Serie 2b - Soluzione
Sistemi Lineari – Metodi Iterativi

Esercizio 1.1

I comandi Matlab® vengono riportati supponendo un unico script per la risoluzione dell'esercitazione, per cui non sono presenti le `>>`.

1. La matrice può essere costruita tramite i seguenti comandi:

   ```
   n = 100;
   R1 = 1;
   R2 = -2;
   A = diag(R2*ones(n,1)) + diag(R1*ones(n-1,1),-1);
   A(1,:) = 1;
   ```

   Il comando `nnz` restituisce il numero di elementi non nulli della matrice data in ingresso; l'output in questo caso è:

   ```
   nz_el = nnz(A)
   nz_el =
   298
   ```

   Possiamo convertire la matrice in formato sparso tramite il comando `sparse` e verificare la diversa occupazione di memoria con il comando `whos`:

   ```
   Asp = sparse(A);
   whos A Asp
     Name        Size              Bytes  Class     Attributes

     A         100x100             80000  double
     Asp       100x100              5576  double    sparse
   ```

   Si può vedere come il formato sparso riduca la memoria richiesta per memorizzare una stessa matrice rispetto al formato pieno.

2. Si calcoli la fattorizzazione LU sulla matrice A (funzione `lugauss` del Laboratorio 5):

   ```
   [L,U] = lugauss(A);
   figure, spy(L)
   figure, spy(U)
   figure, spy(A)
   ```

Il confronto del pattern delle matrici A e U mostra la comparsa del fenomeno del *fill-in*, per cui a partire da una matrice sparsa, la U rimane triangolare superiore ma piena, risultando quindi più complicata della matrice di partenza. Questo fa sì che in questi casi sia preferibile abbandonare la fattorizzazione LU e passare ad altri metodi.

3. Le matrici di iterazione dei due metodi si calcolano a partire dalla definizione:

```
Dinv = diag(1./diag(A));
Bj = eye(n) - Dinv*A;
T = tril(A);
Bgs = eye(n) - inv(T)*A;
rho_j = max(abs(eig(Bj)))
rho_gs = max(abs(eig(Bgs)))
  rho_j =
      0.7071
  rho_gs =
      1.0000
```

   Possiamo costruire esplicitamente la matrice di iterazione del metodo di Jacobi con l'istruzione `Dinv = diag(1./diag(A))`: il comando interno restituisce un vettore con i reciproci dei valori diagonali di A, che il comando esterno sistema sulla diagonale della matrice quadrata `Dinv`. Dal calcolo del raggio spettrale delle matrici si può concludere che in questo caso solo il metodo di Jacobi converge, in quanto l'autovalore massimo risulta in modulo strettamente minore di 1. Al contrario, la condizione non è verificata per la matrice di Gauss–Seidel, il cui raggio spettrale è pari a 1.

4. Si veda il file `jacobi.m`.

5. Per risolvere il sistema è sufficiente richiamare la funzione, dopo aver definito tutti i parametri di ingresso che richiede:

```
b = ones(n,1);
b(1) = 2;
toll = 1e-6;
x0 = zeros(n,1);
nmax = 1000;
[xn,k]=jacobi(A,b,x0,toll,nmax)
    xn =
       50.0000
       24.5000
       11.7500
           ...
       -1.0000
    k =
        47
```

   Il metodo converge in 47 iterazioni.

Esercizio 1.2

1. Si costruiscono la matrice A e il termine noto $\mathbf{b}$:

```
n = 7;
A = diag(9*ones(1,n))   + diag(-3*ones(1,n-1),1) + diag(-3*ones(1,n-1),-1) + ...
    diag(ones(1,n-2),2) + diag(ones(1,n-2),-2)
b = [7 4 5 5 5 4 7]';
```

