Risoluzione dei sistemi lineari e non lineari mediante metodi iterativi

Luca Murru 33619
Sandra Onnis 36239
Fabrizio Vedovelli 34861

Docente: Giuseppe Rodriguez.

Risoluzione dei sistemi lineari e non lineari mediante metodi iterativi.

INTRODUZIONE.

L'obbiettivo di questo breve testo è quello di illustrare alcuni aspetti applicativi dei metodi iterativi adottati nella risoluzione dei sistemi lineari e non lineari.
Il lavoro è stato suddiviso in due parti generali: la prima è relativa ai sistemi lineari, ed è uno studio comparato, abbastanza generale, di tre metodi in particolare; quello di Jacobi, quello di Gauss Seidel e quello del gradiente coniugato.
I termini di paragone sono stati fondamentalmente due:

1 dimensioni del problema;
2 caratteristiche della matrice dei coefficienti (predominanza diagonale).

Più precisamente si è cercato di valutare gli effetti che una variazione di queste due caratteristiche produceva su:

2 errore sulla soluzione finale;
3 numero di iterazioni impiegate.

La seconda, invece, è la analisi di un metodo in particolare, sempre iterativo, adottabile nella risoluzione di equazioni (e sistemi di equazioni) non lineari.
Il metodo in esame è quello di Newton (conosciuto anche come Newton-Raphson).
Rispetto alla prima parte, però, in questo caso si è fatto riferimento a due problemi più specifici (equazione della circonferenza e equazioni di load flow).
Prima di entrare nel dettaglio si rammenta perché un metodo iterativo dovrebbe essere preferibile a un metodo diretto (nel caso questo esista).
Primo: i metodi iterativi appaiono particolarmente convenienti per matrici di grandi dimensioni, soprattutto se queste sono sparse.
Questo perché i metodi in questione non modificano la matrice del sistema e non appesantiscono la sua memorizzazione.
Secondo: i metodi iterativi possono essere adottati nei casi in cui si voglia raffinare una soluzione approssimata ottenuta con altri algoritmi, o mediante informazioni preliminari sul problema.
Terzo: si riesce a gestire il tempo di elaborazione, legandolo strettamente alla precisione richiesta, quindi alle esigenze del problema in esame.
Cosa comporta, in linea di massima, scegliere questa strada?
I metodi iterativi generano una successione di vettori, a partire da uno iniziale $x^{(0)}$,che, sotto alcune ipotesi, può convergere alla soluzione del problema.
Rispetto ai metodi diretti sono presenti gli errori di troncamento derivanti dal fatto che il limite cercato è necessariamente approssimato.
Questo, inoltre, comporta la necessaria introduzione di un criterio di arresto.

CAPITOLO 1.

1.1.0 METODI ITERATIVI PER LA RISOLUZIONE DI SISTEMI LINEARI: Richiami teorici.

I metodi iterativi per la risoluzione dei sistemi lineari che si sono presi in esame sono tre:

- metodo di Jacobi
- metodo di Gauss Seidel
- metodo del gradiente coniugato.

I primi due si realizzano applicando una particolare strategia che spesso s'incontra nella letteratura relativa ai metodi iterativi, nota con il termine di SPLITTING ADDITIVO.
A partire, quindi, da un generico sistema lineare del tipo

$$A\mathbf{x}=\mathbf{b} \qquad (1)$$

la matrice A viene scomposta come:

$$A=P-N \qquad (2)$$

dove P è la matrice di precondizionamento.
Sostituendo l'espressione di sopra nella (1) si ottiene:

$$P\mathbf{x}-N\mathbf{x}=\mathbf{b}$$

quindi

$$P\mathbf{x}=N\mathbf{x}+\mathbf{b} \qquad (3).$$

Il metodo iterativo ,perciò, eseguirà, ad ogni iterazione :

$$P\mathbf{x}^{(k+1)}=N\mathbf{x}^{(k)}+\mathbf{b} \qquad (4)$$

Un metodo così realizzato è sicuramente un metodo consistente, questo significa che per tutti i metodi costruiti su questa base è possibile evitare la verifica della consistenza.
Questo non significa che lo stesso sia anche convergente (la consistenza non implica la convergenza).

1.1.1 Metodi di Jacobi e Gauss-Seidel.

Prendiamo in esame un particolare splitting additivo basato sulla scomposizione della matrice A come:

$$A=D-E-F \qquad (5)$$

dove le matrici D,E,F contengono, rispettivamente, la diagonale di A, il triangolo inferiore della matrice di partenza cambiato di segno e il triangolo superiore della medesima, sempre cambiato di segno.

Adottare il METODO DI JACOBI significa porre

$$P=D$$

$$N=E+F \qquad (6)$$

Dato che P deve essere non singolare, allora

$$a_{ii}\neq 0 \quad i=1,\ldots\ldots n \qquad (7)$$

La (4) diviene in questo caso particolare

$$D\mathbf{x}^{(k+1)} = \mathbf{b}+E\mathbf{x}^{(k)} +F\mathbf{x}^{(k)} \qquad (8)$$

In questo modo si riesce a ricavare le componenti di $x^{(k+1)}$ a partire da quelle di $x^{(k)}$ in qualsiasi ordine e indipendentemente l'una dall'altra (metodo parallelizzabile).
Per studiare la convergenza si analizza la matrice di iterazione ottenuta come:

$$B_j=P^{-1}N=D^{-1}(E+F) =I-D^{-1} \qquad (9)$$

Si otterrà la convergenza quando

$$\rho(B_j)<1 \qquad (10)$$

Adottare il METODO DI GAUSS SEIDEL significa, invece, porre

$$P=D-E$$
$$N=F \qquad (11)$$

Ricordando come un metodo iterativo lineare, stazionario del primo ordine assume la forma

$$x^{(k+1)} = Bx^{(k)} + f \qquad (12)$$

operando le opportune sostituzioni si ottiene che, nel metodo di Gauss Seidel vale

$$(D - E)\mathbf{x}^{(k+1)} = \mathbf{b} + F\mathbf{x}^{(k)} \qquad (13)$$

Al generico passo si ha che il nuovo vettore soluzione risulta essere

$$x_i^{(k+1)} = \frac{1}{a_{ii}}[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}] \quad \text{con i = 1,.....,n.} \qquad (14)$$

Il metodo di Gauss Seidel non è parallelizzabile dal momento che ogni componente i-esima di $x^{(k+1)}$ dipende dalle componenti con indice compreso tra 1 e i - 1
Il metodo di Gauss Seidel in alcune circostanze è preferito a quello di Jacobi in quanto converge più rapidamente.

1.1.2 Metodo del Gradiente Coniugato.

Un altro tipo di approccio alla risoluzione di un sistema lineare, sempre attraverso un metodo iterativo, è quello di risolvere un problema equivalente a quello originario:

$$A\mathbf{x}=\mathbf{b} \qquad (15)$$

e più precisamente un problema di minimizzazione .
Ovviamente la soluzione del problema di minimo deve essere tale da mantenere inalterata quella del problema iniziale.
Una delle forme più utilizzate per questo genere di approccio è quella tipica dei metodi di discesa :

$$x^{(k+1)}=x^{(k)}+\alpha_k * d^{(k)} \qquad (16)$$

I vari metodi sviluppati in tal senso si ottengono effettuando una scelta per quello che riguarda :
-la direzione $d^{(k)}$
-il passo α_k
Un metodo in particolare è il metodo del gradiente ottenuto scegliendo come direzione del metodo al generico passo pari al residuo al medesimo passo:

$$\mathbf{d}^{(k)}=\mathbf{r}^{(k)} \qquad (17)$$

e valutando l'ampiezza del passo come:

$$\alpha^{(k)}=(\mathbf{r}^{(k)T} * \mathbf{r}^{(k)})/ (\mathbf{r}^{(k)T} * A*\mathbf{r}^{(k)}) \qquad (18)$$

Il metodo del gradiente nella pratica, però, non viene mai usato.
Questo è dovuto al fatto che il metodo in genere converge con un numero di passi eccessivamente elevato, e che ad ogni passo si accumulano, ovviamente, gli errori; a discapito del risultato finale.
L'alternativa potrebbe essere usare il metodo del gradiente precondizionato, oppure passare direttamente al metodo del gradiente coniugato.
Per spiegarlo si consideri una matrice A simmetrica definita positiva e la forma quadratica:

$$\phi(y)=\frac{1}{2}y^T Ay - y^T b \qquad (19)$$

il minimo della (19) lo si ottiene nel punto in cui si annulla il gradiente:

$$\nabla\phi(y)=\frac{1}{2}(A+A^T)y-b=Ay-b=0 \qquad (20)$$

Prima di effettuare una scelta per la direzione di discesa, occorre introdurre il concetto di ottimalità: definizione:

$\mathbf{x}^{(k)}$ è ottimale rispetto a "$\mathbf{p}$" $\Longleftrightarrow$ $\mathbf{p}^T\mathbf{r}^{(k)}=0$ (condizione di ortogonalità)

Poniamo allora

$$x^{(k+1)}=x^{(k)}+q \qquad (21)$$

e affinché $x^{(k+1)}$ sia ottimale rispetto a p, deve risultare:

$$0=p^T r^{(k+1)}=p^T(r^{(k)}-Aq)=-p^T Aq \qquad (22)$$

e questo significa che p e q devono essere A-ortogonali (A-coniugate).Partiamo allora considerando uguali le direzioni di p e r al passo 0, ossia $p^{(0)} = r^{(0)}$ e valutiamo le direzioni successive nel seguente modo:

$$p^{(k+1)} = r^{(k+1)} - \beta_k p^{(k)} \qquad (23)$$

Dove β_k è il passo.
Se $p^{(k)}$ e $p^{(k+1)}$ sono A-ortogonali ossia se è verificata $(p^{(k)})^T A p^{(k+1)} = 0$ allora di conseguenza si ottiene che β_k abbia la seguente forma:

$$\beta_k = \frac{(p^{(k)})^T A r^{(k+1)}}{(p^{(k)})^T A p^{(k)}} \qquad (24)$$

In tal modo le direzioni $p^{(i)}$ sono A-ortogonali tra loro , con i=1, ……..,k.
A causa della A-ortogonalità si trova che la direzione di discesa non può essere ulteriormente ridotta, e di conseguenza la funzione obbiettivo da minimizzare $\phi(y)$ non si può ulteriormente ridurre. Aggiungiamo infine che se A non è simmetrica definita positiva, il metodo del gradiente coniugato non può essere applicato.
Tale condizione di ottimo si ottiene per $p^T r^{(k)} = 0$ (condizione di ortogonalità tra i due vettori).

1.2.0 METODI ITERATIVI PER LA RISOLUZIONE DI SISTEMI LINEARI: Algoritmi.

È possibile implementare sul codice commerciale "Matlab" un algoritmo che, chiamando delle funzioni precedentemente create, consente di risolvere dei sistemi lineari caratterizzati da matrici di dimensione "nxn" utilizzando proprio gli algoritmi appena descritti.

In realtà il programma permette di valutare un set di sistemi lineari di dimensioni via via crescenti : ponendo la variabile nmax=100, per esempio, il programma calcola la risoluzioni per le matrici A di dimensioni
10x10
20x20
.
.
.
100x100
L'algoritmo, inoltre, consente il confronto dei tre metodi in termini di errore e di numero di iterazioni.

ALGORITMO PRINCIPALE

L1: **nmax=50**;	dimensione massima matrice
L2: **k=1;**	inizializzazione contatore
L3: **errJ=[];**	inizializzazione vettore errore sul metodo di Jacobi
L4: **errGS=[];**	inizializzazione vettore errore sul metodo di Gauss-Seidel
L5: **errGC=[];**	inizializzazione vettore errore sul metodo del gradiente coniugato
L6: **niterJ=[];**	inizializzazione vettore che registra il numero di iterazioni sul metodo di Jacobi
L7: **niterGS;**	inizializzazione vettore che registra il numero di iterazioni sul metodo di Gauss-Seidel
L8: **niterGC=[];**	inizializzazione vettore che registra il numero di iterazioni sul metodo del gradiente coniugato
L9: **TJ=[];**	inizializzazione del vettore tempo per il metodo di Jacobi
L10: **TGS=[];**	inizializzazione del vettore tempo per il metodo di Gauss-Seidel
L11: **TGC=[];**	inizializzazione del vettore tempo per il metodo del Gradiente Coniugato.
L12: **dimmatrice=[];**	inizializzazione vettore per il plot di n

L13: **for n=10:10:nmax**

L14: **A=rand(n);** crea una matrice random di dimensione 10x10.

L15: **A=A-diag(diag(A));** elimina la diagonale principale dalla matrice di partenza

L16: **A=A'*A;** crea la condizione di simmetria e di positività per la matrice

L17: **S=A*ones(n,1);** crea un vettore con elementi pari alla somma righe della matrice A.

