(P)$ ha un breve incremento, ma subito decresce; per $P < 0$, $R_L(P)$ cresce rapidamente. In conclusione, la funzione $R_L(P)$ è una cubica.
Il modello differenziale è stato poi implementato in Matlab ed è stato utilizzato un metodo ai minimi quadrati in cui viene posta nulla la somma degli scarti al quadrato tra P(t) e il ciclo emozionale di Petrarca E(t). I risultati della soluzione numerica sono qualitativamente in pieno accordo con l'analisi di Frederic Jones: dopo un primo picco elevato, l'amore di Petrarca P(t) tende a un ciclo regolare caratterizzato da picchi positivi e negativi alternati. Anche l'amore di Laura L(t) e l'ispirazione poetica di Petrarca IP(t) tendono a un andamento ciclico. All'inizio, l'ispirazione di Petrarca IP(t) sale molto più lentamente del suo amore e poi rimane positiva per tutto il periodo. Al contrario, l'amore di Laura è sempre negativo. La conclusione è che l'adattamento tra P(t) e E(t) è soddisfacente e che il modello Laura-Petrarca di

Rinaldi supporta fortemente la congettura di Frederic Jones sul ciclo emozionale di Petrarca.

Nell'ultimo capitolo ci concentreremo in particolare su due applicazioni reali: il funzionamento di un braccio robotico e il sistema LORAN (Long Range Navigation).

Capitolo 2

Metodi iterativi semplici

2.1 Metodo di punto fisso semplice

Il problema da indagare è trovare un metodo che ci consenta di trovare la radice, ovvero la soluzione di una funzione non lineare, cioè f(x)=0. Questo problema può essere reso equivalente alla ricerca del punto fisso di una opportuna funzione g, cioè risolvendo $g(x) = x$. Un esempio di funzione $g(x)$ è data dal prendere $g(x) = f(x) + x$.

Definizione 1. *Data una funzione g, si definisce punto fisso della g, quel punto ξ che soddisfa la relazione $g(\xi) = \xi$*

Se l'iterazione $x_{\text{n+1}} = g(x_n)$ converge a ξ, allora ξ è punto fisso per la funzione g.
Non sempre però, questo schema iterativo, applicato a funzioni che ammettono uno o più punti fissi, converge; infatti, ciascun punto fisso ξ ha un proprio bacino di attrazione e se prendo x_0 in questo bacino allora i valori delle iterazioni successive x_n tenderanno a ξ, altrimenti il metodo non convergerà. Bisogna inoltre sapere che una funzione può ammettere più di un punto fisso, ammetterne solo uno o non ammetterne proprio e il Teorema di convergenza per lo schema iterativo del punto fisso ci dice quando una funzione può ammettere punti fissi.

Teorema 1. *A partire da un punto iniziale x_0, lo schema iterativo $x_{n+1} = g(x_n)$ converge al punto fisso ξ di g se $|g'(x)| < 1$ in un intorno di ξ.*

Quando il metodo converge, come capiamo quando fermarci?
Possiamo considerare l'iterazione n-esima una buona approssimazione della soluzione quando lo scarto all'iterazione n, ovvero $d_n = |x_n - x_{\text{n-1}}|$ risulta minore di una certa tolleranza tol prestabilita.
Possiamo implementare l'algoritmo dello schema di punto fisso in MATLAB; vediamo un esempio qui di seguito.

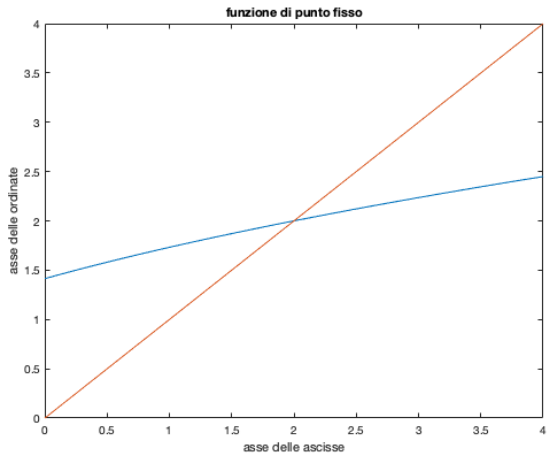
Esempio 1. *Abbiamo una funzione $g(x) = \sqrt{x+2}$ per la quale vogliamo determinare il punto fisso ξ attraverso lo schema iterativo di punto fisso.*
La soluzione esatta può essere determinata anche analiticamente e risulta essere $\xi = 2$. Vogliamo ora creare uno script in Matlab che ci dia come risultato la soluzione approssimata attraverso il metodo di punto fisso. Inoltre, sappiamo che il metodo converge poichè $|g'(x) < 1|$ per qualunque valore di x.

```
% voglio trovare il punto fisso della funzione g(x)=sqrt(x+2)
% lo schema di punto fisso lo scrivo come xn+1=g(xn)
g=@(x) sqrt(x+2); % definisco la funzione
% facciamo il grafico della funzione e della bisettrice del primo
% quadrante per visualizzare graficamente il punto fisso
a=0; b=4; % individuo un intervallo
fplot(g,[a,b])
hold on
plot([a,b],[a,b])
xlabel('asse delle ascisse')
ylabel('asse delle ordinate')
title('funzione di punto fisso')
hold off
% nel grafico vedo la bisettrice in rosso e la funzione g in blu
% vediamo che il punto fisso è x=2, dove si intersecano g(x) e y=x
% assegno il valore di tolleranza, iterazioni massime e x0
x0=1; toll=1e-6; itmax=100;
vettscarti=zeros(itmax,1); % preallocazione vettore scarti
scarto=2*toll; % valore per entrare nel ciclo while
iter=0;
xold=x0;
while scarto>=toll && iter<=itmax
    iter=iter+1; % incremento le iterazioni
    xnew=g(xold);
    scarto=abs(xnew-xold); % scarto all'iterazione iter
    vettscarti(iter)=scarto;
    xold=xnew; % aggiorno la mia variabile
end
vettscarti=vettscarti(1:iter); % taglio il vettore scarti
M=vettscarti(2:iter)./vettscarti(1:iter-1); % rapporto tra scarti
% stampo la soluzione approssimata
fprintf(1,'valore approssimato del punto fisso %12.8e \n',xnew);
fprintf(1,'iterazioni eseguite %i \n',iter);
fprintf(1,'%12.8e \n',M);
figure
vettiter=1:iter; % creo il vettore delle iterazioni
semilogy(vettiter,vettscarti);
title('grafico di convergenza')
```

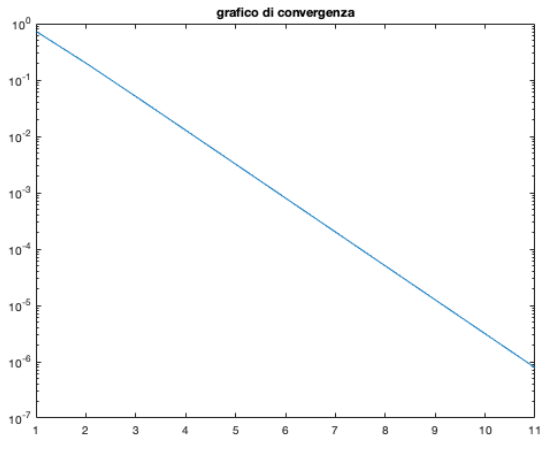
Eseguendo lo script in Matlab otterremo i seguenti risultati:

```
puntofissosemplice
valore approssimato del punto fisso 1.99999974e+00
iterazioni eseguite 11
2.72933030e-01
2.55444716e-01
2.51344216e-01
2.50335009e-01
2.50083687e-01
2.50020918e-01
2.50005229e-01
2.50001307e-01
2.50000327e-01
2.50000082e-01
```

Possiamo quindi vedere come il metodo converga alla soluzione esatta, visibile anche nel grafico (a) sottostante.
Inoltre, il rapporto tra gli scarti M (fattore di convergenza) mostra come la convergenza del metodo di punto fisso sia lineare (possiamo notarlo anche dal grafico di convergenza (b), che è una retta).



(a) funzione g(x) di punto fisso



(b) Convergenza lineare metodo punto fisso

2.2 Metodo di Newton-Raphson semplice

Il metodo in questione è il più valido, nonchè il più utilizzato per risolvere equazioni non lineari. L'obiettivo è trovare la soluzione/radice di una funzione f(x), ovvero individuare il punto in cui la funzione interseca l'asse delle ascisse. Come lo schema di punto fisso, è un metodo iterativo in cui le successive iterazioni dovrebbero (se lo schema converge) portare alla soluzione approssimata del nostro problema.

Per capire in che modo converge possiamo studiarlo da un punto di vista **geometrico**: partiamo da un'approssimazione iniziale x_0 e tracciamo la retta tangente alla funzione $f(x)$ in $(x_0, f(x_0))$. Questa retta interseca l'asse delle ascisse nel punto x_1, quindi poi tracciamo la tangente alla funzione f in $(x_1, f(x_1))$ e chiamiamo x_2 il punto in cui la retta tangente interseca l'asse delle ascisse. Se proseguiamo con questo procedimento, partendo da un'approssimazione iniziale nell'intorno della soluzione esatta, arriviamo in poche iterazioni ad una valida approssimazione della nostra radice esatta.
Partendo da questa costruzione geometrica e supponendo che le derivate prima e seconda di f esistano e siano continue e assumendo che la derivata prima di f non sia identicamente nulla, posso facilmente arrivare alla formulazione esplicativa del metodo di Newton-Raphson, ovvero:

$$x_{\text{n+1}} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Man mano che le approssimazioni si avvicinano alla soluzione esatta ξ, l'errore (definito come la differenza tra le approssimazioni successive e la soluzione esatta) e lo scarto (definito come la differenza tra due approssimazioni successive) si riducono e il mio metodo iterativo si arresta quando questi due valori sono inferiori a una certa tolleranza tol prefissata.
Tendenzialmente l'ordine di convergenza dello schema di Newton è quadratico. Qui di seguito implementeremo questo metodo in Matlab tramite un esempio.

Esempio 2. *Vogliamo trovare, implementando in Matlab lo schema di Newton, la soluzione approssimata della funzione* $f(x) = \sqrt{x+2} - x$. *Sappiamo che la soluzione è* $x = 2$ *e quindi partiamo da un'approssimazione iniziale nel suo intorno, ovvero* $x_0 = 1$.
Dall'esempio precedente, $g(x) = \sqrt{x+2}$ *e sapendo che* $g(x) = f(x) + x$ *e avendo trovato prima il punto fisso della* $g(x)$*, si vuole dimostrare che la soluzione della* $f(x)$ *(uguale quindi al punto fisso della* $g(x)$*) può essere determinata con molte meno iterazioni rispetto all'esempio precedente, in quanto il metodo di Newton converge quadraticamente a differenza dello schema di punto fisso, che converge linearmente.*

```
% Voglio trovare la soluzione della funzione f(x)=sqrt(x+2)-x
f = @(x) sqrt(x + 2) - x; % definisco la funzione
df = @(x) 1 / (2 * sqrt(x + 2)) - 1; % definisco la derivata prima
% faccio il grafico della funzione per individuare la soluzione
a=0; b=5; % individuo un intervallo
fplot(f,[a,b])
hold on
```