2. Una matrice è a dominanza diagonale stretta per righe se l'elemento sulla diagonale principale, in modulo, è maggiore della somma dei moduli degli altri elementi della riga. Si può verificare con i seguenti comandi:

```
Adiag = diag(abs(A));
Aout_diag = sum(abs(A),2) - diag(abs(A));
if (Adiag > Aout_diag)
    disp('La matrice e'' diagonale dominante stretta per righe')
else
    disp('La matrice non e'' diagonale dominante')
end
```

   Si noti l'istruzione condizionale: il risultato del controllo logico è un vettore di 1 o 0 a seconda che il risultato del controllo sui corrispondenti elementi dei due vettori `Adiag` e `Aout_diag` sia vero o falso. A sua volta, il vettore risultante restituirà "vero" (e cioè in sostanza entrerà nell' *if*) se tutti i suoi elementi sono "veri", cioè uguali a 1.

3. Per controllare se la matrice è simmetrica si deve verificare che coincida con la sua trasposta. Se la matrice è simmetrica allora avrà tutti autovalori reali. Per controllare se una matrice simmetrica è definita positiva bisogna verificare se tutti i suoi autovalori sono positivi, tramite il comando Matlab® `eig`.

```
if (A == A')
    if (eig(A) > 0)
        disp('La matrice e'' simmetrica e definita positiva')
    else
        disp('La matrice e'' simmetrica ma non definita positiva')
    end
else
    disp('La matrice non e'' simmetrica')
end
```

   Per il risultato del doppio controllo logico valgono le stesse considerazioni fatte in precedenza.

4. Si veda il file `gs.m`.

5. Per calcolare la soluzione con il metodo di Gauss–Seidel:

```
toll = 1e-6;
x0 = zeros(n,1);
nmax = 1000;
[x_gs,k_gs]=gs(A,b,x0,toll,nmax)
   x_gs =
       1.0000
          ...
       1.0000
   k_gs =
       12
```

Il metodo converge in 12 iterazioni.

6. La soluzione calcolata dal metodo di Jacobi, come è lecito aspettarsi, è analoga a quella calcolata in precedenza con gli stessi parametri, ma viene raggiunta in un numero di iterazioni maggiore:

```
[x_jac,k_jac]=jacobi(A,b,x0,toll,nmax);
  k_jac =
          49
```

Questo è dovuto al fatto che il raggio spettrale della matrice di iterazione del metodo di Jacobi più grande di quello di Gauss–Seidel:

```
Dinv = diag(1/.diag(A));
Bj = eye(n) - Dinv * A;
T = tril(A);
Bgs = eye(n) - inv(T) * A;
rho_j = max(abs(eig(Bj)))
rho_gs = max(abs(eig(Bgs)))
  rho_j =
      0.7823
  rho_gs =
      0.2522
```

Come da teoria, infatti, un raggio spettrale in modulo più basso accelera la convergenza del metodo, poichè vale la stima sull'abbattimento dell'errore

$$\|\mathbf{e}^{(k+1)}\| \leq \rho(B)\|\mathbf{e}^{(k)}\|, \qquad \forall k \geq 0.$$

Iterando a ritroso la disuguaglianza, possiamo scrivere

$$\|\mathbf{e}^{(k)}\| \leq [\rho(B)]^k \, \|\mathbf{e}^{(0)}\|, \qquad k \geq 0$$

grazie alla quale possiamo stimare il numero di iterazioni minimo k_{min} necessario per abbattere l'errore iniziale di un fattore ε:

$$k_{min} \simeq \log(\varepsilon)/\log\left(\rho(B)\right).$$

Attraverso i comandi Matlab® :

```
stima_k_j = log(toll)/log(rho_j)
  stima_k_j =
             56.2575
stima_k_gs = log(toll)/log(rho_gs)
  stima_k_gs =
              10.0290
```

si ottengono le stime corrispondenti al numero di iterazioni necessarie per l'abbattimento dell'errore.

Data la relativa lunghezza della soluzione proposta (un centinaio di righe di codice Matlab®) e la ripetitività di molti comandi, si consiglia di scrivere la soluzione in uno script (*M-file*) tramite un editor.