L18: **A=A+5*diag(S);** crea la condizione di predominanza diagonale

L19: **xgiusta=ones(n,1);** vettore soluzione

L20: **b=A*xgiusta;** crea un vettore di termini noti in modo tale che la soluzione al sistema lineare sia xgiusta

L21: **tau=1e-8;** errore relativo massimo sul calcolo della soluzione del sistema

L22: **Nmax=100;** numero massimo di iterazioni

funzione per la risoluzione del sist. mediante Jacobi

L23: **[niterazioniJ,vettoresoluzioneJ,tJ]=totaleJ(A,b,tau,Nmax);**

funzione per la risoluzione del sist. mediante Gauss-Seidel

L24: **[niterazioniGS,vettoresoluzioneGS,tGS]= totalegs(A,b,tau,Nmax);**

funzione per la risoluzione del sist. mediante Gradiente Coniugato

L25: **[niterazioniGC,vettoresoluzioneGC,tGC]= totaleGC(A,b,tau,Nmax);**

L26: **errJ(k)=norm(xgiusta-vettoresoluzioneJ ,2);** aggiornamento vettore errore (Jacobi)

L27: **errGS(k)=norm(xgiusta-vettoresoluzioneGS ,2);** aggiornamento vettore errore (Gauss Seidel)

L28: **errGC(k)=norm(xgiusta-vettoresoluzioneGC,2);** aggiornamento vettore errore (Gradiente Coniugato)

L29: **niterJ(k)=niterazioniJ;** aggiornamento vettore num. iterazioni (Jacobi)

L30: **niterGS(k)=niterazioniGS;** aggiornamento vettore num. iterazioni (Gauss-Seidel)

L31: **niterGC(k)=niterazioniGC;** aggiornamento vettore num. iterazioni (Gradiente Coniugato)

L32: **dimmatrice(k)=k;** aggiornamento vettore dimensione matrice

L33: **TJ** aggiornamento del vettore tempo (Jacobi)

L34: **TGS** aggiornamento del vettore tempo (Gauss-Seidel)

L35: **TGC** aggiornamento del vettore tempo (gradiente Coniugato)

L36: **k=k+1;** incremento contatore

L37: **end**

plottaggio dell'andamento dell'errore all'aumentare della dimensione della matrice A
L38: **[stampaJ]= plotER(dimmatrice,errJ,errGS,errGC,nmax)**

plottaggio del numero delle iterazioni all'aumentare della dimensione della matrice A
L39: **[stampaITERJ]=plotITERK(dimmatrice,niterJ,niterGS,niterGC,nmax)nmax=50;**

plottaggio del tempo impiegato per risolvere il sist. all'aumentare della dimensione della matrice A
L40: **[stampaITERJ]=plotTEMPO(dimmatrice, TJ, TGS, TGC,nmax)**

In corrispondenza della linee di programma L14-L15-L16-L17-L18 viene creata una matrice A simmetrica, definita positiva simmetrica (L16) e a predominanza diagonale(L18). Queste ipotesi sulla matrice di partenza sono necessarie affinché i metodi possano convergere. In particolare sia i metodi di Jacobi che di Gauss-Seidel necessitano di una matrice a predominanza diagonale, mentre il metodo del Gradiente Coniugato necessita di una matrice simmetrica e definita positiva .
Sono state imposte tutte e le condizioni sulla stessa matrice in modo da poter confrontare i tre metodi.

Il confronto dei metodi viene portato avanti mostrando sia in maniera grafica che analitica l'andamento dell'errore (L38) , il numero di iterazioni (L39) ed il tempo impiegato dai metodi a convergere (L40) in funzione della dimensione della matrice A.
I risultati verranno mostrati nel paragrafo "1.3.0 CONFRONTO SUI RISULTATI RELATIVI AI TRE METODI. "
In corrispondenza della linea L23 viene chiamata una funzione "totaleJ" che presenta 4 argomenti in ingresso (A, b, tau, Nmax) e altri tre in uscita (niterazioniJ, vettoresoluzioneJ ,tJ). Si tratta di una funzione che riceve in ingresso la matrice A ,il vettore dei termini noti **b,** l'errore relativo massimo(tau) ed il numero di iterazioni massimo(Nmax). La funzione "totaleJ" applica il metodo di Jacobi e rende il numero di iterazioni (niterazioniJ), il vettore soluzione (vettoresoluzioneJ) ed il tempo impiegato dal metodo (tJ) per risolvere il sistema lineare.
Di seguito si riporta la funzione implementata.

FUNZIONE PER LA RISOLUZIONE DEL SISTEMA LINEARE MEDIANTE IL METODO DI JACOBI

```
L1:  function[niterazioni,vettoresoluzione,tJ]=totaleJ (A,b,tau,Nmax)

L2:  P=diag(diag(A));          prende la diagonale di A
L3:  E=-triu(A)+P;             prende il triangolo superiore di A e lo cambia di segno
L4:  F=-tril(A)+P;             prende il triangolo inferiore di A e lo cambia di segno
L5:  N=E+F;                    matrice A privata della diagonale e cambiata di segno
L6:  n=size(A,1);              dimensione della matrice A
L7:  x=zeros(n,1);             inizializzazione del vettore soluzione
L8:  k=0;                      inizializzazione di un contatore.
L9:  xv=ones(n,1);             vettore soluzione esatto.
L10:  tic                      inizio contattore temporale
L11:  while (k<Nmax)           criterio d'arresto
   L12:  xv=x;                 assegna il vettore soluzione appena calcolato alla variabile
                               xv
   L13:  x=P\(N*xv+b)          calcolo effettivo della soluzione
   L14:  k=k+1;                incremento contatore
   L15  if(norm(x-xv,2)<tau*norm(x,2)),break,end   uscita dal ciclo se l'errore relativo è
                                                   troppo piccolo.
L16:  end
L17:  tJ=toc;                  mostra il tempo impiegato per il calcolo
L18:  niterazioni=k;           numero delle iterazioni
L19:  vettoresoluzione=x;      soluzione calcolata col metodo di Jacobi.
```

I primi passi che l'algoritmo esegue sono quelli relativi alla creazione delle matrici necessarie ad effettuare lo splitting additivo (linee L2-L3-L4-L5).
In secondo luogo si ha il calcolo della dimensione della matrice A (L6) in quanto servirà al programma per costruire il vettore soluzione di partenza delle dimensioni opportune (L7).
All'interno del ciclo "while" vengono eseguite le operazioni descritte nella parte teorica relativamente al metodo di Jacobi con la particolare accortezza di evitare l'inversione di matrice, ma introducendo nella linea L13 una divisione a sinistra.
Ogni volta che viene eseguito il ciclo il metodo restituisce un vettore soluzione che si avvicina sempre più alla soluzione esatta fino a quando non vi esce o perché il numero delle iterazioni è troppo elevato (superiore alla soglia "Nmax" imposta dall'utente) o perché lo scostamento dalla soluzione esatta è sufficientemente basso (inferiore al parametro "tau" stabilito dall'utente).
È necessario poter avere in uscita, oltre al vettore soluzione, anche il numero delle iterazioni eseguite dall'algoritmo proprio per rendersi conto se questo è arrivato a convergenza o meno.

Se l'algoritmo si dovesse bloccare con un numero di iterazioni pari a "Nmax", allora l'utente dovrà valutare la correttezza della soluzione con altri metodi in quanto questo algoritmo potrebbe non essere arrivato a convergenza.

Anche l'algoritmo relativo all'implementazione del metodo di Gauss Seidel è realizzabile sulla stessa falsariga del metodo di Jacobi. Di seguito si illustra la funzione per la risoluzione dei sistemi lineari attraverso tale metodo. Le considerazioni portate avanti per il metodo di Jacobi sono estendibili al metodo di Gauss-Seidel.

FUNZIONE PER LA RISOLUZIONE DEL SISTEMA LINEARE MEDIANTE IL METODO DI GAUSS-SEIDEL

```
L1:  function[niterazioni,vettoresoluzione,tGS]=totalegs(A,b,tau,Nmax)

L2:  P=diag(diag(A));          prende la diagonale di A
L3:  E=-triu(A)+P;             prende il triangolo superiore di A e lo cambia di segno
L4:  F=-tril(A)+P;             prende il triangolo inferiore di A e lo cambia di segno
L5:  M=P-E;                    matrice A privata del triangolo superiore cambiata di
                               segno
L6:  n=size(A,1);              dimensione della matrice A
L7:  x=zeros(n,1);             inizializzazione del vettore soluzione
L8:  k=0;                      inizializzazione di un contatore.
L9:  xv=ones(n,1);             vettore soluzione esatto.
L10:  tic                      inizio contattore temporale
L11:  while (k<Nmax)           criterio d'arresto
   L12:  xv=x;                 assegna il vettore soluzione appena calcolato alla variabile
                               xv
   L13:  x=M\(F*xv+b)          calcolo effettivo della soluzione
   L14:  k=k+1;                incremento contatore
   L15:  if(norm(x-xv,2)<tau*norm(x,2)),break,end   uscita dal ciclo se l'errore relativo è
                                                    troppo piccolo.
  L16:  end
L17:  tGS=toc;                 mostra il tempo impiegato per il calcolo
L18:  niterazioni=k;           numero delle iterazioni
L19:  vettoresoluzione=x;      soluzione calcolata col metodo di Gauss-Seidel.
```

Per la risoluzione dei sistemi lineari è possibile servirsi anche del metodo del gradiente coniugato. Di seguito se ne riporta la funzione che consente la sua implementazione.

FUNZIONE PER LA RISOLUZIONE DEL SISTEMA LINEARE MEDIANTE IL METODO DEL GRADIENTE CONIUGATO

```
L1:  function[niterazioniGC,vettoresoluzioneGC,tGC]=totalegs(A,b,tau,Nmax)

L2:  n=size(A,1);            determina la dimensione di A;
L3:  x=zeros(n,1);           inizializzazione del vettore soluzione;
L4:  r=b-A*x;                determina il residuo iniziale;
L5:  p=r;                    assegna a p lo stesso significato di r;
L6:  k=0;                    inizializzazione contatore;
L7:  tic;                    inizio contatore temporale;
L8:  while (k<Nmax)          criterio d'arresto;
   L9:  s=A*p;               termine necessario alla determinazione del passo;
   L10: sigma=p'*s;          termine necessario alla determinazione del passo;
   L11: alfa=(p'*r)/sigma;   determinazione del passo;
   L12: xv=x;                assegna il vettore soluzione appena calcolato alla
                             variabile;
   L13: x=x+alfa*p;          calcolo della soluzione;
   L14: r=r-alfa*s;          aggiornamento del residuo;
   L15: beta=(s'*r)=/sigma;  calcolo del parametro beta per cui tutte le direzioni sono
                             tra loro A coniugate;
   L16: p=r-beta*p;          aggiornamento della direzione;
   L17: k=k+1;               incremento contatore
   L18: if(norm(x-xv,2)<tau*norm(x,2)),break,end   uscita dal ciclo se l'errore relativo è
                                                   troppo piccolo.
L19: end

L20: tGC=toc;                mostra il tempo impiegato per il calcolo
L21: niterazioniGC=k;        numero delle iterazioni
L22: vettoresoluzioneGC=x;   soluzione calcolata col metodo del gradiente coniugato.
```

1.3.0 CONFRONTO SUI RISULTATI RELATIVI AI TRE METODI.

Il confronto delle prestazioni dei tre metodi viene portato avanti in funzione della variazione delle dimensioni della matrice con un fattore di predominanza diagonale fissato a 5 (vedi L18 di "Algoritmo principale").
La matrice è stata realizzata attraverso il comando di mathlab "*rand*" e la dimensione della stessa èstata fatta variare attraverso un ciclo *for* da 100x100 a 2000x2000.
Per ciascun metodo si prenderà in analisi

- l'errore,
- il numero di iterazioni,
- il tempo di calcolo necessario alla risoluzione del sistema.

L'errore relativo massimo impostato per l'arresto del sistema è "tau = $1*10^{-8}$" .

Analizziamo i risultati relativi all'errore compiuto dai diversi metodi al variare della dimensione della matrice.

Dimensione matrice n (Fattore pred. diag. 5)	Jacobi	Gauss-Seidel	Gradiente coniugato
100	$6.6378*10^{-9}$	$1.2602*10^{-10}$	$2.17*10^{-9}$
500	$1.7569*10^{-8}$	$3.0858*10^{-10}$	$1.5333*10^{-9}$
1000	$2.5371*10^{-8}$	$4.4217*10^{-10}$	$4.6798*10^{-10}$
1500	$3.1290*10^{-8}$	$5.4398*10^{-10}$	$5.0951*10^{-9}$
2000	$3.6256*10^{-8}$	$6.2927*10^{-10}$	$4.0520*10^{-9}$

Tabella 1. Errore in funzione della dimensione della matrice.

la tabella mostra i risultati dell'operazione || vettroresoluzione – xgiusta ||. "Vettoresoluzione" non è altro che la soluzione calcolata attraverso i metodi di Jacobi, Gauss-Seidel, e del Gradiente Coniugato, mentre " xgiusta" è la soluzione esatta dei sistemi.
Dall'ordine di grandezza degli errori, ci si accorge come, in tutti i casi mostrati in tabella ,il codice permetta l'uscita dal ciclo "while" interno alle funzioni " totaleJ" , "totaleGS" e "totaleGC" proprio perché tali errori risultano minori della tolleranza "tau" prestabilita.
Questo è un buon risultato in quanto in tutti i casi analizzati, i tre metodi stavano convergendo alla soluzione esatta.
L'errore associato al metodo di Gauss-Seidel risulta, comunque, inferiore rispetto a quello associabile agli altri due metodi.