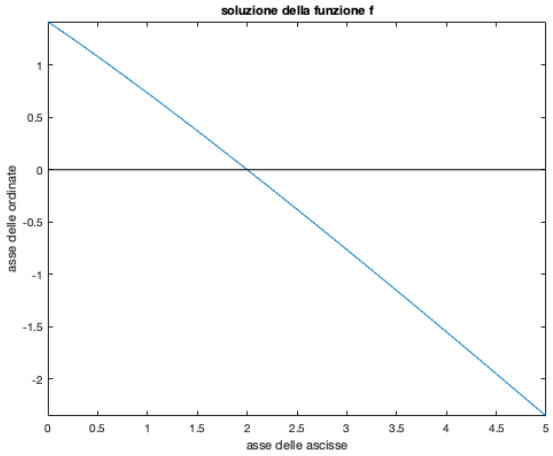
```
% Disegno l'asse delle ascisse
x = [a,b];
y = zeros(size(x));
% Disegno l'asse delle ascisse
plot(x, y, 'k-'); % 'k-' specifica una linea nera
xlabel('asse delle ascisse')
ylabel('asse delle ordinate')
title('soluzione della funzione f')
hold off
% vediamo che la soluzione è x=2
% soluzione=intersezione funzione con asse ascisse
x0=1; toll=1e-6; itmax=100;
vettscarti=zeros(itmax,1); % preallocazione vettore scarti
scarto=2*toll; % valore per entrare nel ciclo while
iter=0;
xold=x0;
% Implementazione del metodo di Newton attraverso ciclo while
% metodo di Newton: x_n+1=x_n-[f(x_n)/df(x_n)]
while scarto>=toll && iter<=itmax
    iter=iter+1; % incremento le iterazioni
    xnew=xold-f(xold)/df(xold);
    scarto=abs(xnew-xold); % scarto all'iterazione iter
    vettscarti(iter)=scarto;
    xold=xnew; % aggiorno la mia variabile
end
vettscarti=vettscarti(1:iter); % taglio il vettore scarti
% stampo la soluzione approssimata
fprintf(1,'valore approssimato della soluzione %12.8e \n',xnew);
fprintf(1,'iterazioni eseguite %i \n',iter);
% Visualizziamo il grafico di convergenza
figure
vettiter=1:iter; % creo il vettore delle iterazioni
semilogy(vettiter,vettscarti);
title('grafico di convergenza')
```

Eseguendo lo script in Matlab otteniamo i seguenti risultati:

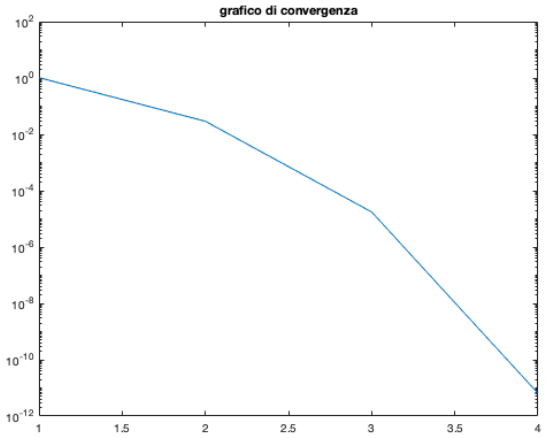
```
>> newtonsemplice2
valore approssimato della soluzione 2.00000000e+00
iterazioni eseguite 4
```

Possiamo quindi vedere come il metodo converga alla soluzione esatta, visibile anche nel grafico (a) sottostante.
Inoltre, il grafico (b) di convergenza (ramo di parabola) mostra come la convergenza dello schema sia quadratica e non lineare.
Rispetto alla funzione g in cui servivano ben 11 iterazioni per determinare il punto fisso, ora con questo esempio speculare, vediamo che il metodo di Newton-Raphson

arriva a soluzione ben più velocemente, qui addirittura abbiamo ottenuto la radice esatta in sole 4 iterazioni.



(a) radice della funzione f(x)



(b) Convergenza quadratica Newton-Raphson

2.3 Convergenza a confronto: Newton-Raphson e punto fisso

Sappiamo che il metodo di Newton converge quadraticamente, mentre quello di punto fisso linearmente, ma cosa significa realmente questa affermazione?
Quando il metodo converge alla soluzione, noi vogliamo studiare in che modo si riduce l'errore ad ogni iterazione (ordine di convergenza p) e secondo quale fattore (fattore di convergenza M).

Definizione 2. *Un metodo ha ordine di convergenza p se si possono definire due costanti $p \geq 1$ e $M > 0$ tali che:*

$$\lim_{n\to\infty} \frac{|x_{n+1} - \xi|}{|x_n - \xi|^p} = M$$

dove:

- *p=ordine di convergenza*
- *M=fattore di convergenza*
- *ξ=soluzione esatta*
- *x_n=iterazione corrente*
- *x_{n+1}=iterazione successiva*

Un metodo iterativo si dice:

- **A convergenza lineare** se vale: $|x_{\text{n+1}} - \xi| \leq M|x_n - \xi|$
- **A convergenza quadratica** se vale: $|x_{\text{n+1}} - \xi| \leq M|x_n - \xi|^2$

Tralasciando i procedimenti necessari per arrivare a questa conclusione, possiamo affermare:

- Nel caso di Newton-Raphson $p = 2$ e $M = \left|\frac{f''(\xi)}{2f'(\xi)}\right|$
- Nel caso di punto fisso $p = 1$ e $M = |g'(\xi)|$

Capitolo 3

Risoluzione di sistemi non lineari tramite metodi iterativi

3.1 Trattazione teorica

Il modo più semplice di risolvere sistemi di equazioni non lineari è adattare i metodi iterativi visti in precedenza (Newton-Raphson e punto fisso). Ora, invece di approssimare la soluzione di una singola equazione non lineare in una variabile, lavoriamo con più funzioni in più variabili.
Un sistema di equazioni non lineari ha la seguente forma:

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0 \\ f_2(x_1, x_2, \ldots, x_n) = 0 \\ f_3(x_1, x_2, \ldots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n) = 0 \end{cases}$$

dove ogni funzione $f_i(x_1, x_2, \ldots, x_n)$, che può anche essere vista come $f(\boldsymbol{x})$, prende un vettore $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T$ nello spazio n-dimensionale $\mathbb{R}^n$ e lo associa a un valore sulla retta reale $\mathbb{R}$.
Questo sistema di n equazioni non lineari in n incognite può anche essere scritto nel seguente modo, identificando una funzione $\boldsymbol{F}$ definita da $\mathbb{R}^n$ in $\mathbb{R}^n$:

$$\boldsymbol{F}(x_1, x_2, \ldots, x_n) = (f_1(x_1, x_2, \ldots, x_n), f_2(x_1, x_2, \ldots, x_n), \ldots, f_n(x_1, x_2, \ldots, x_n))^T$$

In questo modo, se $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, il sistema (4.1) assume la forma:

$$\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0} \tag{4.2}$$

Prima di passare alla risoluzione del sistema di equazioni non lineari è necessario trattare la continuità e la differenzabilità di funzioni definite da $\mathbb{R}^n$ in $\mathbb{R}^n$.

3.1.1 Definizioni e Teoremi premilinari

Definizione 3. *Sia* $f : D \subset \mathbb{R}^n \to \mathbb{R}$. *La funzione* f *tende al limite L per* $\boldsymbol{x}$ *che tende a* $\boldsymbol{x_0}$*, scritta:*

$$\lim_{\boldsymbol{x}\to\boldsymbol{x_0}} f(\boldsymbol{x}) = L$$

se, $\forall\epsilon > 0, \exists\delta$ *tale che:*

$$|f(\boldsymbol{x}) - L| < \epsilon$$

ogni qualvolta che $\boldsymbol{x} \in D$ *e* $0 < ||\boldsymbol{x} - \boldsymbol{x_0}|| < \delta$

La conoscenza della definizione di limite di una funzione ci permette di spiegare la continuità per funzioni definite da $\mathbb{R}^n$ in $\mathbb{R}$.

Definizione 4. *Sia* $f : D \subset \mathbb{R}^n \to \mathbb{R}$. *La funzione* f *è* ***continua*** *in* $\boldsymbol{x_0} \in D$ *purchè il limite* $\lim_{\boldsymbol{x}\to\boldsymbol{x_0}} f(\boldsymbol{x})$ *esista e*

$$\lim_{\boldsymbol{x}\to\boldsymbol{x_0}} f(\boldsymbol{x}) = f(\boldsymbol{x_0})$$

Inoltre, f *è continua in* D *se* f *è continua in ogni punto di* D*, ovvero* $f \in C(D)$.

Possiamo ora definire i concetti di limite e continuità per funzioni definite da $\mathbb{R}^n$ in $\mathbb{R}^n$ considerando le funzioni coordinate $f_i : \mathbb{R}^n \to \mathbb{R}$.

Definizione 5. *Sia* $\boldsymbol{F} : D \subset \mathbb{R}^n \to \mathbb{R}^n$ *una funzione di forma:*

$$\boldsymbol{F}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x}))^t$$

dove $f_i : \mathbb{R}^n \to \mathbb{R}\ \forall i$*. Definiamo:*

$$\lim_{\boldsymbol{x}\to\boldsymbol{x_0}} \boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{L} = (L_1, L_2, \ldots, L_n)^t$$

se e solo se $\lim_{\boldsymbol{x}\to\boldsymbol{x_0}} f_i(\boldsymbol{x}) = L_i$ *per ogni* $i = 1, 2, \ldots, n$.

Avendo visto che $\lim_{\boldsymbol{x}\to\boldsymbol{x_0}} \boldsymbol{F}(\boldsymbol{x})$ esiste e che $\lim_{\boldsymbol{x}\to\boldsymbol{x_0}} \boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{F}(\boldsymbol{x_0})$, allora la funzione $\boldsymbol{F}$ è **continua** in $\boldsymbol{x_0} \in D$.

Per funzioni definite da $\mathbb{R}$ in $\mathbb{R}$ la continuità di una funzione può essere mostrata dimostrando la sua derivabilità. Sebbene questo teorema possa essere generalizzato per funzioni a più variabili, non sarà qui presentato. Tuttavia, il seguente teorema mette in relazione la continuità di una funzione in n variabili in un punto con le derivate parziali della medesima funzione nello stesso punto.

Teorema 2. *Sia $f : D \subset \mathbb{R}^n \to \mathbb{R}$ e $\boldsymbol{x_0} \in D$. Supponiamo che tutte le derivate parziali di f esistano e che esistano le costanti $\delta > 0$ e $K > 0$ tali che, quando $||\boldsymbol{x} - \boldsymbol{x_0}|| < \delta$ e $\boldsymbol{x} \in D$, abbiamo:*

$$\left|\frac{\partial f(\boldsymbol{x})}{\partial x_j}\right| \leq K \quad \textit{per ogni } j = 1, 2, \ldots, n. \tag{3.1}$$

Quindi f è continua in $\boldsymbol{x_0}$.

Dimostrazione. Per ipotesi $||\boldsymbol{x} - \boldsymbol{x_0}|| < \delta \implies \left|\frac{\partial f(\boldsymbol{x})}{\partial x_j}\right| \leq K$.
L'ultimo termine di destra può essere scritto anche come:

$$\left|\frac{\partial f(\boldsymbol{x})}{\partial x_j}\right| = \lim_{\boldsymbol{x}\to\boldsymbol{x_0}} \left|\frac{f(\boldsymbol{x}) - f(\boldsymbol{x_0})}{\boldsymbol{x} - \boldsymbol{x_0}}\right| < K$$

che significa che $\dfrac{\partial f}{\partial \boldsymbol{x}}$ è limitato. Ne consegue che:

$$\lim_{\boldsymbol{x}\to\boldsymbol{x_0}} |f(\boldsymbol{x}) - f(\boldsymbol{x_0})| = \lim_{\boldsymbol{x}\to\boldsymbol{x_0}} |\boldsymbol{x} - \boldsymbol{x_0}| \frac{|f(\boldsymbol{x}) - f(\boldsymbol{x_0})|}{|\boldsymbol{x} - \boldsymbol{x_0}|} = 0$$

La precedente espressione è nulla perchè il primo temine $|\boldsymbol{x} - \boldsymbol{x_0}|$ è nullo per $\lim_{\boldsymbol{x}\to\boldsymbol{x_0}}$ e il secondo termine è limitato.
In conclusione:

$$\lim_{\boldsymbol{x}\to\boldsymbol{x_0}} |f(\boldsymbol{x}) - f(\boldsymbol{x_0})| = 0 \longmapsto \lim_{\boldsymbol{x}\to\boldsymbol{x_0}} f(\boldsymbol{x}) = f(\boldsymbol{x_0})$$

Il limite esiste e questo significa che $f(\boldsymbol{x})$ è continua. □

Ora generalizziamo gli schemi iterativi visti in precedenza, quindi lo schema di punto fisso e lo schema di Newton-Raphson.