Esercizio 2.1

1. Utilizzando uno script si creino la matrice A (con $n = 50$), il termine noto $\mathbf{b}$ ed il vettore soluzione iniziale $\mathbf{x}_0$.

```
n    = 50;
A    = diag(4*ones(n,1))     ...
     + diag(-ones(n-1,1),1)  ...
     + diag(-ones(n-2,1),2)  ...
     + diag(-ones(n-1,1),-1) ...
     + diag(-ones(n-2,1),-2);
x0   = zeros(n,1);
b    = 0.2*ones(n,1);
tol  = 1e-5;
nmax = 1e4;
```

2.
```
disp('Matrice A:')
if (A == A')
   v = eig(A);
   if (v > 0)
      disp('Matrice simmetrica definita positiva')
   else
      disp('Matrice simmetrica ma non definita positiva')
   end
else
   disp('Matrice non simmetrica')
end

disp('Numero di condizionamento della matrice:')
max(v)/min(v)
```

Matlab® ritornerà a schermo:

```
Matrice A:
Matrice simmetrica definita positiva
```

```
Numero di condizionamento della matrice:

ans =

  336.2412
```

3. Si riporta la funzione richiesta:

```
function [x, k] = richardson(A, b, P, x0, tol, nmax, alpha)
%
% Metodo di Richardson stazionario precondizionato
%               o dinamico precondizionato (gradiente precondizionato)
%
% Parametri di ingresso:
%
% A: matrice di sistema
% b: vettore termine noto
% P: precondizionatore
% x: guess iniziale
% tol:  tolleranza criterio d'arresto
% nmax: numero massimo di iterazioni ammesse
% alpha: parametro di accelerazione; se non assegnato si considera
%        il metodo dinamico (gradiente precondizionato)
%
% Parametri di uscita:
%
% x: soluzione
% k: numero di iterazioni
%

n = length(b);
if ((size(A,1) ~= n) || (size(A,2) ~= n) || (length(x0) ~= n))
  error('Dimensioni incompatibili')
end


x = x0;
k = 0;
r    = b - A * x;
res_normalizzato  = norm(r) / norm(b);

while ((res_normalizzato > tol) && (k < nmax))
     z = P \ r;
     if (nargin == 6)
         alpha = (z' * r) / (z' * A * z); % alpha dinamico
     end
     x = x + alpha * z;
     r = b - A * x; % equivalente a: r = r - alpha * A * z;
     res_normalizzato  = norm(r) / norm(b);
     k = k + 1;
end

if (res_normalizzato < tol)
     fprintf('Richardson converge in %d iterazioni \n', k);
else
     fprintf('Richardson non converge in %d iterazioni. \n', k)
```

```
end
```

4. Consideriamo i seguenti comandi Matlab®

```
P = eye(n); % Precondizionatore

disp('Richardson: P = I, alpha = 0.20')
alpha     = 0.2;
B_alpha   = eye(n) - alpha * A; % inv(P)=I
disp('Raggio spettrale:')
max(abs(eig(B_alpha)))
[x02, k02] = richardson(A, b, P, x0, tol, nmax, alpha);


disp('Richardson: P = I, alpha = 0.33')
alpha     = 0.33;
B_alpha   = eye(n) - alpha * A; % inv(P)=I
disp('Raggio spettrale:')
max(abs(eig(B_alpha)))
[x033, k033] = richardson(A, b, P, x0, tol, nmax, alpha);


disp('Richardson: P = I, alpha = OPT')
alpha     = 2/(max(v)+min(v));
B_alpha   = eye(n) - alpha * A; % inv(P)=I
disp('Raggio spettrale:')
max(abs(eig(B_alpha)))
[xopt, kopt] = richardson(A, b, P, x0, tol, nmax, alpha);

fprintf('\nRichardson non precond. dinamico\n')
[xdin, kdin] = richardson(A, b, P, x0, tol, nmax);
```

Dai risultati si può dedurre che:

- se `P = I` e `alpha = 0.20` il raggio spettrale della matrice di iterazione è minore di uno (0.9963) ed il metodo converge dopo 3074 iterazioni;
- se `P = I` e `alpha = 0.33` il raggio spettrale della matrice di iterazione è maggiore di uno (1.0581) ed il metodo non converge;
- se `P = I` e `alpha = 2/(max(v)+min(v))` (valore ottimo) il raggio spettrale della matrice di iterazione è 0.9941, leggermente inferiore a quello del primo caso ed il numero di iterazioni richieste scende a 1921.
- se `P = I` e `alpha` varia ad ogni iterazione, il metodo *dinamico* converge in 1948 iterazioni.