È possibile vedere meglio tale risultato nel seguente grafico che riporta, al variare della dimensione della matrice da n=10 a n=2000, con step pari a 10, proprio l'errore riportato dai tre metodi.

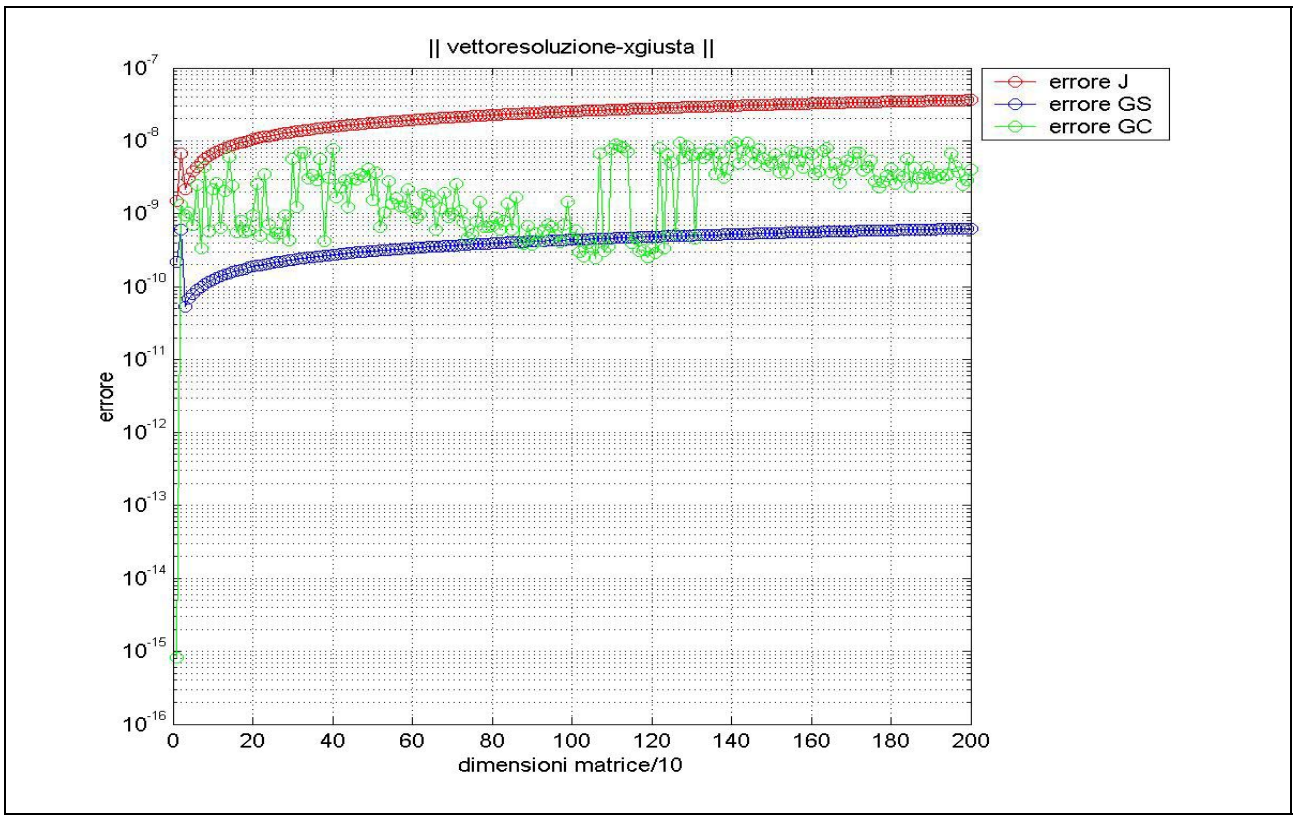


Figura 1. Andamento dell'errore al variare della dimensione della matrice.

I metodi di Jacobi e Gauss Seidel sono praticamente caratterizzati da un andamento asintotico rispettivamente pari a $3{,}5*10^{-8}$ e $6*10^{-10}$, mentre per il metodo del Gradiente Coniugato l'andamento risulta piuttosto irregolare, pare stabilizzarsi per n=1400 intorno al valore $5*10^{-9}$.

Un'altra analisi può essere condotta sul numero di iterazioni impiegato da ciascun metodo per convergere alla soluzione.

	Numero iterazioni Jacobi	Numero iterazioni Gauss-Seidel	Numero iterazioni Gradiente Coniugato
100	13	8	9
500	13	8	7
1000	13	8	7
1500	13	8	6
2000	13	8	6

Tabella 2. Numero di iterazioni al variare della dimensione della matrice

Risulta evidente dai risultati ottenuti che il metodo di Jacobi, anche dal punto di vista del numero di iterazioni è il peggiore.

Risulta più esplicativo mostrare il seguente grafico dove è possibile vedere meglio come, dal punto di vista del numero di iterazioni, il metodo del gradiente coniugato sia quello che si comporti meglio al crescere della dimensione della matrice A.

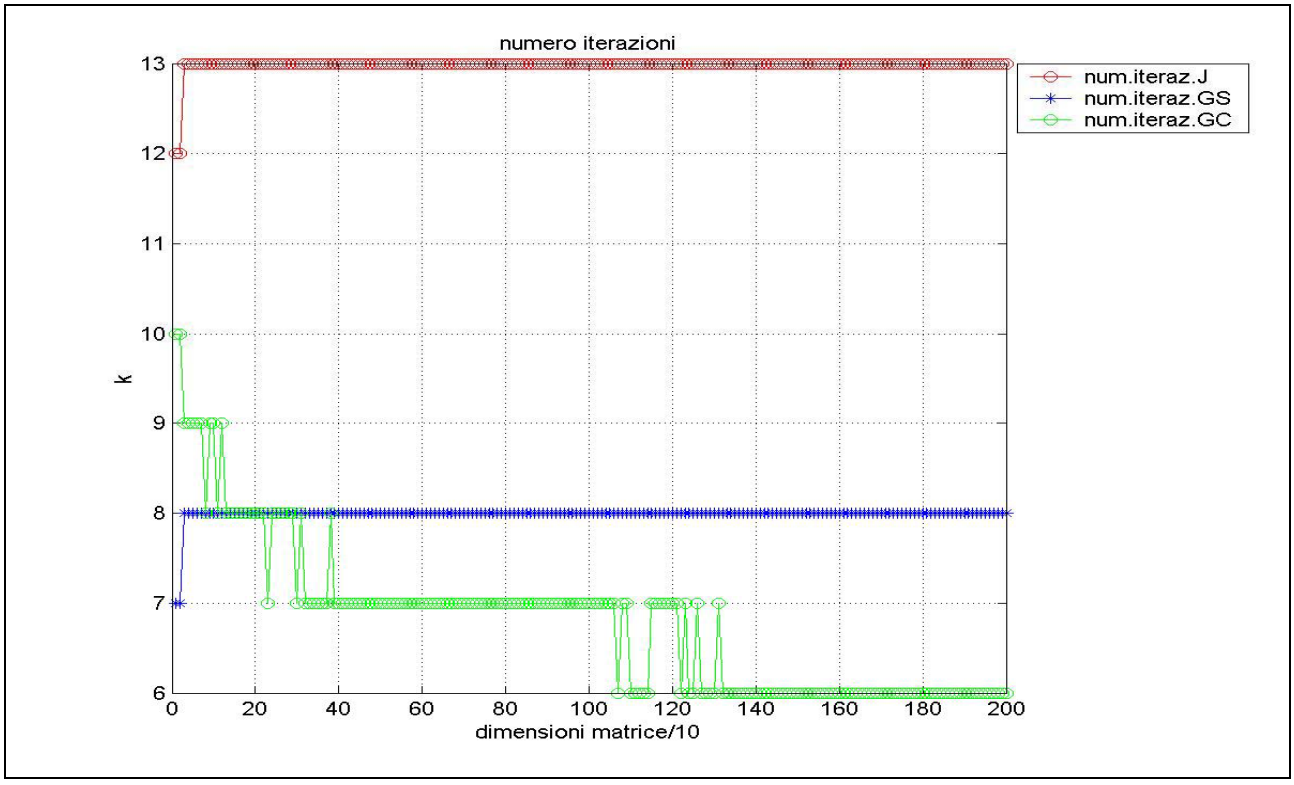


Figura 2. Numero di iterazioni al variare della dimensione della matrice.

Si mostra, infine, la crescita del tempo necessario ad effettuare il calcolo per mezzo dei tre metodi e, come ci si aspettava , vista la complessità computazionale dei tre metodi, quello del Gradiente coniugato, si riconferma il più veloce.

	Tempo Jacobi	Tempo Gauss-Seidel	Tempo Gradiente Coniugato
100	0.0100	0.0100	0
500	0.0900	0.0600	0.0100
1000	0.3800	0.2300	0.0600
1500	0.8320	0.5410	0.1100
2000	1.5720	1.0820	0.2400

Tabella 3. Andamento dell'tempo in funzione della dimensione della matrice.

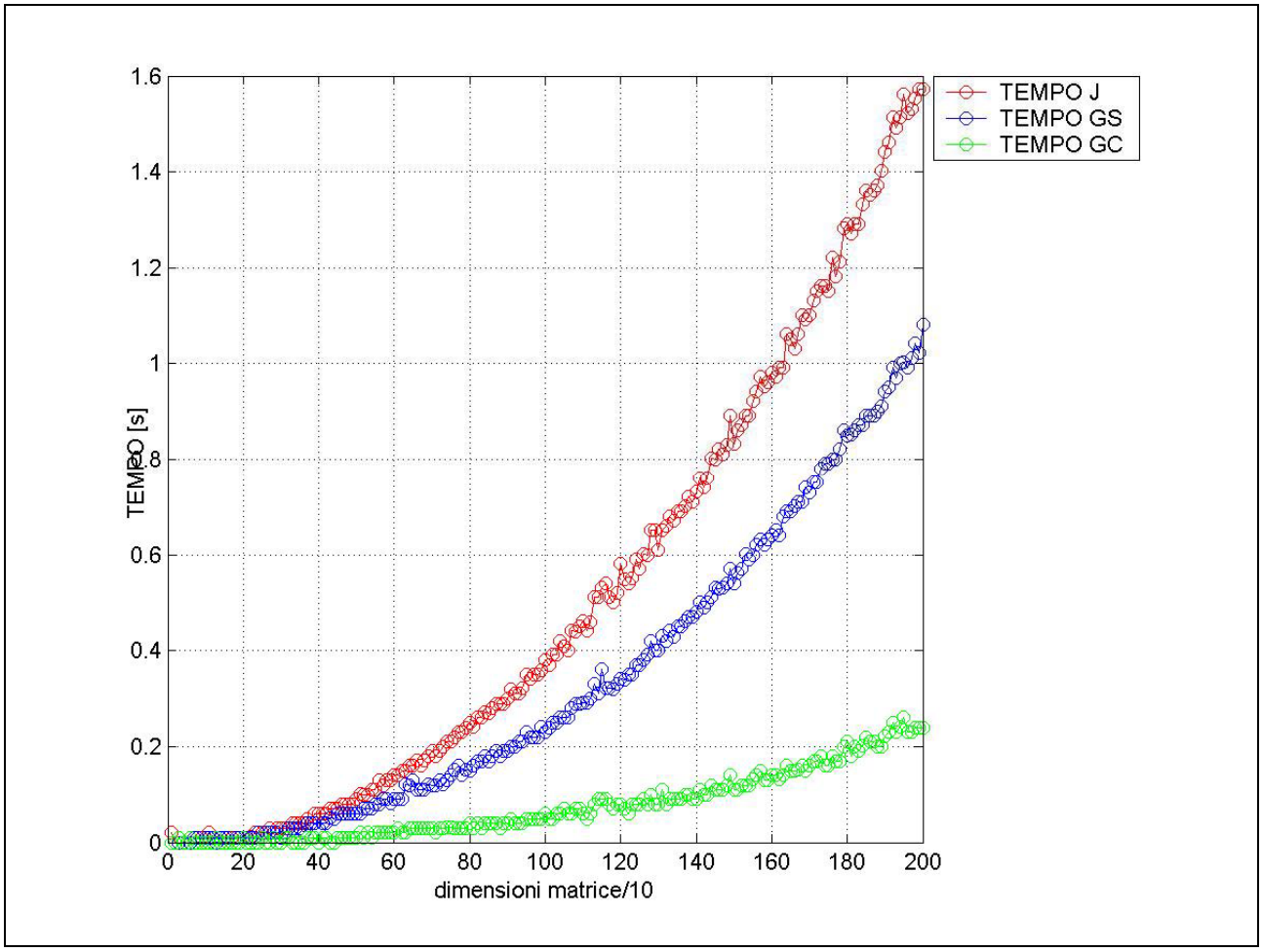


Figura 3. Andamento del tempo in funzione della dimensione della matrice.

1.3.1. Studio sulla predominanza diagonale.

Il seguente algoritmo vuole mostrare il comportamento dei tre metodi al decadere della condizione relativa alla predominanza diagonale della matrice A.