3.1.2 Metodo del punto fisso per funzioni in n variabili

Nel *capitolo 3* abbiamo già parlato del metodo di punto fisso per una funzione a una sola variabile, dicendo che l'equazione $f(x) = 0$, di cui si voleva trovare la radice, poteva essere trasformata nella forma di punto fisso $g(x) = x$, di cui si vuole determinare il punto fisso della g. Ora vogliamo generalizzare lo schema di punto fisso a funzioni in n variabili.

Definizione 6. *Una funzione $G : D \subset \mathbb{R}^n \mapsto \mathbb{R}^n$ ha un **punto fisso** in $\boldsymbol{p} \in D$ se $G(\boldsymbol{p}) = \boldsymbol{p}$.*

Il seguente Teorema estende il Teorema di punto fisso al caso n-dimensionale.

Teorema 3. *Sia* $D = \{(x_1, x_2, \ldots, x_n)^t | a_i \leq x_i \leq b_i, \forall i = 1, 2, \ldots, n\}$ *per le costanti* $a_1, a_2, \ldots, a_n$ *e* $b_1, b_2, \ldots, b_n$*. Supponiamo che* $\boldsymbol{G}$ *sia una funzione continua definita da* $D \subset \mathbb{R}^n$ *in* $\mathbb{R}^n$ *e che* $\boldsymbol{G}(\boldsymbol{x}) \in D$ *se* $\boldsymbol{x} \in D$*. Allora* $\boldsymbol{G}$ *ha un punto fisso in* D*. Inoltre, supponiamo che tutte le funzioni* g_i *componenti di* $\boldsymbol{G}$ *abbiano derivate parziali continue e che esista una costante* $K < 1$ *tale che:*

$$\left| \frac{\partial g_i(\boldsymbol{x})}{\partial x_j} \right| \leq \frac{K}{n} \quad \textit{ogni volta che } \boldsymbol{x} \in D, \tag{3.2}$$

per ogni $j = 1, 2, \ldots, n$ *e per ogni funzione componente* g_i*.*
Allora, la sequenza $\{\boldsymbol{x}^{(k)}\}_{(k=0)}^{\infty}$ *definita da un valore scelto di approssimazione iniziale* $\boldsymbol{x}^{(0)}$ *in* D *e generata da:*

$$\boldsymbol{x}^{(k)} = G(\boldsymbol{x}^{(k-1)})$$

per ogni $k \geq 1$*, converge all'unico punto fisso* $\boldsymbol{p} \in D$ *e*

$$\|\boldsymbol{x}^{(k)} - \boldsymbol{p}\|_\infty \leq \frac{K^k}{1-K} \|\boldsymbol{x}^{(1)} - \boldsymbol{x}^{(0)}\|_\infty$$

Più avanti faremo un esempio pratico di questo metodo, implementandolo in Matlab e mettendolo anche a paragone con lo schema di Newton.

3.1.3 Metodo di Newton-Raphson per funzioni in n variabili

Abbiamo già visto il funzionamento di questo schema iterativo con una funzione a una variabile nel capitolo 2, ora cerchiamo di estenderlo a sistemi di n equazioni non lineari con n incognite. Qui sotto spiegheremo come si arriva all'espressione algebrica del metodo di Newton per una sola equazione non lineare e poi passeremo alla soluzione di un sistema non lineare mediante un semplice parallelismo.

Newton-Raphson con una sola incognita

Consideriamo una funzione non lineare $f(x) = 0 : \mathbb{R} \to \mathbb{R}$. Vogliamo trovare un valore di x tale per cui $f(x) = 0$.
Per farlo, consideriamo un valore iniziale $x^{(0)}$ e in generale sappiamo che $f(x^{(0)}) \neq 0$. Quindi, è necessario trovare un $\Delta x^{(0)}$ tale che $f(x^{(0)} + \Delta x^{(0)}) = 0$. Usando la serie di **Taylor**, possiamo esprimere $f(x^{(0)} + \Delta x^{(0)}) = 0$ come:

$$f(x^{(0)} + \Delta x^{(0)}) = f(x^{(0)}) + \Delta x^{(0)} \frac{df(x^{(0)})}{dx} + \frac{(\Delta x^{(0)})^2}{2} \frac{d^2 f(x^{(0)})}{dx^2} + \ldots$$

Considerando solo i primi due termini nell'equazione precedente e dal momento che stiamo cercando un $\Delta x^{(0)}$ tale che $f(x^{(0)} + \Delta x^{(0)}) = 0$ possiamo calcolare approssimativamente $\Delta x^{(0)}$ come:

$$\Delta x^{(0)} \approx -\frac{f(x^{(0)})}{\frac{df(x^{(0)})}{dx}}$$

Successivamente possiamo aggiornare x come:

$$x^{(1)} = x^{(0)} + \Delta x^{(0)}$$

Quindi, controlleremo se $f(x^{(1)}) = 0$. In tal caso avremo trovato un valore di x che soddisfa $f(x) = 0$. In caso contrario, invece, ripetiamo i passaggi appena visti per trovare $\Delta x^{(1)}$ in modo che $f(x^{(1)} + \Delta x^{(1)}) = 0$ e così via. In generale, possiamo calcolare $x^{(k)}$ (dove k è l'iterazione corrente) come:

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{\frac{df(x^{(k)})}{dx}}$$

Ad ogni iterazione, controlliamo se il modulo della differenza tra due iterazioni successive è minore di una certa tolleranza prefissata ϵ. Se $|x^{(k+1)} - x^{(k)}| < tol$ l'algoritmo ha raggiunto la convergenza e la soluzione è $x^{(k+1)}$, altrimenti proseguo con le iterazioni.

Newton-Raphson a più incognite

Ora estendiamo il metodo di Newton-Raphson appena visto al caso generale di un sistema di n equazioni lineari in n incognite, del tipo già visto $\boldsymbol{F(x)} = \mathbf{0}$.
Diamo un valore iniziale al vettore $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, ovvero $\boldsymbol{x}^{(0)}$. Abbiamo, in generale, che $\boldsymbol{F}(\boldsymbol{x}^{(0)}) \neq 0$, dove $\boldsymbol{F} = (f_1, f_2, \ldots, f_n)$.
Quindi, vogliamo trovare un $\Delta\boldsymbol{x}^{(0)}$ tale che $\boldsymbol{F}(\boldsymbol{x}^{(0)} + \Delta\boldsymbol{x}^{(0)}) = \mathbf{0}$. Usando la serie di **Taylor**, $\boldsymbol{F}(\boldsymbol{x}^{(0)} + \Delta\boldsymbol{x}^{(0)}) = \mathbf{0}$ può essere approssimativamente espresso come:

$$\boldsymbol{F}(\boldsymbol{x}^{(0)} + \Delta\boldsymbol{x}^{(0)}) \approx \boldsymbol{F}(\boldsymbol{x}^{(0)}) + \boldsymbol{J}^{(0)}\Delta\boldsymbol{x}^{(0)}$$

dove $\boldsymbol{J}$ è la matrice jacobiana $n \times n$:

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

Dal momento che noi stiamo cercando $\boldsymbol{F}(\boldsymbol{x}^{(0)} + \Delta\boldsymbol{x}^{(0)}) = \boldsymbol{0}$, possiamo calcolare $\Delta\boldsymbol{x}^{(0)}$ come:

$$\Delta\boldsymbol{x}^{(0)} \approx -[\boldsymbol{J}^{(0)}]^{-1}\boldsymbol{F}(\boldsymbol{x}^{(0)})$$

Quindi, possiamo aggiornare il vettore $\boldsymbol{x}$ come:

$$\boldsymbol{x}^{(1)} = \boldsymbol{x}^{(0)} + \Delta\boldsymbol{x}^{(0)}$$

In generale, possiamo aggiornare il vettore $\boldsymbol{x}$ nel modo seguente:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - [\boldsymbol{J}^{(k)}]^{-1}\boldsymbol{F}(\boldsymbol{x}^{(k)})$$

Tuttavia, risulta parecchio scomodo dover invertire la matrice jacobiana $\boldsymbol{J}(\boldsymbol{x})$ ad ogni iterazione. Possiamo evitare di calcolare $\boldsymbol{J}^{-1}(\boldsymbol{x})$ eseguendo lo stesso procedimento ma in un altro modo, che vediamo qui di seguito.
Innanzitutto troviamo un vettore $\boldsymbol{y}$ tale che $\boldsymbol{J}(\boldsymbol{x}^k)\boldsymbol{y} = -\boldsymbol{F}(\boldsymbol{x}^{(k)})$. Quindi, la nuova approssimazione $\boldsymbol{x}^{(k+1)}$ è ottenuta aggiungendo $\boldsymbol{y}$ a $\boldsymbol{x}^k$, ovvero:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \boldsymbol{y}$$

Il metodo di Newton-Raphson si interrompe quando $\|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\| < tol$.
Più avanti faremo un esempio pratico di questo schema, implementandolo in Matlab e mettendolo a paragone con il metodo di punto fisso.

3.1.4 Convergenza schemi di punto fisso e Newton-Raphson

Sia ξ una soluzione di $f(x) = 0$. Allora deve valere anche $\xi = g(\xi)$.
In questa formulazione, ξ è chiamato punto fisso della funzione g.
Noi siamo interessati alle condizioni per la convergenza dello schema di punto fisso.
Supponiamo che, in un intorno di ξ la derivata di g esista e sia limitata:

$$|g'(x)| \leq q < 1$$

dove il requisito $q < 1$ è essenziale.
Usando il Teorema del Valor Medio otteniamo:

$$g(x) - g(\xi) = g'(\theta)(x - \xi), \quad \theta \in (x, \xi) \tag{3.3}$$

che conduce al seguente risultato:

$$|g(x) - g(\xi)| \leq q|x - \xi|.$$

Se l'approssimazione iniziale x_0 appartiene ad un intorno di ξ allora:

$$|x_1 - \xi| = |g(x_0) - g(\xi)| \leq q|x_0 - \xi|$$

Quindi, dal momento che $q < 1$, x_1 è più vicina alla soluzione esatta ξ rispetto a x_0. Questo stesso ragionamento può essere ripetuto con x_0 sostituito da x_1 e x_1 sostituito da x_2, e si ottiene:

$$|x_2 - \xi| = |g(x_1) - g(\xi)| \leq q|x_1 - \xi| \leq q^2|x_0 - \xi|$$

Di conseguenza,

$$|x_k - \xi| \leq q^k|x_0 - \xi|.$$

In conclusione, il metodo di punto fisso è convergente a condizione che $q < 1$ e che il valore iniziale x_0 sia sufficientemente vicino alla soluzione esatta.

Per vedere la convergenza del metodo di Newton-Raphson invece posso vedere questo schema come un caso particolare dello schema del punto fisso applicato alla funzione

$$g(x) = x - \frac{f(x)}{f'(x)}$$

con l'ipotesi che $f'(x)$ sia diversa da zero.
Infatti, se $f(\xi) = 0$, con ξ soluzione esatta, allora $g(\xi) = \xi - f(\xi)/f'(\xi) = \xi$. Come abbiamo appena visto, perchè lo schema di punto fisso converga devo avere $|g'(x)| < 1$ in un intorno di ξ. Svolgendo la derivata della funzione otteniamo:

$$|g'(x)| = \left|1 - \frac{f'(x)^2 - f(x)f''(x)}{f'(x)^2}\right| = \left|\frac{f(x)f''(x)}{f'(x)^2}\right|$$

Supponendo $f'(\xi) \neq 0$ (che è il caso in cui la radice non è multipla), si ha $|g'(\xi)| = 0$, poichè al numeratore $f(\xi) = 0$ (essendo ξ radice della f). Per continuità, allora, vale $|g'(x)| < 1$ in un intorno di ξ. Pertanto possiamo affermare che il metodo di Newton-Raphson sia generalmente convergente.