5. Applichiamo i seguenti comandi Matlab®

```
P = tril(A); % Precondizionatore

% per una formattazione del testo piu' sofisticata proviamo
% ad utilizzare fprintf al posto di disp.

fprintf('\n Richardson: P = tril(A), alpha = 1.00\n')
```

```
alpha        = 1.0;
B_alpha      = eye(n) - alpha * inv(P) * A;
fprintf('Raggio spettrale: %f\n', max(abs(eig(B_alpha))))
[x_ri, k_ri] = richardson(A, b, P, x0, tol, nmax, alpha);
[x_gs, k_gs] = gs(A, b, x0, tol, nmax);
fprintf('Gauss-Seidel converge in %d iterazioni \n', k_gs);
fprintf('Scarto tra le soluzioni: %e\n', max(abs(x_ri-x_gs)))
```

Matlab® stamperà schermo:

```
Richardson: P = tril(A), alpha = 1.00
Raggio spettrale: 0.990750
Richardson converge in 1231 iterazioni
Gauss-Seidel converge in 1231 iterazioni
Scarto tra le soluzioni: 0.000000e+00
```

6. Consideriamo i seguenti comandi Matlab®

```
P = diag(2*ones(n,1))      ...
  + diag(-ones(n-1,1),1)   ...
  + diag(-ones(n-1,1),-1);

fprintf('\nPrecondizionatore P:\n')
if (P == P')
   v = eig(P);
   if (v > 0)
      disp('Matrice simmetrica definita positiva')
   else
      disp('Matrice simmetrica ma non definita positiva')
   end
else
   disp('Matrice non simmetrica')
end

fprintf('\nRichardson: P = TRIDIAG, alpha = OPT\n')
v         = eig(inv(P)*A);
alpha     = 2/(max(v)+min(v));
B_alpha   = eye(n) - alpha * inv(P) * A;
fprintf('Raggio spettrale: %f\n', max(abs(eig(B_alpha))))
fprintf('Numero di condizionamento: %f\n', cond(inv(P)*A))
[x_tri, k_tri] = richardson(A, b, P, x0, tol, nmax, alpha);


fprintf('\nrichardson: P = TRIDIAG, dinamico\n')
[x, k_tridin] = richardson(A, b, P, x0, tol, nmax);
```

Matlab® stamperà a schermo:

```
Precondizionatore P:
Matrice simmetrica definita positiva

Richardson: P = TRIDIAG, alpha = OPT
Raggio spettrale: 0.664839
```

```
Numero di condizionamento: 7.025895
Richardson converge in 30 iterazioni

richardson: P = TRIDIAG, dinamico
Richardson converge in 25 iterazioni
```

L'uso di un opportuno precondizionatore ha abbassato drasticamente sia il numero di condizionamento (da 336 a 7) che il numero di iterazioni richieste (da migliaia a decine).

Esercizio 3.1

1. Per visualizzare le forme quadratiche in $\mathbb{R}^3$ e le linee di livello nel piano (x, y) si usano rispettivamente i comandi `surf` e `contour` (si veda il file `es31.m`); i grafici ottenuti sono riportati in Figura 1. Le linee di livello di entrambe le forme quadratiche sono delle ellissi, e quelle della forma quadratica associata alla matrice A_2 sono più eccentriche, perché gli autovalori di A_2 sono molto diversi fra loro.