STUDIO SULLA PREDOMINANZA DIAGONALE

L1: **n=100;** dimensioni matrice;
L2: **g=1;** inizializzazione contatore;
L3: **errJ=[];** inizializzazione vettore errore (jacobi);
L4: **errGS=[];** inizializzazione vettore errore (Gauss-Seidel);
L5: **errGC=[];** inizializzazione vettore errore (Gradiente Coniugato);
L6: **A=rand(n);** creazione di una matrice A random;
L7: **A0=A-diag(diag(A));** matrice A senza diagonale;
L8: **A0=A0'*A0;** condizione di simmetria e di matrice definita positiva;
L9: **S=abs(A0)*ones(n,1);** vettore somma elementi riga di A0;

L10: **nJ = [];** inizializzazione vettore numero di iterazioni (Jacobi);
L11: **nGS = [];** inizializzazione vettore numero di iterazioni (Gauss-Seidel);
L12: **nGC = [];** inizializzazione vettore numero di iterazioni (Gradiente Coniugato);
L13: **sv=[5 2 1 .9 .8 .7 .6 .5 .4 .3 .2 .1];** vettore di fattori moltiplicativi della matrice

L14: **for s=sv**
  L15: **A=A0+s*diag(S);** costruzione della matrice;
  L16: **xgiusta=ones(n,1);** soluzione esatta;
  L17: **b=A*xgiusta;** ricerca dei termini noti del sistema;
  L18: **tau=1e-8;** errore massimo di calcolo;
  L19: **Nmax=100;** numero di iterazioni massimo;

risoluzione del sistema mediante il metodo di Jacobi
L20: **[niterazioniJ,vettoresoluzioneJ,tJ]= totaleJ(A,b,tau,Nmax);**

risoluzione del sistema mediante il metodo di Gauss-Seidel
L21: **[niterazioniGS,vettoresoluzioneGS,tGS]= totalegs(A,b,tau,Nmax);**

risoluzione del sistema mediante il metodo del Gradiente Coniugato
L22: **[niterazioniGC,vettoresoluzioneGC,tGC]= totaleGC(A,b,tau,Nmax);**

L23: **errJ(g)=norm(xgiusta-vettoresoluzioneJ ,2);** aggiornamento vettore errore (Jacobi);

L24: **errGS(g)=norm(xgiusta-vettoresoluzioneGS ,2);** _____aggiornamento vettore errore (Gauss-Seidel);
L25: **errGC(g)=norm(xgiusta-vettoresoluzioneGC ,2);** _____aggiornamento vettore errore (Gradiente Coniugato);
L26: **nJ = [nJ;niterazioniJ];**_____ aggiornamento vettore numero di iterazioni (Jacobi);
L27: **nGS = [nGS;niterazioniGS];** _____ aggiornamento vettore numero di iterazioni (Gauss-Seidel);
L28: **nGC=[nGC;niterazioniGC];**_____ aggiornamento vettore numero di iterazioni (Gradiente coniugato);
L29: **g=g+1;** _____ incremento contatore;
L30: **end**

plotaggio numero di iterazioni
L31: **[stampaiter]=iterazioni(sv,nJ,nGS,nGC);**

plotaggio numero di iterazioni
L32: **[stampaerr]=errore(sv,errJ,errGS,errGC);**

Il codice sopra descritto risolve un sistema di dimensioni stabilite dall'utente per mezzo dei metodi di Jacobi, Gauss-Seidel e del Gradiente coniugato al variare delle condizioni di predominanza diagonale della matrice di partenza.

Nella linea L13 viene definito un vettore di valori decrescenti; ciascun valore diverrà di volta in volta fattore moltiplicativo della diagonale della matrice di partenza riducendone e poi annullandone le condizioni per la dominanza diagonale (L15).
Il risultato che si ottiene è quello mostrato dal seguente grafico in uscita alla funzione "iterazioni" (L31).

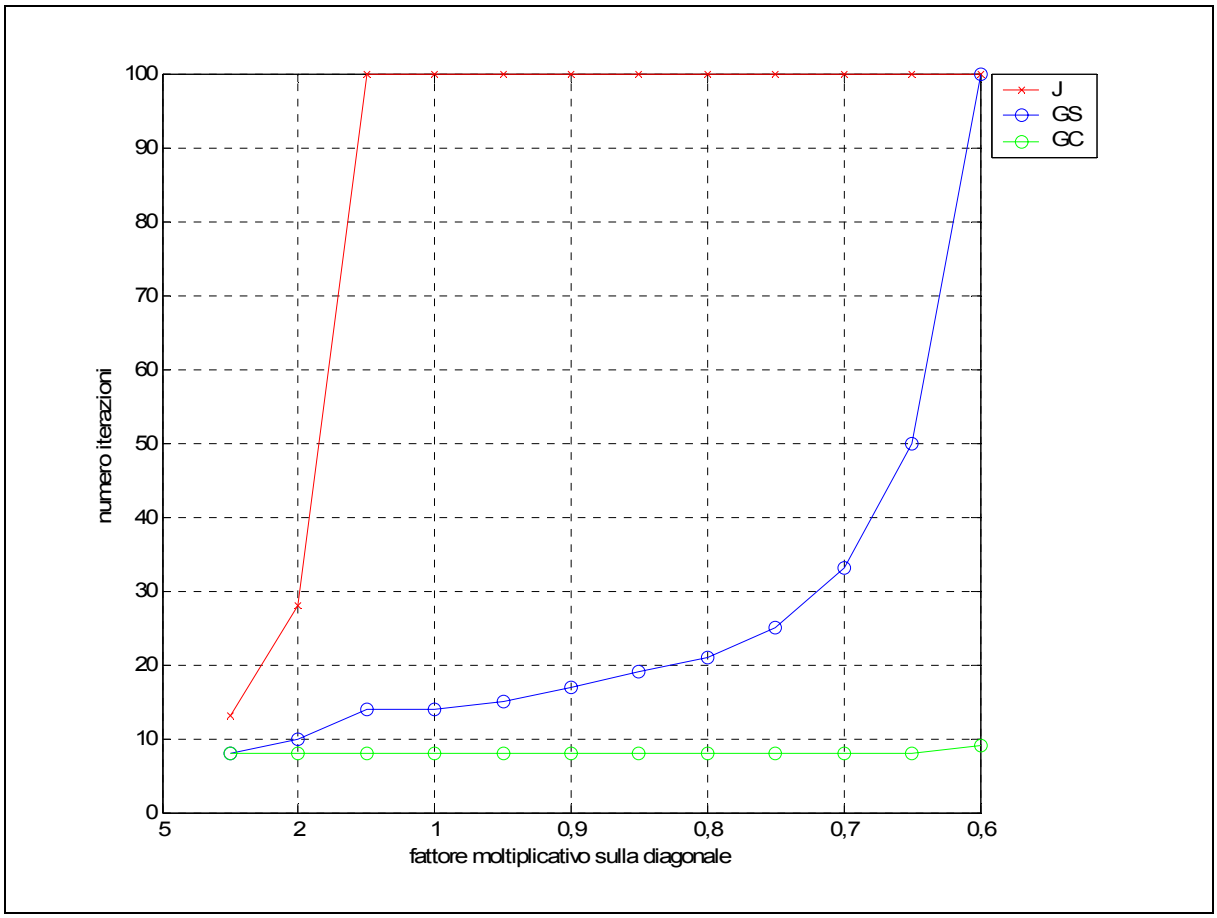


Figura 4. Numero di iterazioni.

nJ	nGS	nGC
13	8	8
28	10	8
100	14	8
100	14	8
100	15	8
100	17	8
100	19	8
100	21	8
100	25	8
100	33	8
100	50	8
100	100	9

Tabella 4. Numero di iterazioni.

Dai risultati ottenuti è possibile notare come via via che la matrice perde la condizione di predominanza diagonale il metodo di Jacobi non fornisca più valori attendibili, inoltre, già quando vale la condizione di eguaglianza

$$a_{ii} = \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij}$$

il metodo esce dal ciclo "while" a causa della condizione sul superamento del numero massimo di iterazioni.
Il metodo di Gauss-Seidel ha un comportamento migliore e non converge più quando il fattore moltiplicativo della diagonale è 0,6.
Il metodo del gradiente coniugato, come ci si aspettava, non risente dell'annullarsi della condizione di predominanza diagonale in quanto la convergenza di tale metodo non dipende da questa ipotesi sulla matrice.

Se la matrice viene creata attraverso la funzione di "Matlab" *randn*, piuttosto che con il comando *rand*, è possibile notare come i tre metodi convergano meglio anche se la matrice non verifica più la condizione di predominanza diagonale.

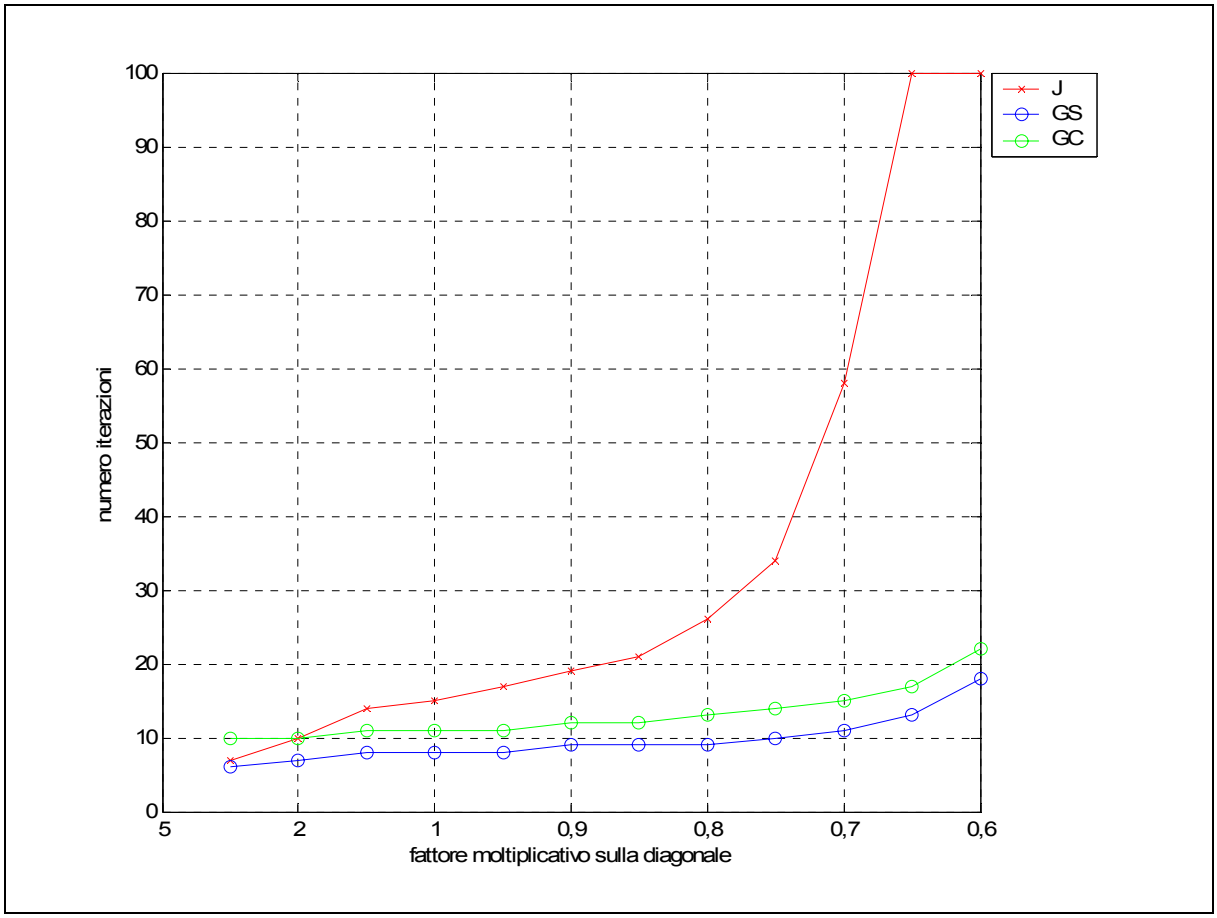


Figura 5. Numero iterazioni.

nJ	nGS	nGC=
7	6	10
10	7	10
14	8	11
15	8	11
17	8	11
19	9	12
21	9	12
26	9	13
34	10	14
58	11	15
100	13	17
100	18	22

Tabella 5. Numero iterazioni.

CAPITOLO 2.

2.0 Introduzione.

Come per i sistemi lineari, si prende ora in esame la risoluzione, sempre tramite metodi iterativi, di sistemi di equazioni non lineari.
Rispecchiando l'ordine seguito nella prima parte del testo saranno richiamati, inizialmente, una serie di concetti teorici importanti per poi analizzare due casi applicativi:

- risoluzione di una semplice equazione non lineare (2.1.2 Algoritmo);
- risoluzione di un piccolo sistema di equazioni non lineari (2.2.2 Introduzione al problema fisico.).

In ambedue i casi si farà riferimento a esempi specifici, rispettivamente: l'equazione di una circonferenza e la risoluzione di un problema di load flow.