Ora che abbiamo mostrato che i metodi convergono, vogliamo capire **come** convergono; mostriamo che lo schema di punto fisso converge linearmente mentre quello di Newton-Raphson quadraticamente.
Riprendiamo la *definizione 2* della *sezione 2.3* che ci dice che un metodo ha ordine di convergenza p se si possono definire due costanti $p \geq 1$ e $M > 0$ tali che $|x_{n+1} - \xi| \leq M|x_n - \xi|^p$.
Nel metodo di punto fisso, nell'ipotesi in cui $g'(\xi) \neq 0$, la convergenza è lineare e questo significa che $p = 1$.
Innanzitutto definiamo gli errori $\varepsilon_{n+1} = x_{n+1} - \xi$ e $\varepsilon_n = x_n - \xi$. Considerando ora gli errori cambiati di segno, la relazione fondamentale dello schema di punto fisso $x_{n+1} = g(x_n)$ può essere riscritta nel seguente modo:

$$\xi + \varepsilon_{n+1} = g(\xi + \varepsilon_n)$$

Applicando ora la formula polinomiale di Taylor si ha:

$$\xi + \varepsilon_{n+1} = g(\xi) + \varepsilon_n g'(\xi) + \ldots \longmapsto \xi + \varepsilon_{n+1} = \xi + \varepsilon_n g'(\xi) + \ldots$$

Semplificando poi ξ da entrambe le parti otteniamo:

$$\varepsilon_{n+1} = \varepsilon_n g'(\xi) + \ldots$$

e, al limite per $n \mapsto \infty$, ottengo infine:

$$\varepsilon_{n+1} = g'(\xi)\varepsilon_n$$

Quindi, nell'ultimo passaggio sono ben visibili l'ordine di convergenza $p = 1$ (metodo converge linearmente) e il fattore di convergenza $M = |g'(\xi)|$.

Nello schema di Newton-Raphson, invece, per vedere come si riduce l'errore via via che le approssimazioni si avvicinano alla soluzione esatta ξ, consideriamo gli errori ε_n e ε_{n+1} da sostituire nell'espressione del metodo di Newton così scritta: $x_{n+1} = x_n - [f(x_n)/f'(x_n)]$. Otteniamo quindi:

$$\varepsilon_{n+1} + \xi = \varepsilon_n + \xi - \frac{\xi + \varepsilon_n}{\xi + \varepsilon_n} \longmapsto \varepsilon_{n+1} = \varepsilon_n - \frac{\xi + \varepsilon_n}{\xi + \varepsilon_n}$$

Applicando la formula polinomiale di Taylor sia su f che su f' di centro ξ, si ha:

$$\varepsilon_{n+1} = \varepsilon_n - \frac{f(\xi) + \varepsilon_n f'(\xi) + \varepsilon_n^2 f''(\xi)/2 + \ldots}{f'(\xi) + \varepsilon_n f''(\xi) + \ldots}$$

Poichè $f(\xi) = 0$, raccogliendo i termini otteniamo:

$$\varepsilon_{n+1} = \frac{\varepsilon_n f'(\xi) + \varepsilon_n^2 f''(\xi) - \varepsilon_n f'(\xi) - \varepsilon_n^2 f''(\xi)/2 + \ldots}{f'(\xi) + \varepsilon_n f''(\xi) + \ldots} = \frac{\varepsilon_n^2 f''(\xi)/2 + \ldots}{f'(\xi) + \varepsilon_n f''(\xi) + \ldots}$$

Trascurando i termini dal secondo grado in poi ($\varepsilon_n f''(\xi) + \ldots$) al denominatore e le potenze maggiori o uguali e ε_n^3 al numeratore, troviamo:

$$\varepsilon_{n+1} = \frac{f''(\xi)}{2f'(\xi)}\varepsilon_n^2 = A\varepsilon_n^2$$

L'ultima relazione ottenuta ci dice che l'errore al passo $n+1$ è proporzionale, secondo il fattore A, al quadrato dell'errore al passo precedente, quindi possiamo parlare di convergenza quadratica per lo schema di Newton-Raphson, dove quindi l'ordine di convergenza vale $p = 2$ e il fattore di convergenza $M = |A| = f''(\xi)/2f'(\xi)$. Infatti, tenendo conto del fatto che abbiamo considerato l'errore cambiato di segno,

quando prendiamo i valori assoluti, il legame tra i valori assoluti degli errori è tramite $M = |A|$.

Tuttavia, sorge immediatamente un problema, ovvero il fatto che non sempre, anzi quasi mai, abbiamo a disposizione la soluzione esatta ξ, e di conseguenza non possiamo sapere il valore dell'errore ($\varepsilon_n = x_n - \xi$) e quindi operiamo con gli scarti, cioè con la differenza tra un'iterazione e la sua precedente ($x_{n+1} - x_n$), valori di cui abbiamo conoscenza, per misurare la bontà dell'approssimazione della radice.
Come per gli errori, se gli scarti si riducono via via che le iterazioni aumentano, allora il metodo sta portando a convergenza.
Per il **metodo di Newton-Raphson** possiamo dimostrare da un punto di vista teorico che lo scarto approssima bene l'errore.
Consideriamo la radice ξ e l'approssimazione finale x_n e applichiamo il Teorema del Valor Medio, scriviamo:

$$f(\xi) - f(x_n) = f'(\xi_n)(\xi - x_n)$$

dove $\xi < \xi_n < x_n$. Poi sappiamo che $f(\xi) = 0$ e che, se x_n è una buona approssimazione della radice, possiamo considerare $\xi_n \approx x_n$, quindi otteniamo:

$$-f(x_n) \approx f'(x_n)(\xi - x_n)$$

Sostituendo questa espressione allo schema di Newton, si ricava:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \approx x_n + (\xi - x_n)$$

Possiamo riscriverla come:

$$x_{n+1} - x_n \approx \xi - x_n \longmapsto d_{n+1} \approx \varepsilon_n$$

In condizioni di convergenza $d_{n+1} < d_n$ da cui, per l'errore vale la maggiorazione $\varepsilon_n < d_n$. Possiamo quindi concludere dicendo che gli scarti sono molto vicini agli errori e possono essere usati sia per controllare il numero di iterazioni da effettuare per approssimare la radice entro una certa tolleranza sia per approssimare M usando la formula $M \approx d_{n+1}/d_n^2$.

Dopo aver dimostrato la possibilità di approssimare ξ tramite gli scarti invece che tramite gli errori per un metodo che converge quadraticamente, ora vogliamo dimostrarlo per metodi lineari, e quindi per lo **schema di punto fisso**.
Sia tol la tolleranza richiesta per approssimare ξ con gli scarti. Sappiamo che se il metodo converge vale la relazione: $\varepsilon_{n+1} \approx M\varepsilon_n$, dove $M < 1, M \neq 0$ è la costante

asintotica.
Riscriviamo la precedente formula come:

$$|\xi - x_{n+1}| \approx M|\xi - x_n| = M|\xi - x_n + x_{n+1} + x_{n+1}| \leq M(|\xi - x_{n+1}| + |x_{n+1} - x_n|)$$

Quindi si ottiene:

$$\varepsilon_{n+1} \leq M(\varepsilon_{n+1} + d_{n+1}) \longmapsto (1 - M)\varepsilon_{n+1} \leq M d_{n+1}$$

Se siamo arrivati ad un punto tale per cui $d_{n+1} \leq tol$, allora:

$$\varepsilon_{n+1} \leq \frac{M}{1 - M} d_{n+1} \leq \frac{M}{1 - M} tol$$

Quindi, per $M/(1 - M) < 1$ (ovvero $M < 1/2$), se $d_{n+1} \leq tol$ anche $\varepsilon_{n+1} \leq tol$. Se, invece, $M \geq 1/2$, allora l'errore può essere un pò più grande della tolleranza richiesta.
Abbiamo quindi visto come gli scarti approssimino bene gli errori anche per schemi lineramente convergenti.
Asintoticamente, quindi, nella definizione di ordine di convergenza di un metodo, possiamo sostituire l'errore con lo scarto in questo modo: $d_n \approx M d_{n-1}^p$.

3.2 Matlab: risoluzione di un sistema di due funzioni non lineari

Implementiamo i metodi visti finora in Matlab, facendo un esempio di sistema a due funzioni in due incognite. Questo sistema con due variabili x_1, x_2 verrà risolto con entrambi gli schemi iterativi (punto fisso e Newton-Raphson), in modo da mettere a paragone le velocità di convergenza.
Noi faremo qui solo un esempio in due variabili, dal momento che in questo modo la soluzione può essere anche vista graficamente; per sistemi con tre o più variabili, comunque, il modus operandi sarà sempre lo stesso che vedremo qui di seguito.
Inoltre, per rendere più generale il nostro esempio, scriveremo una **function** da richiamare poi nello script, così da poter cambiare i dati di input (che devono essere passati dall'esterno) come e quanto vogliamo.

Schema di Newton-Raphson

Definiamo due funzioni $f_1(x_1, x_2) = 2x_1 + x_2 - (x_1 x_2)/2 - 2$ e $f_2(x_1, x_2) = x_1 + 2x_2 - cos(x_2)/2 - 3/2$ e le poniamo nulle; vogliamo infatti trovare la loro intersezione (visibile anche graficamente). Questo sistema a due incognite può essere risolto attraverso lo schema iterativo di Newton-Raphson, che ci permette di

determinare la soluzione approssimata (x_1, x_2).
Creiamo uno script in Matlab contenente la function di Newton-Raphson per funzioni non lineari a più variabili:

```
function[xnew,iter,vettscarti]=newtonmultiplo(f1,f2,df11,df12,df21,df22,
x0,toll,itmax)
% scrivo questa function per risolvere sistemi di due funzioni
% f1 e f2 in due incognite x1 e x2, in modo del tutto generale
% gli output sono il vettore xnew, che è la soluzione approssimata,
% il numero di iter necessarie per la convergenza del metodo,
% il vettore degli scarti, in cui ogni componente è la norma
% della differenza tra i vettori xnew e xold ad ogni iterazione
% gli input, che saranno dati da uno script esterno sono le due
% funzioni f1 e f2, le rispettive derivate parziali, il vettore x0
% di approssimazione iniziale, una tolleranza toll e il numero max di iter
iter=0; % inizializzo la variabile di conteggio delle iterazioni
scarto=2*toll; % valore fittizio per entrare nel ciclo while
vettscarti=zeros(itmax,1); % preallocazione vettore scarti
xold=x0;
% entro nel ciclo while per svolgere le iterazioni necessarie
while scarto>=toll && iter<=itmax
    iter=iter+1;
    F=[f1(xold(1),xold(2)); f2(xold(1),xold(2))];
    J=[df11(xold(1),xold(2)) df12(xold(1),xold(2));
        df21(xold(1),xold(2)) df22(xold(1),xold(2))];
    y=-J\F;
    xnew=xold+y;
    scarto=norm(xnew-xold);
    vettscarti(iter)=scarto;
    xold=xnew; % aggiorno la variabile
end % fine del ciclo while
vettscarti=vettscarti(1:iter); % taglio il vettore scarti
end % fine della function
```

Ora in uno script a parte richiamiamo questa **function** e inseriamo manualmente i dati di input, quali le due funzioni $f1$ e $f2$, le derivate parziali, l'approssimazione iniziale x_0, la tolleranza e il numero massimo di iterazioni. Nella command window dovremmo avere la soluzione al mio problema, ovvero (x_1, x_2), l'intersezione delle due funzioni.