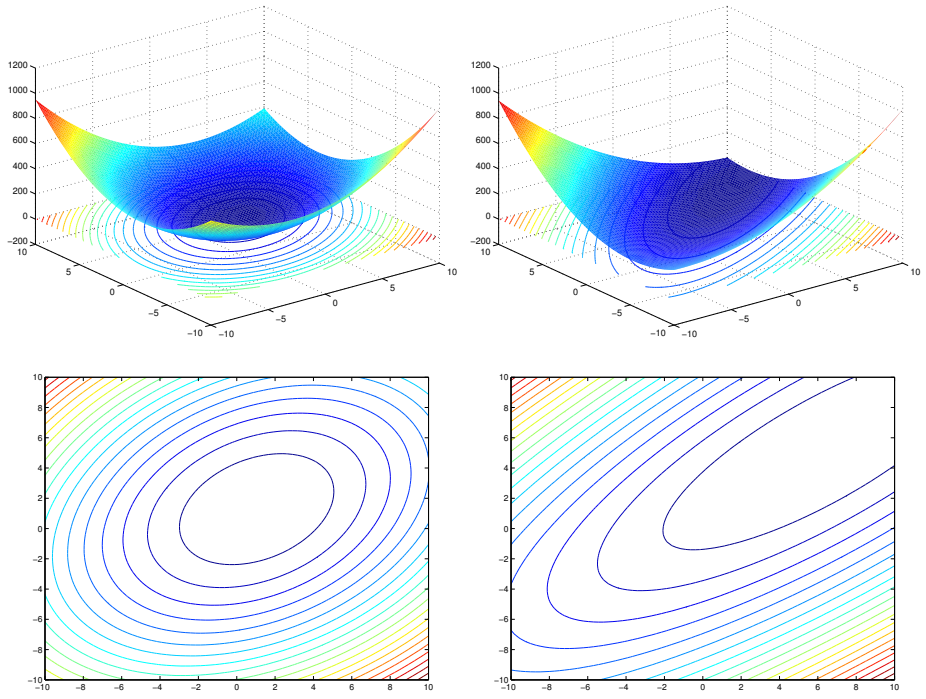

Figura 1: Forme quadratiche (sopra) e linee di livello (sotto): a sinistra φ_1, a destra φ_2

2. La `function` da utilizzare in questo punto è `richardson.m`, con una leggera modifica delle variabili di input e di output, in modo da restituire tutti i valori $\mathbf{x}_k$. Si veda il file `richardson_it.m`.

   Utilizzando come tolleranza 10^{-7}, il metodo di Richardson stazionario con passo $\alpha = 0.05$ converge in 331 iterazioni, mentre con $\alpha = 0.24$ diverge. Dalla Figura 2 è evidente che

la scelta di utilizzare un passo costante per risolvere questo tipo di sistemi può essere estremamente penalizzante anche in caso di convergenza.

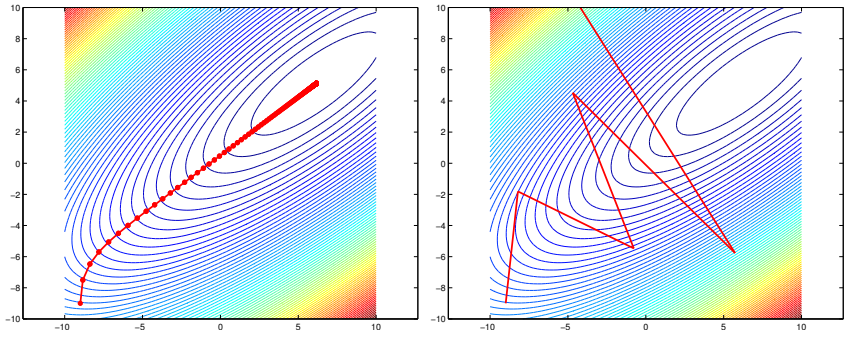

Figura 2: Iterazioni di Richardson: a sinistra il caso convergente, a destra il caso non convergente

Il metodo del gradiente, che adatta il passo ad ogni iterazione, è molto più efficiente e converge in 87 iterazioni. Tuttavia anche la discesa effettuata da questo metodo non è ottimale, come si può vedere in Figura 3, dove è evidente la proprietà del metodo del gradiente di muoversi lungo direzioni perpendicolari.