2.1.0 Funzioni non lineari: richiami teorici.

I metodi iterativi volti alla risoluzione dei sistemi non lineari forniscono, a partire da un punto iniziale x_0, un set di soluzioni $\{x_0, x_1, \ldots\ldots\ldots\ldots, x_n\}$ che alla convergenza consentono di trovare la radice α che verifica la relazione $f(\alpha)= 0$.

Un concetto molto importante, relativo a questi metodi è la *velocità di convergenza,* questa, a sua volta, è espressa in funzione dell'*ordine del metodo (P).*
Un metodo convergente si dice di *ordine p* se:

$$\lim_{k-\infty} (|e_{k+1}|/|e_k|^p) = C \qquad (1)$$

dove C è la *costante asintotica dell'errore.*

Questo significa che al limite il rapporto

$$(|e_{k+1}|/|e_k|^p) \cong C \qquad (2)$$

e quindi

$$|e_{k+1}|=C*|e_k|^p . \qquad (3)$$

L'errore si riduce tanto più velocemente quanto più *p* è grande. Per p=1, affinché l'errore si riduca, C deve essere minore di uno.

Si consideri una radice semplice (α):

$$f(\alpha) = 0$$
$$f'(\alpha) \neq 0 \qquad (4)$$

$$f(x) = f(\alpha) + f'(\xi)(x-\alpha) \qquad (5)$$

dove

$$f'(\xi)(x-\alpha) \qquad (6)$$

rappresenta l'errore espresso nella forma di Lagrange in un punto ξ interno all'intervallo (x, α). Questo significa che

$$(x-\alpha) = f(x)/f'(\xi) \qquad (7)$$

ponendo

$$|f(x)|<\varepsilon \qquad (8)$$

risulta

$$|x-\alpha| = (1/|f'(\xi)|)* \varepsilon. \quad (9)$$

Dove:

$|x-\alpha| \Longrightarrow$ rappresenta l'errore sulla soluzione

$\varepsilon \Longrightarrow$ rappresenta l'errore sulla funzione.

Analogo discorso se la radice è di ordine *n*, anche se è da ricordare che maggiore è la molteplicità della radice, peggiore è il condizionamento.

Uno dei metodi iterativi utilizzabili per la risoluzione delle funzioni non lineari è quello di Newton. Tale metodo, a partire da una condizione iniziale x_0, esegue una iterazione descritta come:

$$x_{k+1} = x_k + (f(x_k)/f'(x_k)) \qquad (10)$$

con

$$f'(x_k) \neq 0. \qquad (11)$$

Come converge questo metodo?

Per analizzare la convergenza si ha necessariamente bisogno di un espressione dell'errore, a partire da un punto iniziale, e una radice:

$x_k \Longrightarrow$ punto iniziale

$\alpha \Longrightarrow$ radice

e sviluppiamo la funzione f(x) in serie di Taylor calcolata in α con punto iniziale in x_k. Si tronchi la serie al secondo termine e si esprima l'errore in forma di Lagrange:

$$f(\alpha) = f(x_k) + f'(x_k)*(\alpha- x_k) + 1/2*(f''(\xi)*(\alpha-x_k)^2) \quad (12)$$

dove

$$1/2*(f''(\xi)*(\alpha-x_k)^2) \Longrightarrow \text{esprime l'errore in forma di Lagrange}$$

sviluppando l'espressione e considerando l'implementazione del metodo si sostituirà

$e_{k+1} \Longrightarrow$ errore al passo "k+1" $(\alpha-x_{k+1})$

$e_k \Longrightarrow$ errore al passo "k" $(\alpha-x_k)$

ottenendo

$$|e_{k+1}| = 1/2*(|f''(\xi_k)|/|f'(x_k)|)*|e_k|^2 \qquad (13)$$

2.1.2 Algoritmo per la risoluzione di un semplice sistema non lineare.

Il seguente algoritmo vuole mostrare come sia possibile la risoluzione di una semplice sistema non lineare mediante l'applicazione del metodo di Newton.
Il semplice sistema non lineare è costituito da un'equazione non lineare (equazione della circonferenza) e un'equazione lineare:

$$(x_1)^2+(x_2)^2=2$$
$$x_1=x_2$$

intersecata dalla bisettrice.

ALGORITMO PER LA RISOLUZIONE DI UNA FUNZIONE
NON LINEARE
"PROGRAMMA SORGENTE"

```
L1: t=1;                          inizializzazione contatore;
L2: iterazione=[];                inizializzazione vettore iterazioni;
L3: Nmax=20;                      numero massimo di iterazioni;
L4: tau1=1e-10;                   errore massimo;

L5: for alfa=1:0.1:5
   L6: x=[alfa alfa]'             punto iniziale;
   L7:  k=0;                      inizializzazione contatore;
      L8: while (k<Nmax)
      L9: xv=x                    vettore punto iniziale;
      L10: [f]= effe(x);          richiama la funzione "effe";
      L11: [j]= jacob(x);         richiama la funzione "jacob";
      L12:  x=x-j\f;              calcola la soluzione;
         L13: if(norm(x-xv,2)<tau1*norm(x,2));   uscita dal ciclo per non osservanza della
                                                 condizione sulla tolleranza
         L14: break
       L15: else
          L16: k=k+1;             incremento contatore;
       L17: end
      L18: iterazione(t)=k;       incremento vettore iterazione;
   L19: end
   L20: t=t+1;                    incremento contatore ;
L21: end
L22: [stampa]=plotiter(iterazione);   grafico numero di iterazioni.
```

Il programma sopra riportato richiama nelle linee L10 ed L11 due funzioni che consentono di avere in ingresso l'equazioni che si intende studiare ("effe") e il calcolo dello Jacobiano per l'applicazione del metodo ("jacob").

Le radici della funzione che si vuole risolvere non sono altro che i punti di intersezione tra l'equazione di una circonferenza e l'equazione di una retta.

EFFE

```
function[f]=effe(x);
f =[x(1)^2+x(2)^2-2;x(1)-x(2)];
```

Le linee L5 ed L6, attraverso la definizione del parametro alfa, consentono di considerare la risoluzione del problema al variare del punto iniziale

Il grafico seguente permette di valutare il numero di iterazioni necessario alla convergenza del metodo in funzione del punto iniziale.

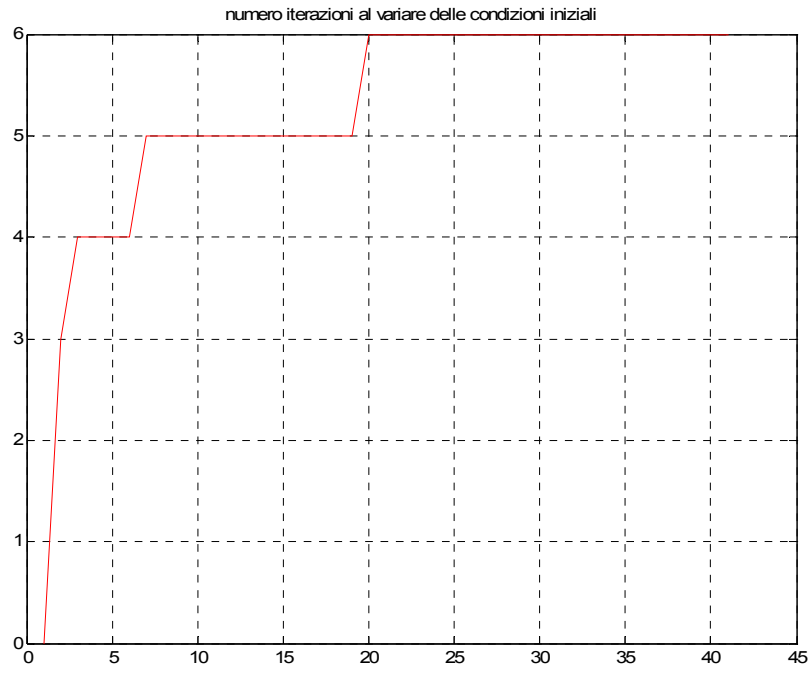


Figura 6. Numero delle iterazioni.

Con un numero molto basso di iterazioni si arriva alla convergenza del metodo indipendentemente dal punto iniziale scelto.
Il problema analizzato costituisce, quindi, un caso particolare per la risoluzione delle funzioni non lineari in quanto si può dimostrare come la convergenza del metodo di Newton sia strettamente legata alla scelta del punto iniziale.

2.2.0 Sistemi. Richiami teorici.

Consideriamo un sistema di n equazioni non lineari in n incognite

$$\begin{cases} f_1(x_1,.....,x_n) = 0 \\ f_2(x_1,.....,x_n) = 0 \\ . \\ . \\ . \\ . \\ f_n(x_1,.....,x_n) = 0 \end{cases} \qquad (14)$$

che si può anche scrivere nella forma

$$F(x)=0 \qquad (15)$$

dove $x = (x_1, \ldots, x_n)^T$ appartiene a R_n ϵ $F : R_n \rightarrow R_n$
F è una funzione vettoriale e la possiamo indicare come:

$$F(\underline{x}) = \begin{bmatrix} f_1(x_1,.....,x_n) \\ \\ \\ f_n(x_1,.....,x_n) \end{bmatrix} \qquad (16)$$

La funzione vettoriale F(x) può essere valutata attraverso vari metodi:

- Metodo di Newton Multidimensionale
- Metodi quasi-Newton
  - Metodo delle corde
  - Metodo di Brayden
  - Metodo di Newton-Jacobi
  - Metodo di Newton-Gauss-Seidel

Metodo di Newton multidimensionale:

La matrice F viene approssimata attraverso lo sviluppo in serie di Taylor troncata al primo ordine. Viene scelto come punto iniziale $x^{(k)}=(x_1^{(k)}, \ldots\ldots\ldots x_n^{(k)})^T$
quindi:

$$F(x) \sim F(x^{(k)}) + F'(x^{(k)})(x-x^{(k)}) \qquad (17)$$

Dove la F'($x^{(k)}$) è la matrice Jacobiana:

$$F'(x^{(k)}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}\left(x_1^{(k)},.....,x_n^{(k)}\right) & . & . & \frac{\partial f_1}{\partial x_n}\left(x_1^{(k)},.....,x_n^{(k)}\right) \\ . & . & . & . \\ . & . & . & . \\ \frac{\partial f_n}{\partial x_1}\left(x_1^{(k)},.....,x_n^{(k)}\right) & . & . & \frac{\partial f_n}{\partial x_n}\left(x_1^{(k)},.....,x_n^{(k)}\right) \end{bmatrix} \qquad (18)$$

dove i termini della matrice Jacobiana rappresentano le derivate parziali delle funzioni rispetto alle n variabili.
L'iterazione successiva viene valutata eguagliando a zero lo sviluppo in serie di Taylor:

$$x^{(k+1)}=x^{(k)} - (F'(x^{(k)})-1*F(x^{(k)}) \qquad (19)$$

Posto $x^{(k+1)}-x^{(k)} = h$, il problema precedente diventa equivalente alla soluzione del seguente sistema lineare:

$$(F'(x^{(k)})*h = - F(x^{(k)}) \qquad (20)$$

il problema di tale metodo è l'elevato costo computazionale legato al fatto che ad ogni passo viene risolto un sistema lineare e valutato lo Jacobiano.
Un alleggerimento nella computazione può essere ottenuto evitando di valutare lo Jacobinao per ogni iterazione e valutandolo solo ogni r iterazioni..
Quali sono le condizioni che si devono rispettare per la convergenza?

Definizione di funzione F: Rn→Rn totalmente differenziabile o differenziabile secondo Frechet in x ε Rn.
Se la Matrice Jacobiana F'(x) esiste e se:

$$\lim_{\|h\|\to 0} \frac{\|F(x+h)-F(x)-F'(x)h\|}{\|h\|} = 0 \qquad (21)$$

Definizione di punto di attrazione x*.
Sia x* ε R_n un punto fisso di G: $R_n \to R_n$, questo viene detto punto di attrazione se esiste un intorno S di x* tale che per ogni punto iniziale x(0) ε S esiste G(x(k)) con k=1,2,…… e la successione delle iterate converge a x*

$$x^*=G(x^*) \qquad (22)$$

Teorema di Ostrowski:
Se G: Rn→Rn è differenziabile in x*=G(x*) e se

$$\rho(G'(x^*))<1 \qquad (23)$$

allora x* è un punto di attrazione per l'iterazione $x^{(k+1)}=G(x^{(k)})$.
Se si applicano tali definizioni ai metodi quasi-Newton e Newton, e ricordando che lo Jacobiano della funzione di iterazione è

$$G(x)=x-(C(x))^{-1}F(x) \qquad (24)$$

se F è differenziabile in x* allora F(x*)=0 e se C(x*) è non singolare, allora risulta

$$G'(x^*)=I-(C(x^*))^{-1}F'(x^*) \qquad (25)$$

Teorema:
Se $F:R_n \rightarrow R_n$ è differenziabile in un intorno I_x della radice x*, se F' è continuo in x* e F'(x*) è non singolare, allora Newton converge localmente e la convergenza è detta superlineare:

$$\lim_{k\to\infty} \frac{\left\|x^{(k+1)} - x^*\right\|}{\left\|x^{(k)} - x^*\right\|} = 0 \quad (26)$$

inoltre se per qualsiasi valore di $x \in I_{x^*}$ esiste una costante L tale che

$$\left\|F'(x) - F'(x^*)\right\| \le L\left\|x - x^*\right\| \qquad (27)$$

allora si ha convergenza quadratica.