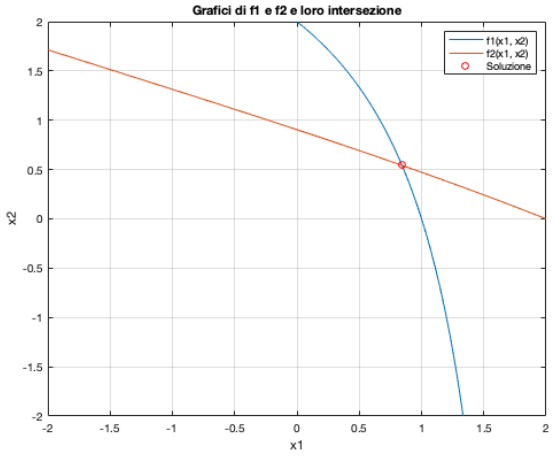
```
% definiamo le funzioni f1(x1,x2) e f2(x1,x2)
% definiamo anche le derivate parziali di f1 e f2 rispetto a x1 e x2
clear
close all
f1 = @(x1,x2) 2*x1+x2-((x1.*x2)/2)-2;
f2 = @(x1,x2) x1+2*x2-(cos(x2)/2)-3/2;
df11 = @(x1,x2) 2-x2/2;
df12 = @(x1,x2) 1-x1/2;
```

```
df21 = @(x1,x2) 1;
df22 = @(x1,x2) 2+(sin(x2)/2);
x0=[1;0.5]; % vettore x0 di approssimazione iniziale
toll = 1e-6;
itmax=100;
% dopo aver scritto gli input richiamo la function newtonmultiplo
[xnew, iter, vettscarti]=newtonmultiplo(f1,f2,df11,df12,df21,df22,x0,toll,itmax);
% stampo sulla command window le soluzioni
fprintf(1,'Soluzione approssimata: x1= %f \n',xnew(1));
fprintf(1,'Soluzione approssimata: x2= %f \n',xnew(2));
fprintf(1,'Numero di iterazioni effettuate: %i \n', iter);
fprintf(1,'vettore scarti: %12.8e \n',vettscarti);
% faccio il grafico dove visualizzo f1,f2 e l'intersezione (x1,x2)
fimplicit(f1, [-2, 2, -2, 2], 'DisplayName', 'f1(x1, x2)');
hold on;
fimplicit(f2, [-2, 2, -2, 2], 'DisplayName', 'f2(x1, x2)');
hold on
plot(xnew(1),xnew(2), 'ro', 'DisplayName', 'Soluzione');
legend;
grid on;
xlabel('x1');
ylabel('x2');
title('Grafici di f1 e f2 e loro intersezione');
% definisco il vettore delle iterazioni
vettiter=1:iter;
% apro un'altra figura per il grafico della convergenza
figure
semilogy(vettiter,vettscarti)
title('grafico di convergenza')
```

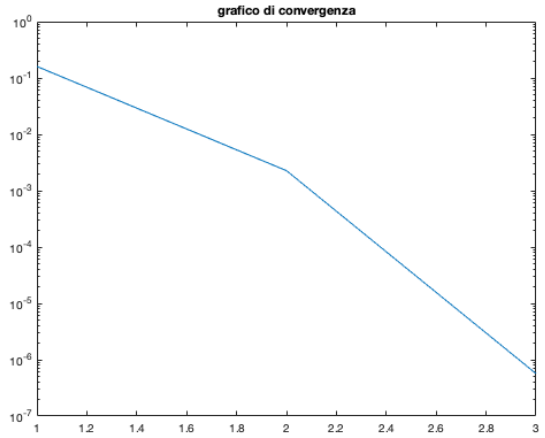
Sulla Command Window appaiono i risultati in questo modo:

```
>> scriptNR
Soluzione approssimata: x1= 0.843075
Soluzione approssimata: x2= 0.542560
Numero di iterazioni effettuate: 3
vettore scarti: 1.60336371e-01
vettore scarti: 2.26558703e-03
vettore scarti: 5.67328902e-07
```

Invece, i grafici (convergenza e soluzione $\boldsymbol{x}$) sono riportati di seguito:



(a) grafici di $f1, f2$ e la loro intersezione $\boldsymbol{x}$



(b) Convergenza quadratica Newton-Raphson multiplo

Schema di punto fisso

Ora possiamo trasformare il precedente sistema di due funzioni in due incognite in un sistema equivalente da risolvere però con il metodo di punto fisso, in modo da paragonare le velocità di convergenza dei due schemi.
L'esempio precedente è quindi equivalente a scrivere:

$$\begin{cases} g(x_1, x_2) = x_1 = (2 + (x_1 x_2)/2 - x_2)/2 \\ g(x_1, x_2) = x_2 = (3/2 + \cos(x_2)/2 - x_1)/2 \end{cases}$$

Vogliamo ora trovare il punto fisso, cioè il vettore $\boldsymbol{x} = [x_1, x_2]$, punto fisso delle funzioni $g1$ e $g2$, attraverso lo schema di punto fisso.
Scriviamo quindi una function di punto fisso in Matlab:

```
function[xnew,iter,vettscarti]=puntofissomultiplo(g1,g2,x0,toll,itmax)
% scrivo questa function per risolvere sistemi di due funzioni g1 e g2
% % gli output sono il vettore xnew, che è il vettore di punto fisso,
% il numero di iter necessarie per la convergenza del metodo,
% il vettore degli scarti, in cui ogni componente è la norma
% della differenza tra i vettori xnew e xold ad ogni iterazione
% gli input, che saranno dati da uno script esterno sono le due
% funzioni g1 e g2, il vettore x0 di approssimazione iniziale,
% una tolleranza toll e il numero massimo di iterazioni itmax
iter=0; % inizializzo la variabile di conteggio delle iterazioni
scarto=2*toll; % valore fittizio per entrare nel ciclo while
vettscarti=zeros(itmax,1); % preallocazione vettore scarti
xold=x0;
% entro nel ciclo while per svolgere le iterazioni necessarie
while scarto>=toll && iter<=itmax
    iter=iter+1;
    xnew=[g1(xold(1),xold(2)),g2(xold(1),xold(2))];
```

```
    scarto=norm(xnew-xold);
    vettscarti(iter)=scarto;
    xold=xnew; % aggiorno la variabile
end % fine del ciclo while
vettscarti=vettscarti(1:iter); % taglio il vettore scarti
end % fine della function
```

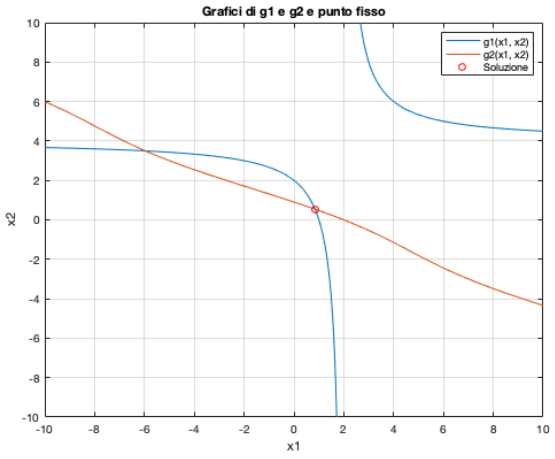
Creiamo poi uno script in cui inserisco i dati di input, e quindi i valori delle due funzioni.

```
% definiamo le funzioni g1(x1,x2) e g2(x1,x2)
clear
close all
g1 = @(x1,x2) (2+(x1.*x2/2)-x2)/2;
g2 = @(x1,x2) (3/2+(cos(x2)/2)-x1)/2;
x0=[1;0.5]; % vettore x0 di approssimazione iniziale
toll = 1e-6;
itmax=100;
% dopo aver scritto gli input richiamo la function di punto fisso
[xnew,iter,vettscarti]=puntofissomultiplo(g1,g2,x0,toll,itmax);
% stampo sulla command window le soluzioni
fprintf(1,'Soluzione approssimata: x1= %f \n',xnew(1));
fprintf(1,'Soluzione approssimata: x2= %f \n',xnew(2));
fprintf(1,'Numero di iterazioni effettuate: %i \n', iter);
fprintf(1,'vettore scarti: %12.8e \n',vettscarti);
% faccio il grafico dove visualizzo g1,g2 e il punto fisso (x1,x2)
% graficamente il punto fisso [x1,x2] sarà il punto in cui
% le due funzioni g1 e g2 si intersecano o si avvicinano molto
gg1 = @(x1,x2) (2+(x1.*x2/2)-x2)/2-x1;
gg2 = @(x1,x2) (3/2+(cos(x2)/2)-x1)/2-x2;
fimplicit(gg1, [-10, 10, -10, 10], 'DisplayName', 'g1(x1, x2)');
hold on;
fimplicit(gg2, [-10, 10, -10, 10], 'DisplayName', 'g2(x1, x2)');
hold on
plot(xnew(1),xnew(2), 'ro', 'DisplayName', 'Soluzione');
legend;
grid on;
xlabel('x1');
ylabel('x2');
title('Grafici di g1 e g2 e punto fisso');
% definisco il vettore delle iterazioni
vettiter=1:iter;
% apro un'altra figura per il grafico della convergenza
figure
semilogy(vettiter,vettscarti)
title('grafico di convergenza')
```

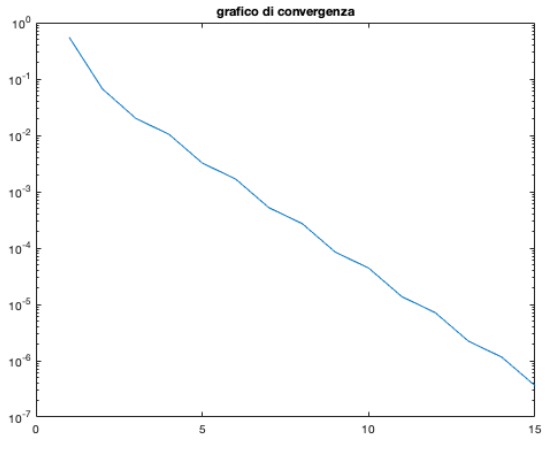
Sulla Command Window appaiono i risultati nel seguente modo:

```
>> scriptPF
Soluzione approssimata: x1= 0.843075
Soluzione approssimata: x2= 0.542560
Numero di iterazioni effettuate: 15
vettore scarti: 5.50436667e-01
vettore scarti: 6.64364790e-02
vettore scarti: 2.00194410e-02
vettore scarti: 1.04105321e-02
vettore scarti: 3.21949050e-03
vettore scarti: 1.67734964e-03
vettore scarti: 5.20857532e-04
vettore scarti: 2.71582549e-04
vettore scarti: 8.43988432e-05
vettore scarti: 4.40235900e-05
vettore scarti: 1.36875288e-05
vettore scarti: 7.14013007e-06
vettore scarti: 2.22142624e-06
vettore scarti: 1.15856160e-06
vettore scarti: 3.60783709e-07
```

I grafici (convergenza e punto fisso) appaiono in questo modo:



(a) grafici di $g1$,$g2$ e punto fisso



(b) Convergenza lineare Punto Fisso

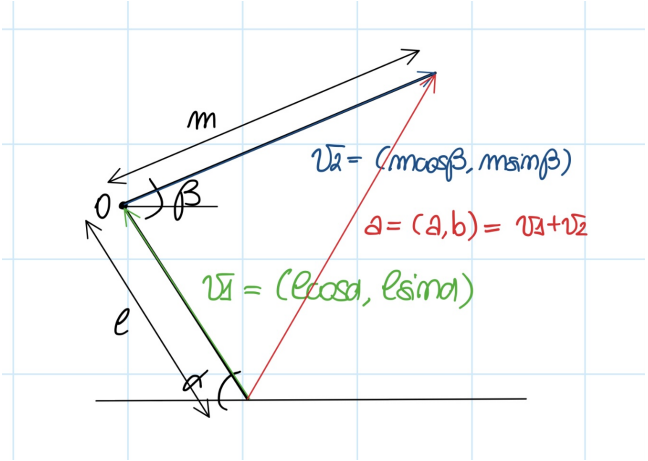
Come possiamo vedere, rispetto all'esempio predecente, con questo sistema equivalente otteniamo la stessa soluzione $[x_1, x_2]$ di prima, ma le iterazioni necessarie per raggiungere la convergenza sono decisamente un numero maggiore; inoltre, come mostrato dal grafico (b) la convergenza è ora lineare, confrontata con quella quadratica di Newton-Raphson.