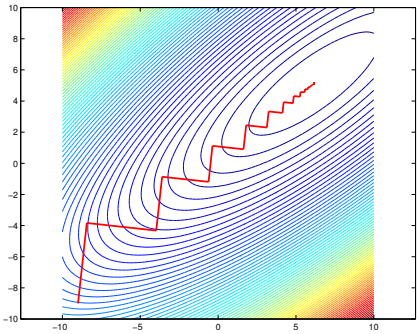

Figura 3: Iterazioni del metodo del gradiente

3. La forma quadratica associata al sistema precondizionato è $\varphi_{prec} = \frac{1}{2}\left(\mathbf{x}^T P^{-1} A\mathbf{x}\right) - \mathbf{x}^T P^{-1}\mathbf{b}$. Con la particolare scelta di P indicata, gli autovettori della matrice $P^{-1}A_2$ sono gli stessi della matrice A_2, ma gli autovalori sono $\lambda_1 = 1, \lambda_2 = 5$, e quindi le linee di livello sono simili a delle circonferenze (vedi Figura 4).

   Applicare il metodo del gradiente precondizionato al sistema lineare originale non è equivalente ad applicare il metodo del gradiente standard al sistema lineare precondizionato $P^{-1}A_2\mathbf{x} = P^{-1}\mathbf{b}$, come si vede in Figura 5. Tuttavia le iterazioni richieste sono simili: 14 nel primo caso e 9 nel secondo.

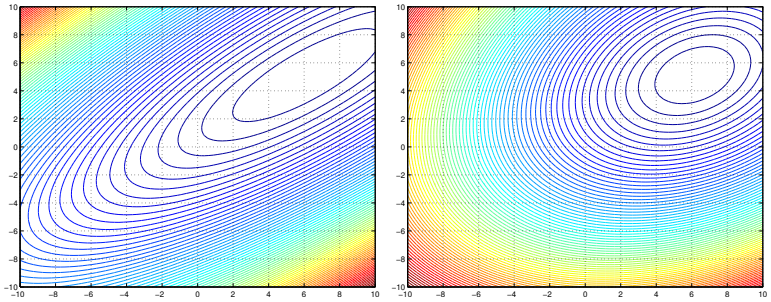

Figura 4: Linee di livello: a sinistra quelle di φ_2, a destra quelle di φ_{prec}

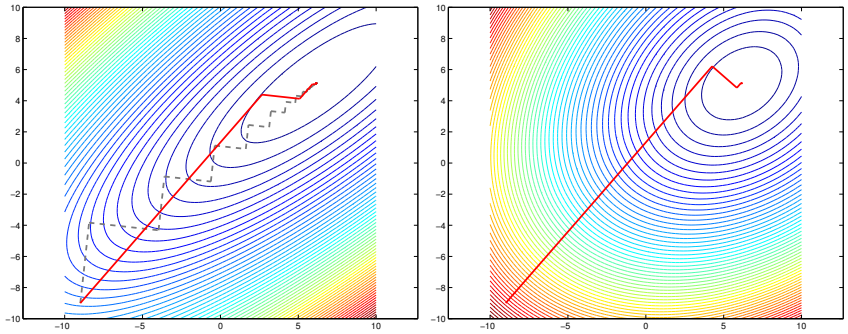

Figura 5: La figura di sinistra riporta le iterazioni del gradiente precondizionato in rosso e quelle del gradiente standard in grigio; quella di destra le iterazioni del gradiente standard applicato al sistema precondizionato.

4. Si veda il file `conjgrad_it.m`.

5. Applicare il metodo del gradiente coniugato consente di mantenere l'ottimalità fra tutte le direzioni, e di convergere alla soluzione esatta in 2 iterazioni. Le direzioni selezionate dal metodo del gradiente coniugato sono riportate in Figura 6.