2.2.2 Introduzione al problema fisico.

[2] Il secondo esempio preso in considerazione è relativo ad una applicazione tipica nell'ingegneria elettrica e in particolar modo dell'analisi dei sistemi elettrici di potenza.
Le reti elettriche di potenza sono costituite dall'insieme delle macchine generatrici e degli utilizzatori collegati fra loro da sistemi elettrici di trasmissione aerei o sotterranei con interposti dei trasformatori.
Per lo studio in regime permanente di tali reti, le linee, i trasformatori e i carichi vengono ricondotti a circuiti equivalenti semplici costituiti da elementi passivi a costanti concentrate mentre i generatori a sorgenti ideali di tensione o corrente sinusoidali con in serie o in parallelo elementi di impedenza o ammettenza.
Per lo studio di questi sistemi a livello teorico si può ricorrere al *metodo dell'analisi nodale* che per una rete di *n* nodi fornisce un sistema di *n* equazioni congruenti in *n* incognite.
Quello al quale si farà riferimento nella totalità delle applicazioni, non è il metodo precedentemente citato, ma un metodo da esso derivante: *il metodo delle equazioni di load flow.*
Questo permette di definire un insieme di equazioni matematiche che simulano il comportamento di una rete elettrica di potenza in regime permanente.
La loro risoluzione permette di conoscere le grandezze significative di ogni nodo del sistema.
Trascurando la trattazione teorica del problema [2] , la forma finale delle equazioni viene di seguito riportata:

$$P_i = V_i^2 Y_{ii} \cos(-\theta_{ii}) + \sum_{\substack{j=1 \\ i \neq j}}^{n} V_i V_j Y_{ij} \cos(\delta_i - \delta_j - \theta_{ij})$$

$$Q_i = V_i^2 Y_{ii} sen(-\theta_{ii}) + \sum_{\substack{j=1 \\ i \neq j}}^{n} V_i V_j Y_{ij} sen(\delta_i - \delta_j - \theta_{ij}) \qquad (28)$$

Le equazioni riportate sopra si riferiscono a condizioni di funzionamento statico del sistema, non sono lineari e per un sistema a *n* nodi presentano *6n* variabili scalari:

- *n* moduli di tensione alle sbarre V_i
- *n* argomenti di tensione alle sbarre δ_i
- *n* potenze attive generate P_{gi}
- *n* potenze reattive generate Q_{gi}
- *n* potenze attive consumate P_{ci}
- *n* potenze reattive consumate Q_{ci}

Dato che le equazioni di load flow stabiliscono *2n* relazioni delle *6n* variabili definite sopra, *4n* devono essere arbitrarie.
In questo caso particolare però ciò non costituisce un problema perché abbiamo una serie di informazioni aggiuntive sul sistema, in particolare: i nodi di una rete possono essere classificati in due tipologie:

- nodi di generazione;
- nodi di carico,

Per i primi sono noti i valori di V_i e P_{gi} (*2n variabili*), mentre per i secondi sono noti i valori di P_{ci} e Q_{ci} (*2n variabili*), per un totale proprio di *4n* variabili note.
Le incognite effettive perciò saranno:

- argomenti di tensione ai nodi δ_i
- potenze reattive generate Q_{gi}.

Gli algoritmi risolutivi più diffusamente impiegati sono basati sui *metodi alla Gauss-Seidel* e *metodi alla Newton-Raphson.*
L'idea è quella di partire da un profilo ragionevolmente presunto delle tensioni ai nodi, e aggiornarlo iterativamente, approssimando la soluzione fino ai livelli di precisione prefissati.
L'approccio più efficiente al problema è quello utilizzare i *metodi alla Newton-Raphson,* basati su una linearizzazione successiva del problema di load flow.

Per vedere nel dettaglio l'applicazione del metodo si rimanda all'Appendice1.

2.2.3 Algoritmo per la risoluzione dei sistemi non lineari e risultati.

In questa parte, come già si è accennato nell'introduzione, si cerca di studiare un metodo in particolare, quello di Newton Raphson, applicato a un problema specifico.
Allo stesso modo, però, di quanto fatto per i sistemi lineari anche questa analisi ha bisogno di un riferimento, di una soluzione che possa essere considerata "esatta".
Per i sistemi lineari questa veniva creata appositamente, ma in questo caso non è possibile.
Tuttavia si dispone di un software specifico per la risoluzione del "problema del load flow" denominato **"powerworld** educational version**".**
Inizialmente perciò si è risolto il problema attraverso quest'ultimo (Appendice2) e la soluzione ottenuta è stata poi utilizzata come punto iniziale del metodo di Newton.
Si evita perciò che una scelta non corretta del punto iniziale possa determinare dei problemi e se il metodo funziona correttamente, lo stesso non dovrebbe discostarsi troppo dal punto iniziale, dato che questo rappresenta anche la soluzione a cui dovrebbe convergere.
Analisi dell'algoritmo:

ALGORITMO FUNZIONE NON LINEARE
"PRINCIPALE"

L1: **x=[0 -2.79 138 -106.96 109.37 237.575]';** inizializzazione del vettore iniziale nei prossimo alla soluzione reale ottenuta attraverso n software specifico
L2: **Nmax=10;** numero massimo di iterazioni
L3: **tau1=1e-2;** errore relativo massimo sul calcolo della soluzione del sistema
L4: **k=0;** inizializzazione contatore
L5: **while (k<Nmax)**
L6: **xv=x;** vettore soluzione vecchia
L7: **[f]=sist(x);** definizione della funzione delle equazioni del sistema fisico
L8: **[j]=jacsist(x);** valutazione dello Jacobiano nel punto di coordinate x

L9: **h=j\f';** calcolo del passo
L10: **x=x-h** metodo
L11: **if(norm(x-xv,2)<tau1*norm(x,2));** criterio di uscita
L12: **break** uscita dal ciclo
L13: **else**
L14: **k=k+1;** incremento contatore
L15: **end**
L16: **end**

I risultati ottenuti sono confrontati con quelli ottenuti attraverso un software specifico (Appendice 2).

"principale" 1.0e+016 *	Software specifico
6.9132	0
6.9132	-2.79
0.0000	138
6.9132	-106.96
0.0000	109.37
0.0000	237.575

Tabella 6. confronto soluzioni.

Il metodo così come viene applicato in Matlab diverge in maniera evidente.
Per capirlo meglio si può calcolare quanto la soluzione calcolata si discosti da quella "esatta", ad ogni passo.
Quantità esprimibile attraverso una norma due della differenza fra la i-esima soluzione, calcolata all'i-esimo passo, e la soluzione valutata attraverso il software specifico.

```
err=norm(x-x0,2);
scostamento=[scostamento,err];
```

Dove "x" è la soluzione calcolata al generico passo e "x0" è la soluzione esatta.
I risultati sono raccolti nella tabella riportata sotto:

passo	Scostamento 1.0e+017 *
1	0.9800
2	0.9232
3	0.7801
4	1.0375
5	0.6417
6	1.7025
7	1.3890
8	1.3133
9	1.3002
10	1.1974

Tabella 7. scostamento della soluzione calcolata.

Questo mostra come la distanza della soluzione calcolata da quella che si vorrebbe ottenere rimane dal primo all'ultimo passo pressoché uguale, il problema è che già dal primo passo il metodo si allontana in maniera drastica dall'obbiettivo (la soluzione calcolata con il software specifico).

Un'indicazione sul perché questo accada la fornisce direttamente lo stesso programma attraverso la segnalazione di warning riportata sotto:

Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 4.588685e-018.
> In C:\matlabR12\work\sistemi nn lineari\prova\principrova.m at line 47

Le possibili cause d'errore, quindi, potrebbero essere:
- valutazione errata delle funzioni;
- valutazione errata delle derivate;
- stima dello jacobiano;
- fenomeni di dipendenza lineare tra alcune variabili.

Una volta ricontrollate le funzioni e le derivate non resta che passare alla stima dello jacobiano.
Per affrontare questo punto si può ricorrere alla funzione **"fsolve"** dell' **"optimization toolbox"** di Matlab.
Questa consente la risoluzione del medesimo problema attraverso **il metodo dei minimi quadrati.**
Quella di seguito riportato è la chiamata della funzione

[x,fval,exitflag,output,jacobian]=fsolve(@sist, [0 -2 138 -106.96 109.37 237.575]')

La funzione produce una serie di uscite fra le quali:
- jacobian;
- x.

La prima è quella che fornisce una stima dello jacobiano, riportata di sotto:

---

jacobian =
1.0e+006 *

2.2836	**-1.5155**	**-0.0002**	**-0.7681**	**0**	**0**
-0.3756	**0.2922**	**-0.0022**	**0.0834**	**-0.0000**	**0**
-1.5199	**2.8474**	**0.0000**	**-1.3276**	**0**	**0**
0.2686	**-0.2515**	**-0.0037**	**-0.0171**	**0**	**-0.0000**
-0.7249	**-1.2871**	**0.0001**	**2.0120**	**0**	**0**
0.2674	**0.3257**	**0.0052**	**-0.5931**	**0**	**0**

---

questa valutazione viene confrontata con quella, riportata sotto, ottenuta calcolando manualmente la forma delle derivate e sostituendo, nel corso delle iterazioni, i valori delle variabili "x" calcolate .

Il valore finale è:

J=
1.0e+006 *

1.4443	**-1.5177**	**0.0019**	**0.0734**	**0**	**0**
-0.1564	**0.2804**	**0.0011**	**-0.1240**	**-0.0000**	**0**
-1.5177	**1.3683**	**0.0030**	**0.1495**	**0**	**0**
0.2804	**-0.0830**	**-0.0023**	**-0.1975**	**0**	**-0.0000**
0.1210	**0.1914**	**-0.0032**	**-0.3123**	**0**	**0**
0.0783	**0.1572**	**0.0068**	**-0.2355**	**0**	**0**

Le due matrici mostrano valori estremamente simili, dello stesso ordine di grandezza, il ché significa che la valutazione dello jacobiano utilizzata nell'algoritmo "principale" può essere ritenuta sufficientemente corretta.
Nonostante questo, le soluzioni ottenute sono fortemente differenti:

"fsolve" 1.0e+003 *	"principale" 1.0e+016 *	Software specifico
-0.0008	6.9132	0
-0.0008	6.9132	-2.79
0.3546	0.0000	138
-0.1077	6.9132	-106.96
1.2741	0.0000	109.37
3.3410	0.0000	237.575

Tabella 8. confronto soluzioni

In ambedue i casi i risultati si discostano da quelli "esatti" (ottenuti con il software specifico), ma è piuttosto evidente come nel caso di "principale" non sia rispettato nemmeno l'ordine di grandezza.
C'è inoltre da precisare che l'uso migliore della funzione "fsolve" richiede l'impostazione di una serie di parametri ("options") che in tal sede non è presa in considerazione.
Dei possibili problemi elencati in precedenza non resta che considerare la possibilità che ci siano dei fenomeni di dipendenza lineare tra alcune variabili.
Resta da capire quali.
Per provare ad individuarne almeno una analizziamo il rango della matrice jacobiana: più precisamente valutiamo inizialmente la possibilità che una equazione del sistema dipenda linearmente dalle altre.

```
rank(j([1 2 4 5 6],:))
ans = 5
rank(j([1 3 4 5 6],:))
ans =4
```

Da questo si nota come la terza riga sia dipendente dalle altre.
Possiamo supporre, quindi di "eliminare" la terza riga.
Analogo ragionamento lo si ripete per le colonne:

---

rank(j(:,[1 3 4 5 6]))
ans =5
rank(j(:,[1 2 4 5 6]))
ans =4

Questa analisi evidenzia una dipendenza lineare della seconda variabile nella terza equazione. (non si esclude che vi sia dipendenza tra altre variabili).
Alla luce di questo si prova ad operare il cambio di variabili in modo da ridurre il sistema 6x6 ad un sistema 5x5.
Come?
In sostanza "eliminando" la terza equazione e effettuando un cambio di variabili come quello illustrato sotto:

$$alfa(1) = x(1) - x(2)$$
$$alfa(2) = x(2) - x(4)$$

mentre le restanti variabili sono considerate come

$$alfa(3) = x(3)$$
$$alfa(4) = x(5)$$
$$alfa(5) = x(6).$$

Seguendo queste stesse relazioni si valuta il nuovo vettore iniziale pari a:

---

alfa=[2.79, 104.17, 138, 109.37, 237.575]'

---

mentre il sistema di equazioni diventa:

---

f(1) = -P1+V1^2*Y11*cos(teta11)+V1*V2*Y12*cos(alfa(1)-teta12)
+V1*alfa(3)*Y13*cos(alfa(2)+alfa(1)-teta13);

f(2) = -alfa(4)-V1^2*Y11*sin(teta11)+V1*V2*Y12*sin(alfa(1)-teta12)
+V1*alfa(3)*Y13*sin(alfa(1)-teta13);

f(3) = -alfa(5)-V2^2*Y22*sin(teta22)+V2*V1*Y21*sin(-alfa(1)-teta21)
+V2*alfa(3)*Y23*sin(alfa(2)-teta23);

f(4) = -P3+alfa(3)^2*Y33*cos(teta33)+alfa(3)*V1*Y31*cos(-alfa(1)-alfa(2)-teta31)
+alfa(3)*V2*Y32*cos(-alfa(2)-teta32);

f(5) = -Q3-alfa(3)^2*Y33*sin(teta33)+alfa(3)*V1*Y31*sin(-alfa(1)-alfa(2)-teta31)
+alfa(3)*V2*Y32*sin(-alfa(2)-teta32);

Ovviamente, anche lo jacobiano risulta modificato.
Applicando il metodo esattamente nello stesso modo di prima si ottiene il seguente risultato:

valore iniziale (soluzione)	Soluzione finale 1.0e+005 *
2.79	-0.0002
104.17	0.0010
138	0.0034
109.37	0.7286
237.575	1.3984

Tabella 9. soluzioni del "sistema ridotto".