Capitolo 4

Applicazioni reali

In questo capitolo vedremo due applicazioni reali di sistemi di funzioni non lineari risolvibili mediante i metodi iterativi visti finora.

4.1 Esempio del braccio robotico



Un braccio robotico è formato da due aste rigide collegate tra loro da un punto nel piano che noi assumiamo come origine **O**. Le aste sono libere di ruotare e il problema consiste nel configurarle in modo che l'estremità superiore del robot si trovi nella posizione stabilita $\boldsymbol{a} = (a, b)^T$.

La prima asta ha lunghezza l e crea una angolo α con l'orizzontale, quindi termina nella posizione $\boldsymbol{v_1} = (l\cos\alpha, l\sin\alpha)^T$.
La seconda asta ha lunghezza m e crea un angolo β con l'orizzontale; è rappresentata dal vettore $\boldsymbol{v_2} = (m\cos\beta, m\sin\beta)^T$.
La mano del robot, fissata all'estremità finale del secondo braccio, è dunque nella posizione $\boldsymbol{v_1}+\boldsymbol{v_2}$ e noi vogliamo determinare dei valori di α e β per i quali $\boldsymbol{v_1}+\boldsymbol{v_2} = \boldsymbol{a}$. A tal fine, dobbiamo risolvere il seguente sistema di equazioni:

$$\begin{cases} l\cos\alpha + m\cos\beta = a \\ l\sin\alpha + m\sin\beta = b \end{cases}$$

Per trovare la soluzione, applichiamo il metodo di Newton-Raphson, implementandolo in Matlab (riportiamo gli script di seguito).

Prima però diamo dei valori alle nostre grandezze; assumiamo $l = 2$, $m = 1$, $\boldsymbol{a} = (1, 1)^T$. Per cominciare ad iterare ci serve un'approssimazione iniziale, quindi scegliamo $\alpha^{(0)} = 0$ e $\beta^{(0)} = \pi/2$, quindi la prima asta giace lungo l'asse delle ascisse, mentre la seconda asta è a lei perpendicolare. Riportiamo qui la function 'newtonmultiplo' già utilizzata in precedenza leggermente modificata.

```
function[matxnew,iter,vettscarti]=functionbraccio(f1,f2,df11,df12,df21,df22,
x0,toll,itmax)
% è la stessa function di 'newtonmulptiplo' con delle piccole modifiche
% voglio una matrice dove conservare i valori di xold ad ogni iterazione
% in output voglio la matrice matxnew e non solo xnew ultima iterazione
matxnew=zeros(itmax,2);
scarto=2*toll; % valore fittizio per entrare nel ciclo while
vettscarti=zeros(itmax,1); % preallocazione vettore scarti
iter=0; % inizializzo la variabile di conteggio delle iterazioni
xold=x0;
% entro nel ciclo while per svolgere le iterazioni necessarie
while scarto>=toll && iter<=itmax
    iter=iter+1;
    F=[f1(xold(1),xold(2)); f2(xold(1),xold(2))];
    J=[df11(xold(1),xold(2)) df12(xold(1),xold(2));
        df21(xold(1),xold(2)) df22(xold(1),xold(2))];
    y=-J\F;
    xnew=xold+y;
    matxnew(iter,:)=xnew;
    scarto=norm(xnew-xold);
    vettscarti(iter)=scarto;
    xold=xnew; % aggiorno la variabile
end % fine del ciclo while
vettscarti=vettscarti(1:iter); % taglio il vettore scarti
matxnew=matxnew(1:iter,:); % taglio la matrice delle iterazioni
end % fine della function
```

Poi possiamo vedere lo script in cui viene richiamata la function:

```
clear
close all
% devo risolvere il sistema di funzioni f1(x1,x2) e f2(x1,x2)
% per determinare la radice (x1,x2)=(alpha,beta)
% per semplificare ho chiamato infatti x1=alpha e x2=beta
l=2; m=1; a=1; b=1; % definiamo le variabili
% definiamo le due funzioni
f1 = @(x1,x2) l*cos(x1)+m*cos(x2)-a;
f2 = @(x1,x2) l*sin(x1)+m*sin(x2)-b;
% definiamo le derivate parziali df11,df12,df21,df22
df11 = @(x1,x2) -l*sin(x1);
df12 = @(x1,x2) -m*sin(x2);
df21 = @(x1,x2) l*cos(x1);
df22 = @(x1,x2) m*cos(x2);
```

```
% definiamo anche gli altri input della function
toll = 1e-6;
itmax=100;
x0=[0;pi/2]; % vettore approssimazione iniziale
% richiamo la funcion functionbraccio
[matxnew,iter,vettscarti]=functionbraccio(f1,f2,df11,df12,df21,df22,
x0,toll,itmax);
% ora determino altri vettori, vettc, vettd, vette, vettf
% v1=(vettc,vettd), dove v1 indica la posizione del gomito del braccio
% meccanico ad ogni iterazone, mentre v2=(vette,vettf), dove v2 indica
% la posizione della mano robotica ad ogni iterazione
vettc=l*cos(matxnew(:,1));
vettd=l*sin(matxnew(:,1));
vette=m*cos(matxnew(:,2));
vettf=m*sin(matxnew(:,2));
% creo i vettori con tutte le iterazioni di alpha(x1) e di beta(x2)
vettx1=matxnew(:,1);
vettx2=matxnew(:,2);
vettiter=1:iter;% creo il vettore iterazioni
vettiterT=vettiter'; % vettore iterazioni trasposto
% stampo sulla command window le soluzioni in tabella
% creo una prima tabella
% sulla prima colonna vettiterT, sulla seconda x1=alpha, sulla terza
% x2=beta e sull'ultima colonna gli scarti ad ogni iterazione
% specifico il formato di stampa per ciascuna colonna
formatoiter = '%d';
formatoalpha = '%f';
formatobeta = '%f';
formatoscarti = '%12.8e';
T1=table(vettiterT,vettx1,vettx2,vettscarti,'VariableNames',{'iter','alpha','beta',
% Applico i formati alle colonne specifiche
T1.iter=arrayfun(@(x)sprintf(formatoiter,x),T1.iter,'UniformOutput',false);
T1.alpha=arrayfun(@(x)sprintf(formatoalpha,x),T1.alpha,'UniformOutput',false);
T1.beta=arrayfun(@(x)sprintf(formatobeta,x),T1.beta,'UniformOutput',false);
T1.scarti=arrayfun(@(x)sprintf(formatoscarti,x),T1.scarti,'UniformOutput',false);
% creo una seconda tabella
% nelle quattro colonne metto v1=(x1,y1) e v2=(x2,y2) ad ogni iterazione
% specifico il formato di stampa per ciascuna colonna
formatox1 = '%f';
formatoy1 = '%f';
formatox2 = '%f';
formatoy2 = '%f';
T2=table(vettc,vettd,vette,vettf,'VariableNames',{'x1','y1','x2','y2'});
T2.x1=arrayfun(@(x)sprintf(formatox1,x),T2.x1,'UniformOutput',false);
T2.y1=arrayfun(@(x)sprintf(formatoy1,x),T2.y1,'UniformOutput',false);
T2.x2=arrayfun(@(x)sprintf(formatox2,x),T2.x2,'UniformOutput',false);
T2.y2=arrayfun(@(x)sprintf(formatoy2,x),T2.y2,'UniformOutput',false);
% creo una terza tabella in cui mostro che v1(1)+v2(1)=a
% e che v1(2)+v2(2)=b ad ogni iterazione
```

```
% creo i vettori a e b ad ogni iterazione
vetta = vettc+vette;
vettb = vettd+vettf;
% specifico il formato di stampa per ciascuna colonna
formatoa = '%f';
formatob = '%f';
T3=table(vettiterT,vetta,vettb,'VariableNames',{'iter','a','b'});
% Visualizzo le tabelle
disp(T1);
disp(T2);
disp(T3);
```

Di seguito il risultato dello script, visibile sulla Command Window:

```
>> bracciomeccanico
    iter         alpha            beta              scarti
    _____    _____________    _____________    ___________________

    {'1'}    {'-0.000000'}    {'2.570796'}    {'1.00000000e+00'}
    {'2'}    {'0.353296' }    {'2.864204'}    {'4.59245377e-01'}
    {'3'}    {'0.291670' }    {'2.708440'}    {'1.67511626e-01'}
    {'4'}    {'0.298700' }    {'2.717589'}    {'1.15373947e-02'}
    {'5'}    {'0.298703' }    {'2.717562'}    {'2.77246497e-05'}
    {'6'}    {'0.298703' }    {'2.717562'}    {'4.96066397e-10'}

         x1              y1              x2              y2
    _____________    _____________    _____________    _____________

    {'2.000000'}    {'-0.000000'}    {'-0.841471'}    {'0.540302'}
    {'1.876475'}    {'0.691984' }    {'-0.961774'}    {'0.273845'}
    {'1.915530'}    {'0.575105' }    {'-0.907647'}    {'0.419734'}
    {'1.911440'}    {'0.588555' }    {'-0.911449'}    {'0.411413'}
    {'1.911438'}    {'0.588562' }    {'-0.911438'}    {'0.411438'}
    {'1.911438'}    {'0.588562' }    {'-0.911438'}    {'0.411438'}

    iter       a          b
    ____    _______    _______

     1       1.1585     0.5403
     2       0.9147    0.96583
     3       1.0079    0.99484
     4      0.99999    0.99997
     5            1          1
     6            1          1
```

Dopo solo sei iterazioni, abbiamo visto che lo schema converge e arriviamo ad una soluzione: $\alpha \approx 0.29$ e $\beta \approx 2.71$.
Abbiamo quindi trovato gli angoli α e β che le due aste formano con l'orizzontale tali per cui la mano del braccio robotico può raggiungere la posizione $\boldsymbol{a} = (a, b)$.

Nella seconda tabella troviamo poi i valori $\boldsymbol{v1}$ e $\boldsymbol{v2}$, che indicano rispettivamente la posizione del gomito del braccio meccanico e la posizione della mano robotica (prendendo come origine $\mathbf{O}$ il gomito).
Nella terza tabella, infine, troviamo ad ogni iterazione $\boldsymbol{a}(1) = a = v_1(1) + v_2(1)$ e $\boldsymbol{a}(2) = b = v_1(2) + v_2(2)$, dimostrando che ad ogni iterazione successiva ci avviciniamo sempre più alla posizione finale $\boldsymbol{a} = (1, 1)$.

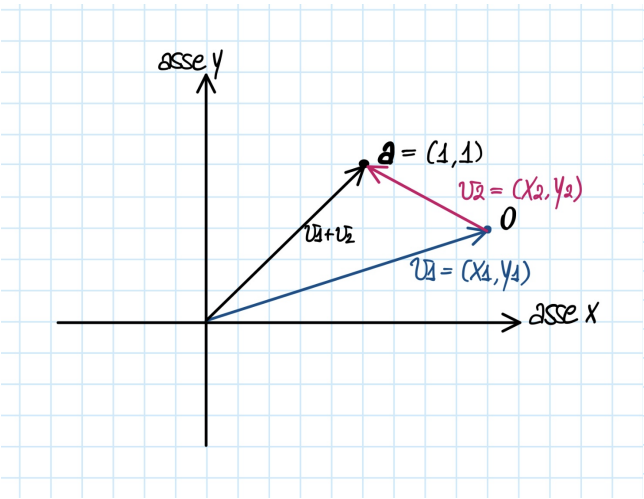
Se provassimo poi nello script a cambiare i valori delle variabili iniziali, imponendo ad esempio, $l = 1$, $m = 1$, $\boldsymbol{a} = (2, 2)$, vedremmo che le iterazioni superano la condizione $itmax = 100$, quindi il metodo non converge.
Infatti, se $||\boldsymbol{a}|| > l + m$, non ci sono soluzioni possibili, poiché i bracci sono troppo corti perché la mano possa raggiungere la posizione desiderata e il braccio robotico si muoverà in modo caotico.