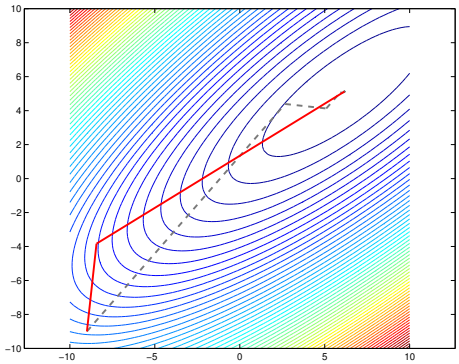

Figura 6: La figura mostra in grigio le iterazioni del gradiente precondizionato e in rosso quelle del gradiente coniugato

Esercizio 3.2

1. Con i seguenti comandi si ottiene la soluzione richiesta. Si osservi che per ripetere tutte le istruzioni necessarie a costruire la matrice A, risolvere il sistema per ogni dimensione è necessario racchiudere tutti i comandi in un ciclo `for` che scandisce la dimensione del sistema.

```
N = [5:20];
Itnp = [];
Itp = [];
for n = N;

    A = diag(4*ones(n,1)) + diag(ones(n-1, 1), 1) + diag(2*ones(n-2, 1), 2) + ...
        diag(ones(n-1, 1), -1) + diag(2*ones(n-2, 1), -2);
    KA = cond(A);
    K = [K KA];
    b = ones(n,1);

    toll = 1e-6;
    nmax = 5*1e3;
    P = tril(A);
    Kprec=[Kprec, cond(inv(P)*A)];

    x0 = zeros(n,1);

    % metodo del gradiente
    [xnp, knp] = richardson(A, b, eye(n), x0, toll, nmax);
    Itnp = [Itnp knp];

    % metodo del gradiente precondizionato
    [xp, kp] = richardson(A, b, P, x0, toll, nmax);
    Itp = [Itp kp];
end
```

   Ad ogni passo del ciclo `for` è necessario salvare il numero di iterazioni `knp` e `kp` fornite dalla function `richardson`, per essere in grado di rappresentarlo graficamente al variare delle dimensioni del sistema.

2. La rappresentazione grafica richiesta si ottiene con i seguenti comandi:

```
figure(1);
semilogy(N, Itnp, 'r', N, Itp, 'b')
grid on
title('metodo del gradiente VS gradiente precondizionato')
xlabel('dimensioni sistema');
ylabel('iterazioni')
legend('gradiente', 'gradiente precondizionato', 'Location', 'Northwest')
```

   Il grafico ottenuto è rappresentato in Figura 7.