Dato che il punto iniziale, anche in questo caso, rappresenta la soluzione "esatta" appare evidente come anche ora, dopo aver "ridotto" il sistema, il metodo non converga.
Rispetto a prima, però, lo scostamento è maggiormente contenuto e il ché può indicare un qualche effetto positivo della riduzione del sistema.
Analogamente a quanto fatto per il sistema originale si ri-applica la funzione "fsolve" e si mettono a confronto i risultati:

[alfa,fval,exitflag,output,jacobian]=fsolve(@sistprova, [2.79, 104.17, 138, 109.37, 237.575]')

---

valore iniziale (soluzione)	Soluzione finale	Soluzione finale (fsolve)
	1.0e+005 *	1.0e+003 *
2.79	-0.0002	-0.0000
104.17	0.0010	0.1069
138	0.0034	0.3553
109.37	0.7286	0.5037
237.575	1.3984	1.2334

Tabella 10. confronto soluzioni

Conclusioni.

L'analisi effettuata ha mostrato risultati piuttosto diversi: mentre per i sistemi lineari le soluzioni ottenute rispecchiano quanto ci si aspettava dalla letteratura tecnica di riferimento, per i sistemi non lineari si sono ottenuti risultati abbastanza scadenti.

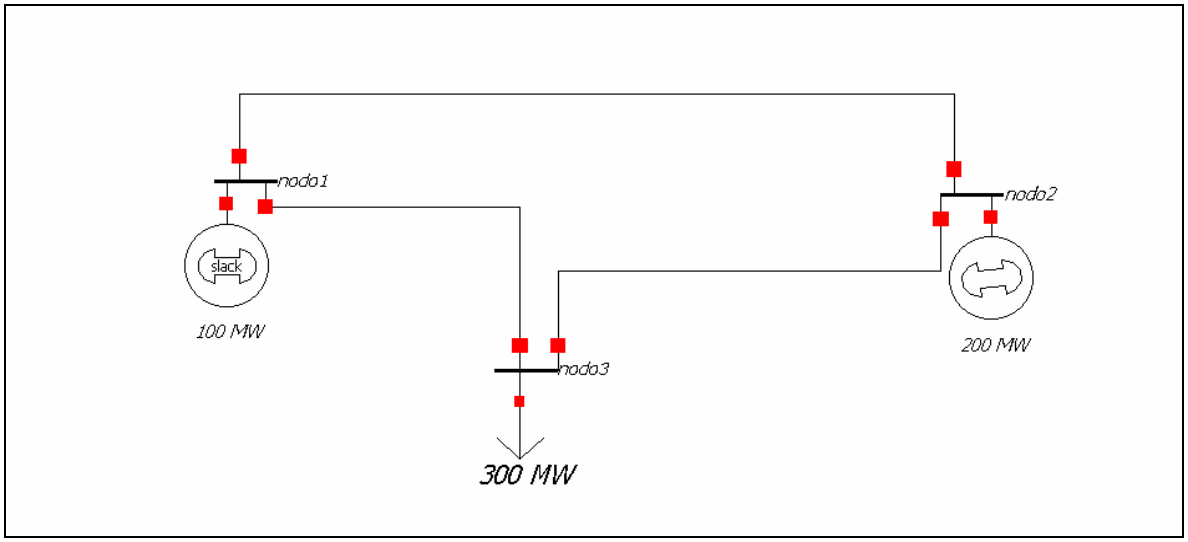
Questo era in parte prevedibile data la complessità sia del metodo che del problema fisico esaminato (load-flow). Si nota infatti la differenza fra i risultati ottenuti per la semplice funzione non lineare e il sistema preso in esame.

Ciò può significare che i metodi generali adottati per la risoluzione dei sistemi non lineari siano talvolta inaffidabili se non si ha a priori la conoscenza specifica del problema. Sarà dunque necessario condurre un'analisi ad hoc per il problema fisico con cui si ha a che fare in modo da poter implementare il miglior metodo possibile adattandolo di volta in volta alle specificità del caso.

APPENDICE1

Applicazione del metodo di load flow

Il problema applicativo preso in esame è stato risolto con il già citato metodo delle equazioni di load flow.
Ovvero con la risoluzione del sistema di equazioni non lineari risolte attraverso il metodo di Newton Raphson.
Riportiamo di seguito lo schema di riferimento:



Il sistema è costituito da tre nodi:

- nodo1: è un nodo di generazione per il quale si definisce P1=100e3 [KW] e V1=380[KV];
- nodo2: è un nodo di generazione per il quale si definisce P2=200e3 [KW] e V2=380[KV];
- nodo3: è un nodo di carico per il quale si definisce P3=300e3 [KW] e Q3=150e3[MVAR].

I nodi sono interconnessi fra loro attraverso tre linee di trasmissione di caratteristiche:

- linea12: R12= 0.017, X12=0.092;
- linea13: R13= 0.039, X13= 0.17;
- linea23: R23= 0.0119, X23= 0.1008.

Attraverso la procedura illustrata in **2.2.2**, si ricavano i valori complessi di ammettenza:

-Y: modulo della ammettenza;
- teta: argomento della ammettenza.

Per quanto visto il sistema sarà decritto attraverso un modello matematico di sei equazioni non lineari:

(nodo1)
$P_1= V_1^2*Y_{11}*\cos(teta_{11}) + V_1*V_2*Y_{12}*\cos(alfa_1-alfa_2-teta_{12}) + V_1*V_3*Y_{13}*\cos(alfa_1-alfa_3-teta_{13})$
$Q_1= -V_1^2*Y_{11}*\sin(teta_{11}) + V_1*V_2*Y_{12}*\sin(alfa_1-alfa_2-teta_{12}) + V_1*V_3*Y_{13}*\sin(alfa_1-alfa_3-teta_{13})$
(nodo2)
$P_2= V_2^2*Y_{22}*\cos(teta_{22}) + V_2*V_1*Y_{21}*\cos(alfa_2-alfa_1-teta_{21}) + V_2*V_3*Y_{23}*\cos(alfa_2-alfa_3-teta_{23})$
$Q_2= -V_2^2*Y_{22}*\sin(teta_{22}) + V_2*V_1*Y_{21}*\sin(alfa_2-alfa_1-teta_{21}) + V_2*V_3*Y_{23}*\sin(alfa_2-alfa_3-teta_{23})$
(nodo3)
$P_3= V_3^2*Y_{33}*\cos(teta_{33}) + V_3*V_1*Y_{31}*\cos(alfa_3-alfa_1-teta_{31}) + V_3*V_2*Y_{32}*\cos(alfa_2-alfa_2-teta_{32})$

$Q_3 = -V_3^2 * Y_{33} * \sin(teta_{33}) + V_3 * V_1 * Y_{31} * \sin(alfa_3 - alfa_1 - teta_{31}) + V_3 * V_2 * Y_{32} * \sin(alfa_2 - alfa_2 - teta_{32})$

Le incognite del sistema riportato sopra sono:

- $alfa_1 = x(1)$
- $alfa_2 = x(2)$
- $V_3 = x(3)$
- $alfa_3 = x(4)$
- $Q_1 = x(5)$
- $Q_2 = x(6)$.

Sostituendo nelle equazioni riportate sopra si ottiene:

(nodo1)

$f(1) = -P_1 + V_1^2 * Y_{11} * \cos(teta_{11}) + V_1 * V_2 * Y_{12} * \cos(x(1) - x(2) - teta_{12}) + V_1 * x(3) * Y_{13} * \cos(x(1) - x(4) - teta_{13})$

$f(2) = -x(5) - V_1^2 * Y_{11} * \sin(teta_{11}) + V_1 * V_2 * Y_{12} * \sin(x(1) - x(2) - teta_{12}) + V_1 * x(3) * Y_{13} * \sin(x(1) - x(4) - teta_{13})$

(nodo2)

$f(3) = -P_2 + V_2^2 * Y_{22} * \cos(teta_{22}) + V_2 * V_1 * Y_{21} * \cos(x(2) - x(1) - teta_{21}) + V_2 * x(3) * Y_{23} * \cos(x(2) - x(4) - teta_{23})$

$f(4) = -x(6) - V_2^2 * Y_{22} * \sin(teta_{22}) + V_2 * V_1 * Y_{21} * \sin(x(2) - x(1) - teta_{21}) + V_2 * x(3) * Y_{23} * \sin(x(2) - x(4) - teta_{23})$

(nodo3)

$f(5) = -P_3 + x(3)^2 * Y_{33} * \cos(teta_{33}) + x(3) * V_1 * Y_{31} * \cos(x(4) - x(1) - teta_{31}) + x(3) * V_2 * Y_{32} * \cos(x(4) - x(2) - teta_{32})$

$f(6) = -Q_3 - x(3)^2 * Y_{33} * \sin(teta_{33}) + x(3) * V_1 * Y_{31} * \sin(x(4) - x(1) - teta_{31}) + x(3) * V_2 * Y_{32} * \sin(x(4) - x(2) - teta_{32})$

Per risolvere il sistema attraverso il metodo scelto, come illustrato in **2.2.3**, occorre calcolare le derivate parziali delle funzioni riportate rispetto a tutte le variabili.
La matrice Jacobiana avrà dimensioni sei per sei:

$(\delta f_{(1)} \backslash \delta x_{(1)}) = -V_1 * V_2 * Y_{12} * \sin(x(1) - x(2) - teta_{12}) - V_1 * x(3) * Y_{13} * \sin(x(1) - x(4) - teta_{13})$

$(\delta f_{(1)} \backslash \delta x_{(2)}) = V_1 * V_2 * Y_{12} * \sin(x(1) - x(2) - teta_{12})$

$(\delta f_{(1)} \backslash \delta x_{(3)}) = V_1 * Y_{13} * \cos(x(1) - x(4) - teta_{13})$

$(\delta f_{(1)} \backslash \delta x_{(4)}) = V_1 * x(3) * Y_{13} * \sin(x(1) - x(4) - teta_{13})$

$(\delta f_{(1)} \backslash \delta x_{(5)}) = 0$

$(\delta f_{(1)} \backslash \delta x_{(6)}) = 0$

$(\delta f_{(2)} \backslash \delta x_{(1)}) = V_1 * V_2 * Y_{12} * \cos(x(1) - x(2) - teta_{12}) + V1 * x(3) * Y_{13} * \cos(x(1) - x(4) - teta_{13})$

$(\delta f_{(2)} \backslash \delta x_{(2)}) = -V_1 * V2 * Y_{12} * \cos(x(1) - x(2) - teta_{12}) + V_1 * x(3) * Y_{13} * \cos(x(1) - x(4) - teta_{13})$

$(\delta f_{(2)} \backslash \delta x_{(3)}) = V_1 * Y_{13} * \sin(x(1) - x(4) - teta_{13})$
$(\delta f_{(2)} \backslash \delta x_{(4)}) = -V_1 * x(3) * Y_{13} * \cos(x(1) - x(4) - teta_{13})$
$(\delta f_{(2)} \backslash \delta x_{(5)}) = -1$
$(\delta f_{(2)} \backslash \delta x_{(6)}) = 0$

$(\delta f_{(3)} \backslash \delta x_{(1)}) = V_2 * V_1 * Y_{21} * \sin(x(2) - x(1) - teta_{21})$
$(\delta f_{(3)} \backslash \delta x_{(2)}) = -V_2 * V_1 * Y_{21} * \sin(x(2) - x(1) - teta_{21}) + V_2 * x(3) * Y_{23} * \sin(x(2) - x(4) - teta_{23})$
$(\delta f_{(3)} \backslash \delta x_{(3)}) = V_2 * Y_{23} * \cos(x(2) - x(4) - teta_{23})$
$(\delta f_{(3)} \backslash \delta x_{(4)}) = V_2 * x(3) * Y_{23} * \sin(x(2) - x(4) - teta_{23})$
$(\delta f_{(3)} \backslash \delta x_{(5)}) = 0$
$(\delta f_{(3)} \backslash \delta x_{(6)}) = 0$