Facciamo un'ultima prova, dove $l = 2, m = 3, \boldsymbol{a} = (1, 1)$ e vediamo che risultati otteniamo sulla Command Window di Matlab:

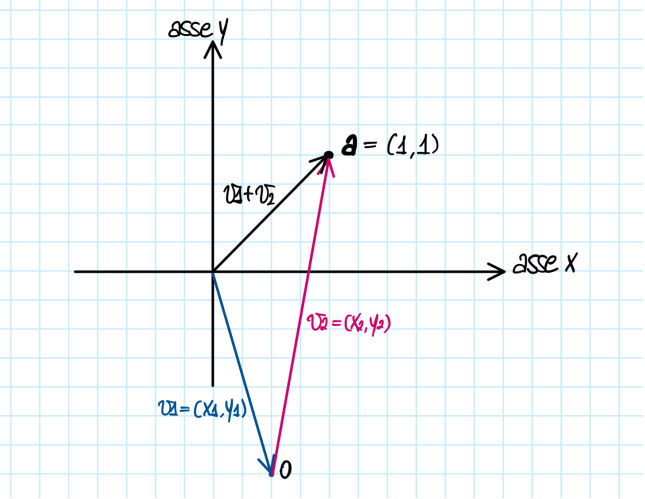
```
>> bracciomeccanico
    iter         alpha            beta              scarti
    _____    _____________    _____________    ____________________

    {'1'}    {'-1.000000'}    {'1.904130'}    {'1.05409255e+00'}
    {'2'}    {'-1.931743'}    {'1.033173'}    {'1.27542550e+00'}
    {'3'}    {'-0.964818'}    {'1.669317'}    {'1.15742026e+00'}
    {'4'}    {'-1.330329'}    {'1.415817'}    {'4.44814373e-01'}
    {'5'}    {'-1.345874'}    {'1.385182'}    {'3.43534971e-02'}
    {'6'}    {'-1.344387'}    {'1.386068'}    {'1.73069680e-03'}
    {'7'}    {'-1.344388'}    {'1.386067'}    {'1.42370016e-06'}
    {'8'}    {'-1.344388'}    {'1.386067'}    {'3.82348689e-13'}

         x1               y1               x2               y2
    _____________    _____________    _____________    _____________

    {'1.080605' }    {'-1.682942'}    {'-0.981584'}    {'2.834871'}
    {'-0.706320'}    {'-1.871126'}    {'1.536288' }    {'2.576785'}
    {'1.139132' }    {'-1.643891'}    {'-0.295083'}    {'2.985452'}
    {'0.476313' }    {'-1.942454'}    {'0.463078' }    {'2.964044'}
    {'0.446061' }    {'-1.949623'}    {'0.553651' }    {'2.948469'}
    {'0.448960' }    {'-1.948957'}    {'0.551039' }    {'2.948958'}
    {'0.448958' }    {'-1.948958'}    {'0.551042' }    {'2.948958'}
    {'0.448958' }    {'-1.948958'}    {'0.551042' }    {'2.948958'}
```

iter	a	b
1	0.099021	1.1519
2	0.82997	0.70566
3	0.84405	1.3416
4	0.93939	1.0216
5	0.99971	0.99885
6	1	1
7	1	1
8	1	1

Il metodo converge, ma con più iterazioni rispetto al primo tentativo.
Avendo cambiato solo le lunghezze delle aste rispetto a prima, quindi, il braccio robotico raggiunge la stessa posizione $\boldsymbol{a} = (1, 1)$ ma con una configurazione diversa delle aste, che si dispongno in maniera differente rispetto alla prima situazione, come mostrato nelle rappresentazioni grafiche qui di seguito (primo braccio disegnato in blu, mentre il secondo in rosa).



(a) prima prova con $l = 2, m = 1$



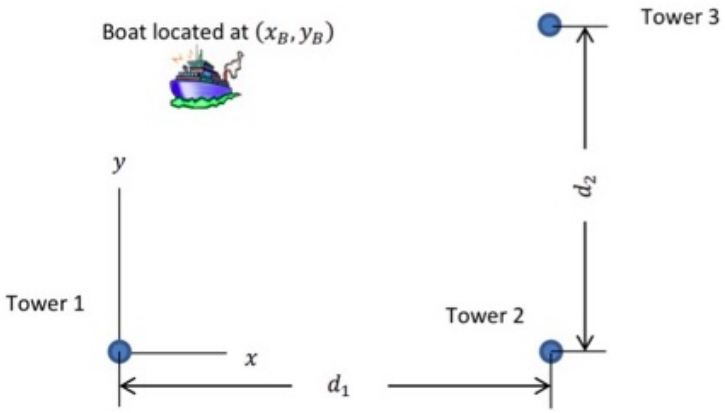
(b) terza prova con $l = 2, m = 3$

4.2 Esempio del sistema LORAN

Il **LORAN** (LOng RAnge Navigation, ovvero navigazione a lungo raggio) è un sistema di radionavigazione terrestre basato su onde radio LF (a bassa frequenza) che sfrutta l'intervallo di tempo tra i segnali ricevuti da tre o più stazioni per determinare la posizione di una nave o di un aereo.
Possiamo considerare il LORAN un antenato del GPS, sistema di posizionamento

satellitare più tecnologicamente avanzato e preciso.
Il sistema LORAN fu sviluppato dal Dipartimento della Difesa degli Stati Uniti a partire dal sistema britannico GEE, usato durante la Seconda guerra mondiale. L'uso di sistemi di radionavigazione andava a sostituire la navigazione astronomica in ambito aeronautico, rivoluzionando così i sistemi di determinazione della posizione di aerei e navi.
Ora che abbiamo capito per cosa viene utilizzato, ci chiediamo quale sia il suo principio di funzionamento: il sistema LORAN si basa sull'intervallo di tempo che intercorre tra segnali ricevuti da una coppia di radiotrasmettitori terrestri sincronizzati. L'intervallo di tempo è direttamente proporzionale alla differenza delle distanze dai due trasmettitori e definisce quindi una linea di posizione iperbolica, detta linea TD (time delay), i cui fuochi sono occupati dalle due stazioni che emettono il segnale. Se si conoscono le posizioni delle stazioni, la posizione del ricevente deve appartenere all'iperbole corrispondente all'intervallo di tempo misurato. Per questo motivo, una sola coppia di stazioni non permette di determinare la posizione, che potrebbe essere uno degli infiniti punti dell'iperbole. È necessario, quindi, che lo stesso principio sia applicato anche su un'altra coppia di trasmettitori, che individuerà una seconda iperbole: la posizione cercata si troverà all'intersezione delle due distinte iperboli. Nella pratica, il sistema LORAN si basa su tre stazioni, accoppiandone una per due volte: una stazione, detta master, è usata per entrambe le applicazioni del principio ed è confrontata separatamente con ognuna delle due stazioni secondarie.

Forniamo ora un esempio di risoluzione di sistema LORAN tramite il metodo iterativo di Newton-Raphson.



Vogliamo stabilire la posizione della nave (x_B, y_B), e per farlo dobbiamo risolvere queste due equazioni, che rappresentano iperboli, definite secondo il principio di funzionamento precedentemente spiegato.

Il sistema di equazioni è il seguente (da risolvere con Newton-Raphson in Matlab):

$$\begin{cases} \frac{4\left(x_B-\frac{d_1}{2}\right)^2}{c^2(t_2-t_1)^2} - \frac{4y_B^2}{d_1^2-c^2(t_2-t_1)^2} = 1 \\ \frac{4\left(y_B-\frac{d_2}{2}\right)^2}{c^2(t_3-t_2)^2} - \frac{4(x_B-d_1)^2}{d_2^2-c^2(t_3-t_2)^2} = 1 \end{cases}$$

dove c è la velocità della luce, d_1, d_2 sono le distanze tra le stazioni e t_1, t_2, t_3 sono sono i tempi in cui arrivano i segnali alla nave dai 3 trasmettitori fissi. Dallo studio del sistema si vede che il tempo t_2 è comune a entrambe le equazioni: questo ci dice che la stazione Master è quella indicata come Torre 2.
Diamo ora dei valori numerici a queste grandezze: $d_1 = 3km, d_2 = 4km$ e $t_1 = 0\mu s, t_2 = 5.72\mu s, t_3 = 8.58\mu s$.
Riportiamo per completezza la function newtonmultiplo utilizzata per risolvere casi studio in cui è necessario utilizzare lo schema di Newton per sistemi di funzioni a più variabili e successivamente creiamo uno script in Matlab per trovare la soluzione (x_B, y_B) in cui richiamiamo la precedente function.

```
function[xnew,iter,vettscarti]=newtonmultiplo(f1,f2,df1dx,df1dy,df2dx,df2dy,
x0,toll,itmax)
iter=0; % inizializzo la variabile di conteggio delle iterazioni
scarto=2*toll; % valore fittizio per entrare nel ciclo while
vettscarti=zeros(itmax,1); % preallocazione vettore scarti
xold=x0;
% entro nel ciclo while per svolgere le iterazioni necessarie
while scarto>=toll && iter<=itmax
    iter=iter+1;
    F=[f1(xold(1),xold(2)); f2(xold(1),xold(2))];
    J=[df1dx(xold(1),xold(2)) df1dy(xold(1),xold(2));
        df2dx(xold(1),xold(2)) df2dy(xold(1),xold(2))];
    y=-J\F;
    xnew=xold+y;
    scarto=norm(xnew-xold);
    vettscarti(iter)=scarto;
    xold=xnew; % aggiorno la variabile
end % fine del ciclo while
vettscarti=vettscarti(1:iter); % taglio il vettore scarti
end % fine della function
```

Di seguito quindi lo script in cui viene richiamata la function:

```
clear
close all
% scrivo le due funzioni iperboliche f1(xB,yB) e f2(xB,yB)
% trovo attraverso Newton la loro intersezione (xB,yB)
% che è la posizione in cui si trova la nave
c=299792458; % velocità della luce in m/s
```

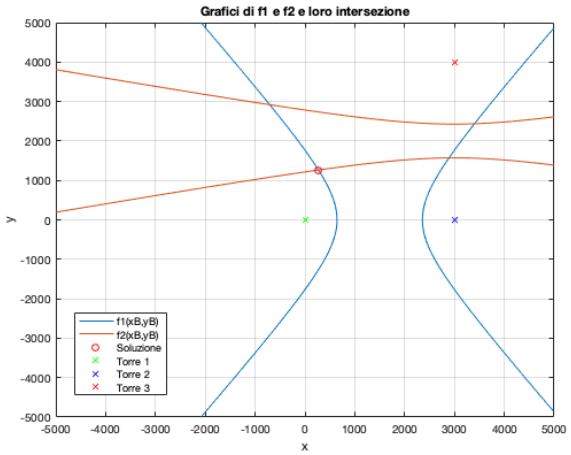
```
d1=3000; % distanza torre1-torre2 in m
d2=4000; % distanza torre2-torre3 in m
t1=0; t2=5.72*10^(-6); t3=8.58*10^(-6);
f1=@(x,y) (4*(x-d1/2).^2)./(c^2*(t2-t1)^2)-(4*y.^2)./(d1^2-c^2*(t2-t1)^2)-1;
f2=@(x,y) (4*(y-d2/2).^2)./(c^2*(t3-t2)^2)-(4*(x-d1).^2)./(d2^2-c^2*(t3-t2)^2)-1;
df1dx = @(x,y) (8*x-4*d1)./(c^2*(t2-t1)^2);
df1dy = @(x,y) -((8*y)./(d1^2-c^2*(t2-t1)^2));
df2dx = @(x,y) -((8*x-8*d1)./(d2^2-c^2*(t3-t2)^2));
df2dy = @(x,y) (8*y-4*d2)./(c^2*(t3-t2)^2);
x0=[0;0]; % vettore x0 di approssimazione iniziale
toll = 1e-6;
itmax=100;
% richiamo la function newtonmultiplo
[xnew,iter,vettscarti]=newtonmultiplo(f1,f2,df1dx,df1dy,df2dx,df2dy,
x0,toll,itmax);
% stampo le soluzioni sulla Command Window
fprintf(1,'Soluzione approssimata: xB= %f \n',xnew(1));
fprintf(1,'Soluzione approssimata: yB= %f \n',xnew(2));
fprintf(1,'Numero di iterazioni effettuate: %i \n', iter);
fprintf(1,'vettore scarti: %12.8e \n',vettscarti);
% faccio il grafico delle due iperboli e della loro intersezione
fimplicit(f1, [-5000, 5000, -5000, 5000], 'DisplayName', 'f1(xB,yB)');
hold on;
fimplicit(f2, [-5000, 5000, -5000, 5000], 'DisplayName', 'f2(xB,yB)');
hold on
plot(xnew(1),xnew(2), 'ro', 'DisplayName', 'Soluzione');
% disegno anche le tre torri con delle x per evidenziarle
% le tre torri sono i fuochi delle nostre iperboli
hold on
plot(0, 0, 'x','Color','green','DisplayName','Torre 1');
plot(3000,0,'x','Color','blue','DisplayName','Torre 2');
plot(3000,4000,'x','Color','red','DisplayName','Torre 3');
legend;
grid on;
xlabel('x');
ylabel('y');
title('Grafici di f1 e f2 e loro intersezione');
% definisco il vettore delle iterazioni
vettiter=1:iter;
% apro un'altra figura per il grafico della convergenza
figure
semilogy(vettiter,vettscarti)
title('grafico di convergenza')
```

Vediamo i risultati ottenuti nella Command Window eseguendo il codice precedente:

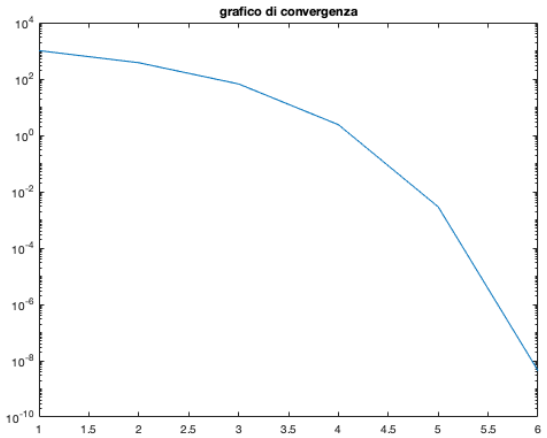
```
>> LORAN
Soluzione approssimata: xB= 271.215018
Soluzione approssimata: yB= 1263.525962
Numero di iterazioni effettuate: 6
```

```
vettore scarti: 1.01646613e+03
vettore scarti: 3.78216248e+02
vettore scarti: 6.67345178e+01
vettore scarti: 2.38834356e+00
vettore scarti: 2.88666297e-03
vettore scarti: 4.42687224e-09
```

Osserviamo ora i grafici che riportano la convergenza e la soluzione:



(a) grafici delle due iperboli e della loro intersezione



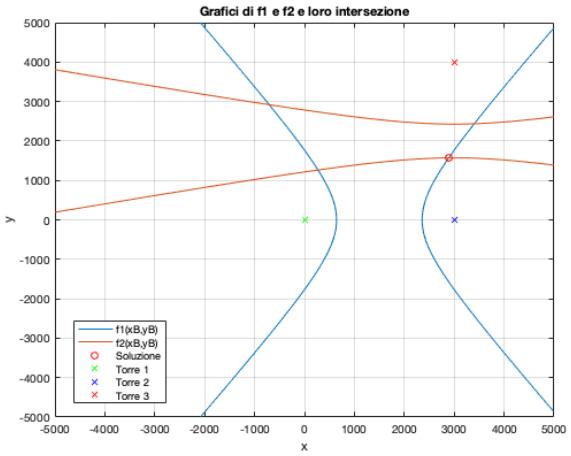
(b) Convergenza quadratica Newton-Raphson

Quindi, attraverso il metodo iterativo di Newton-Raphson siamo riusciti a trovare la posizione della nave: $(x_B, y_B) = (271, 1263)$, che nel grafico è ben rappresentata come l'intersezione delle due iperboli.
Inoltre, possiamo osservare che la convergenza è come al solito quadratica.

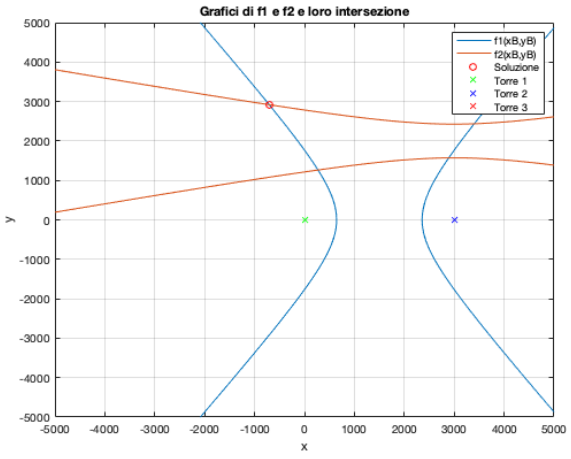
Come possiamo vedere, però, abbiamo altre tre possibili soluzioni, in base alle intersezioni tra i rami delle due iperboli; questo dipende dal valore di approssimazione iniziale del vettore $\boldsymbol{x_0}$, che possiamo scegliere vicino alla soluzione che vogliamo approssimare, come mostrato qui di seguito.

Imponendo nello script di Matlab $\boldsymbol{x_0} = [3000; 1000]$ otteniamo:

```
>> LORAN
Soluzione approssimata: xB=2890.06
Soluzione approssimata: yB=1570.61
Numero di iterazioni effettuate: 6
vettore scarti: 4.59496526e+02
vettore scarti: 1.65964568e+02
vettore scarti: 2.48844333e+01
vettore scarti: 6.24933476e-01
vettore scarti: 3.85318458e-04
vettore scarti: 1.47792889e-10
```

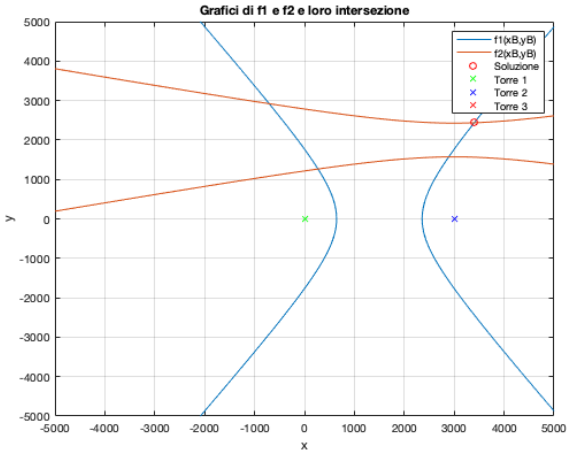


Assegnando poi il seguente valore al vettore $\boldsymbol{x_0} = [-1000; 4000]$ otteniamo:




```
>> LORAN
Soluzione approssimata:xB= -707.21
Soluzione approssimata:yB= 2919.59
Numero di iterazioni effettuate: 6
vettore scarti: 7.90994140e+02
vettore scarti: 2.88903197e+02
vettore scarti: 4.50861364e+01
vettore scarti: 1.09455178e+00
vettore scarti: 6.32597466e-04
vettore scarti: 2.09331028e-10
```

Infine, se $\boldsymbol{x_0} = [4000; 4000]$ otteniamo:




```
>> LORAN
Soluzione approssimata:xB= 3402.35
Soluzione approssimata:yB= 2437.70
Numero di iterazioni effettuate: 7
vettore scarti: 9.91327070e+02
vettore scarti: 4.81417744e+02
vettore scarti: 1.75878906e+02
vettore scarti: 2.97183277e+01
vettore scarti: 8.81765568e-01
vettore scarti: 7.73512401e-04
vettore scarti: 5.94285371e-10
```

Ma come facciamo, quindi, a capire in quale delle quattro possibili intersezioni (formate dai rami delle iperboli) si trova realmente la nave?
Infatti, dal punto di vista fisico solo una di queste soluzioni ha senso e scriviamo qui di seguito il ragionamento che porta alla soluzione corretta.
Nel nostro esempio, la torre 2 (T2) è quella master, mentre le altre due (T1 e T3) sono le secondarie. Anche graficamente si vede, infatti, che T1 e T2 sono i fuochi dell'iperbole segnata in blu, mentre T2 e T3 sono i fuochi dell'altra iperbole, segnata in rosso. La torre T2 è comune ad entrambe le iperboli.

Ora, se l'ordine di arrivo nelle due torri T1 e T2 dei tempi t_1 e t_2 è tale che $t_2 - t_1 > 0$ allora significa che il segnale viene ricevuto prima da T1 e quindi il ramo di iperbole che dobbiamo prendere è quello più vicino alla torre T1, altrimenti, se $t_2 - t_1 < 0$, dobbiamo prendere l'altro ramo dell'iperbole. Lo stesso procedimento si ripete per le due torri T2 e T3, quindi, se $t_3 - t_2 > 0$, allora consideriamo il ramo di iperbole più vicino a T2. Dal momento che $t_2 - t_1 > 0$, dobbiamo prendere il ramo

di iperbole (sul grafico, in colore blu), più vicino a T1. Inoltre $t_3 - t_2 > 0$, quindi dobbiamo prendere il ramo di iperbole (di colore rosso) più vicino alla stazione T2. La soluzione corretta sarà data dunque data da $[271, 1263]$, che è la prima soluzione mostrata.

Capitolo 5

Conclusioni

In sintesi, abbiamo trattato la risoluzione di funzioni non lineari tramite gli schemi iterativi di Punto Fisso e di Newton-Raphson. Siamo dunque partiti dagli schemi semplici per risolvere singole funzioni a una incognita per poi andare a generalizzarli con sistemi di funzioni in più variabili. Abbiamo discusso una breve e per lo più introduttiva trattazione teorica di questi schemi e poi li abbiamo implementati in Matlab, svolgendo degli esempi numerici e mettendo a paragone le diverse velocità di convergenza. Infine, abbiamo visto, tramite esempi di applicazioni pratiche, quali un braccio robotico e il sistema LORAN, come problemi di vita quotidiana possano essere schematizzati tramite modelli matematici che vengono poi risolti attraverso questi metodi iterativi. Abbiamo così dimostrato che ciò che potrebbe sembrare astratto a prima vista può, in realtà, rivelarsi molto più pratico di quanto immaginiamo.

Bibliografia

[1] Antonio J. Conejo; Luis Baringo. *Power System Operations*, chapter appendix A. Springer Cham, 04 September 2018.

[2] Richard L. Burden; J. Douglas Faires; Annette M. Burden. *Numerical Analysis*, chapter 10. Cengage Learning, 10th edition edition, 2015.

[3] Chegg.com. Solved newton's method: Loran navigation. esempio LORAN.

[4] Annamaria Mazzia. *Programmare in Matlab*. libreriauniversitaria.it, 2019.

[5] Annamaria Mazzia. Lezioni di calcolo numerico. Dispense, 2020-21. Capitolo 5, Dipartimento di Ingegneria Civile, Edile e Ambientale Università degli Studi di Padova.

[6] Daene C. McKinney. Numerical methods for civil engineers. Lecture Notes, 2009. Introduction to Computer Methods Department of Civil, Architectural and Environmental Engineering The University of Texas at Austin.

[7] Junichi Takagi; Hirotaka Kanazawa; Kotaro Ichikawa; Hiromici Mitamura. *A simple intuitive method for seeking intersections of hyperbolas for acoustic positioning biotelemetry*, page 4. Kyoto University School of Platforms,Kyoto,Japan, 2022. research article.

[8] Todd Young; Martin J. Mohlenkamp. *Introduction to Numerical Methods and Matlab Programming for Engineers*, chapter 13. Department of Mathematics Ohio University, 2004.

[9] Peter J. Olver. *Mathematical Methods in Computer Vision*, chapter 18, pages 815–816. Springer-Verlag New York Inc., 22 Oct. 2003.

[10] Wikipedia. Loran — wikipedia, l'enciclopedia libera, 2023. Online; in data 15-novembre-2023.