$(\delta f_{(4)} \backslash \delta x_{(1)}) = -V_2 * V_1 * Y_{21} * \cos(x(2) - x(1) - teta_{21})$
$(\delta f_{(4)} \backslash \delta x_{(2)}) = V_2 * V_1 * Y_{21} * \cos(x(2) - x(1) - teta_{21}) + V_2 * x(3) * Y_{23} * \sin(x(2) - x(4) - teta_{23})$
$(\delta f_{(4)} \backslash \delta x_{(3)}) = V_2 * Y_{23} * \sin(x(2) - x(4) - teta_{23})$
$(\delta f_{(4)} \backslash \delta x_{(4)}) = -V_2 * x(3) * Y_{23} * \cos(x(2) - x(4) - teta_{23})$
$(\delta f_{(4)} \backslash \delta x_{(5)}) = 0$
$(\delta f_{(4)} \backslash \delta x_{(6)}) = -1$

$(\delta f_{(5)} \backslash \delta x_{(1)}) = x(3) * V_1 * Y_{31} * \sin(x(4) - x(1) - teta_{31})$
$(\delta f_{(5)} \backslash \delta x_{(2)}) = x(3) * V_2 * Y_{32} * \sin(x(4) - x(2) - teta_{32})$
$(\delta f_{(5)} \backslash \delta x_{(3)}) = 2 * x(3) * Y_{33} * \cos(teta_{33}) + V_1 * Y_{31} * \cos(x(4) - x(1) - teta_{31}) + V_2 * Y_{32} * \cos(x(4) - x(2) - teta_{32})$
$(\delta f_{(5)} \backslash \delta x_{(4)}) = -x(3) * V1 * Y_{31} * \sin(x(4) - x(1) - teta_{31}) - x(3) * V2 * Y_{32} * \sin(x(4) - x(2) - teta_{32})$
$(\delta f_{(5)} \backslash \delta x_{(5)}) = 0$
$(\delta f_{(5)} \backslash \delta x_{(6)}) = 0$

$(\delta f_{(6)} \backslash \delta x_{(1)}) = -x(3) * V_1 * Y_{31} * \cos(x(4) - x(1) - teta_{31})$
$(\delta f_{(6)} \backslash \delta x_{(2)}) = -x(3) * V_2 * Y_{32} * \cos(x(4) - x(2) - teta_{32})$
$(\delta f_{(6)} \backslash \delta x_{(3)}) = -2 * x(3) * Y_{33} * \sin(teta_{33}) + V_1 * Y_{31} * \sin(x(4) - x(1) - teta_{31}) + V_2 * Y_{32} * \sin(x(4) - x(2) - teta_{32})$
$(\delta f_{(6)} \backslash \delta x_{(4)}) = x(3) * V1 * Y_{31} * \cos(x(4) - x(1) - teta_{31}) + x(3) * V2 * Y_{32} * \cos(x(4) - x(2) - teta_{32})$
$(\delta f_{(6)} \backslash \delta x_{(5)}) = 0$
$(\delta f_{(6)} \backslash \delta x_{(6)}) = 0$

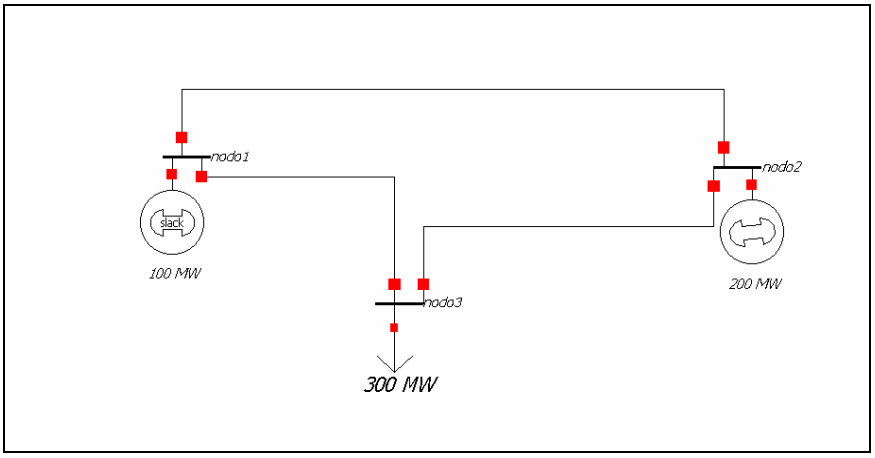
APPENDICE2

Uso del software specifico “powerworld”

L’uso del software per analisi delle reti elettriche denominato “powerworld ” offre svariate possibilità, ma in questa parte si descriverà brevemente i passi effettuati per la risoluzione del problema in esame.

1) definizione del sistema.

E’ sufficiente disegnare lo schema della rete che si vuole analizzare

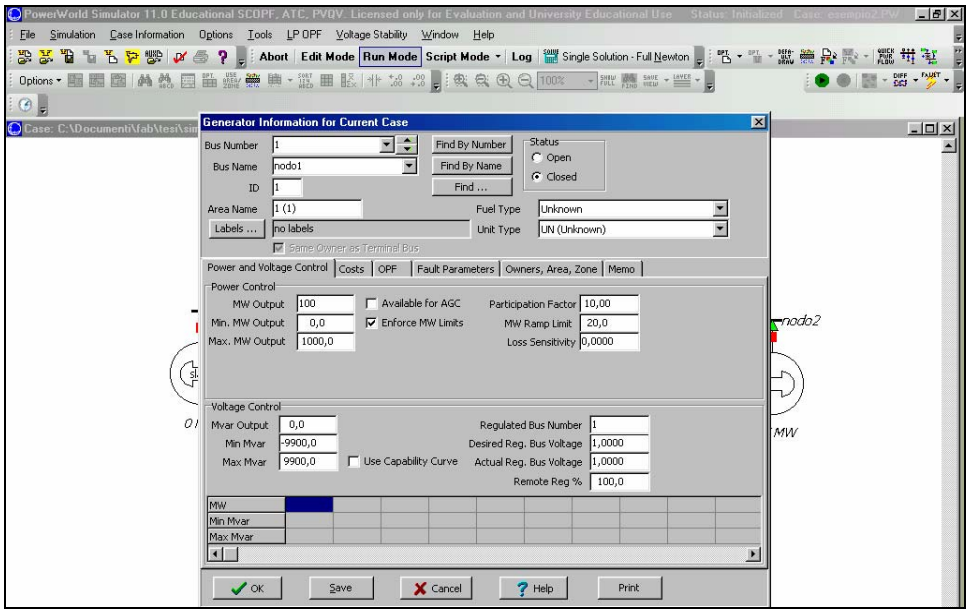


I nodi sono caratterizzati attraverso il componente collegato:

- nodo1: generazione;
- nodo 2: generazione;
- nodo 3: carico.

2) caratteristiche del sistema

Le caratteristiche dei nodi (P1,P2,P3,V1,V2,…Y12,Y13…) sono fissate attraverso le apposite finestra.
Si riportano sotto un esempio relativo al nodo1:

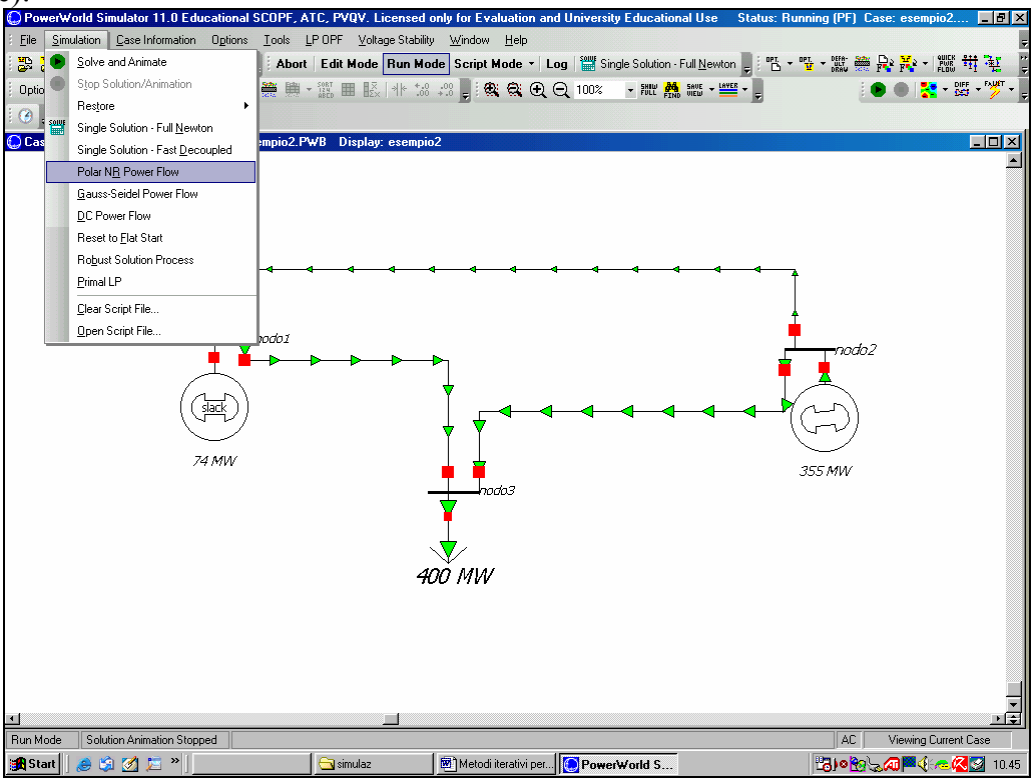


3) risoluzione del sistema

Il problema del load flow matematicamente si traduce nella risoluzione di un sistema di equazioni di non lineari.

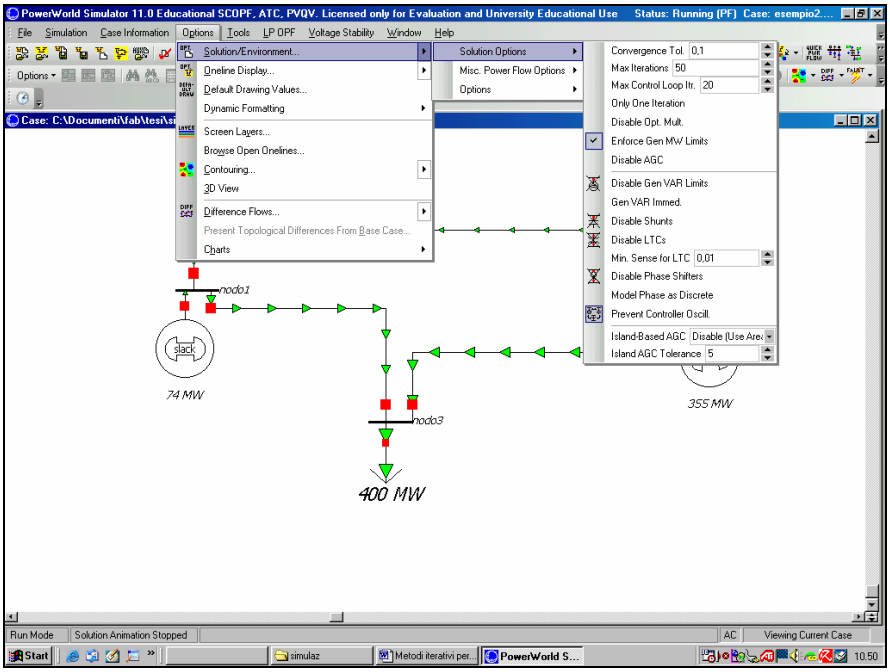
Tale risoluzione può essere ottenuta attraverso diversi metodi.

La possibilità di scelta che offre il software in esame è abbastanza ampia (come mostra l'immagine sottostante):



Nell'immagine è evidenziata la scelta del metodo di Newton-Raphson.

Una volta scelto il metodo è possibile anche stabilirne i parametri (precisione, numero massimo di iterazioni,ecc.) sempre attraverso una apposita finestra di dialogo:


PowerWorld Simulator 11.0 Educational SCOPF, ATC, PVQV. Licensed only for Evaluation and University Educational Use
Status: Running (PF) Case: esempio2....
File Simulation Case Information Options Tools LP OPF Voltage Stability Window Help
Solution/Environment...
Oneline Display...
Default Drawing Values...
Dynamic Formatting
Screen Layers...
Browse Open Onelines...
Contouring...
3D View
Difference Flows...
Present Topological Differences From Base Case...
Charts
Solution Options
Misc. Power Flow Options
Options
Convergence Tol. 0,1
Max Iterations 50
Max Control Loop Itr. 20
Only One Iteration
Disable Opt. Mult.
Enforce Gen MW Limits
Disable AGC
Disable Gen VAR Limits
Gen VAR Immed.
Disable Shunts
Disable LTCs
Min. Sense for LTC 0,01
Disable Phase Shifters
Model Phase as Discrete
Prevent Controller Oscill.
Island-Based AGC Disable (Use Area
Island AGC Tolerance 5
nodo1
slack
74 MW
nodo3
400 MW
355 MW
Run Mode
Solution Animation Stopped
AC
Viewing Current Case
Start
simulaz
Metodi iterativi per...
PowerWorld S...
10.50

BIBLIOGRAFIA.

[1] G.Rodriguez
"Dispense di calcolo numerico2"
Facoltà di ingegneria-corso di laurea in ingegneria elettrica.
[2] Vincenzo Cataliotti
"Impianti Elettrici" vol. II .
pag. 95-99.