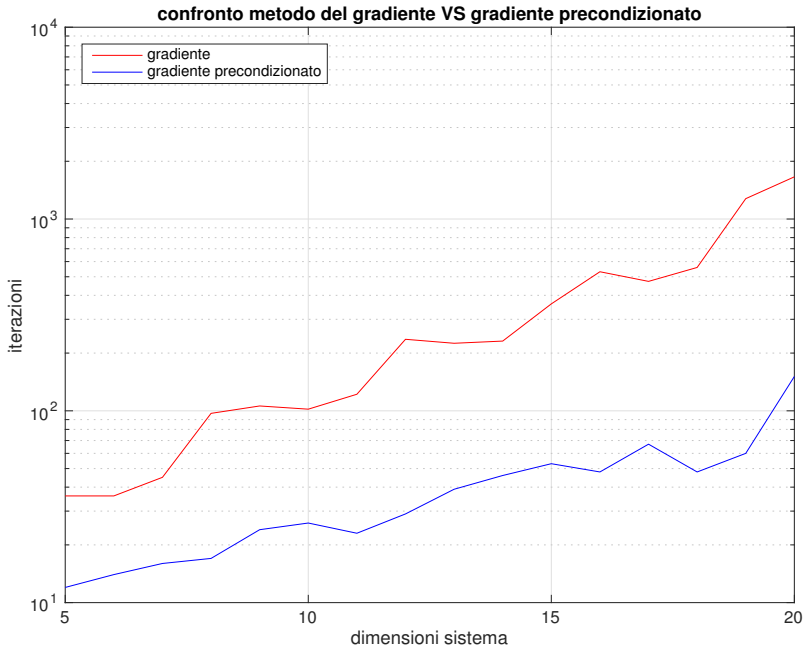

Figura 7: Confronto grafico tra il numero di iterazioni impiegate dal metodo del gradiente e del gradiente precondizionato per risolvere il sitema lineare $A\mathbf{x} = \mathbf{b}$ al variare delle dimensioni del sistema.

3. Il calcolo del numero di condizionamento della matrice al variare delle dimensioni del sistema si ottiene includendo nel precedente listato le seguenti istruzioni:

```
K =[];
for n = N;
 ...
 KA = cond(A);
 K = [K KA];
 ...
end
```

Il grafico si ottiene con

```
figure(2);
plot(N, K, 'r')
grid on
title('Numero di condizionamento di A')
xlabel('dimensione sistema');
ylabel('cond(A)')
```

Il grafico ottenuto è mostrato in Figura 8.

Con comandi del tutto analoghi si può calcolare il condizionamento della matrice precondizionata $P^{-1}A$ al variare di n; il risultato è riportato in Figura 9. Si osserva che il

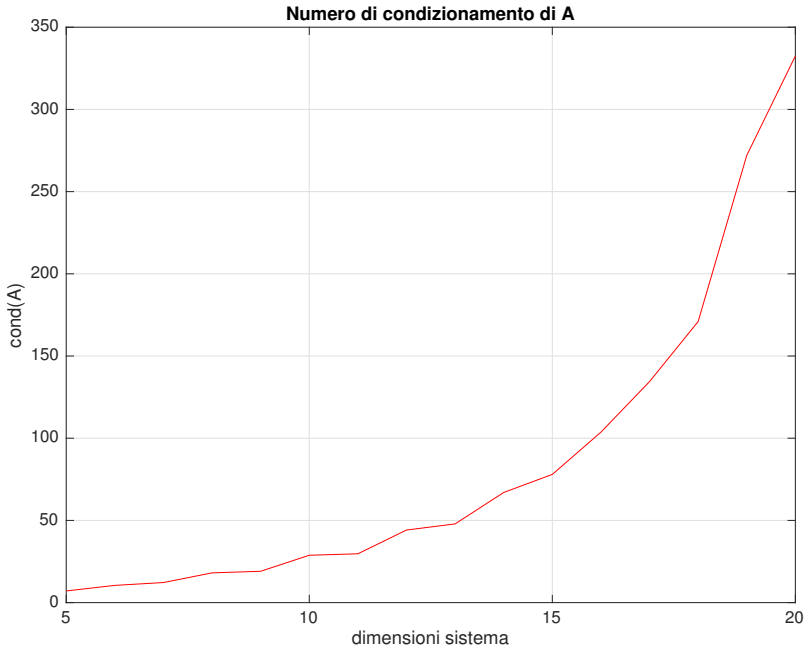

Figura 8: Rappresentazione grafica del numero di condizionamento di A al variare delle dimensioni del sistema.

precondizionamento è tale da rendere il numero di condizionamento pari ad un terzo del valore del caso precedente.

4. Analogamente a quanto fatto nella prima parte dell'esercizio, la soluzione si ottiene con i comandi:

```
N = [5:20];
Itcg = [];

for n = N;

    A = diag(4*ones(n,1)) + diag(ones(n-1, 1), 1) + diag(2*ones(n-2, 1), 2) + ...
        diag(ones(n-1, 1), -1) + diag(2*ones(n-2, 1), -2);

    b = ones(n,1);

    toll = 1e-6;
    nmax = 5000;
    x0 = ones(n,1);

    [xkcg, kcg] = conjgrad_it(A, b, x0,nmax,toll);
    Itcg = [Itcg kcg];
end
```

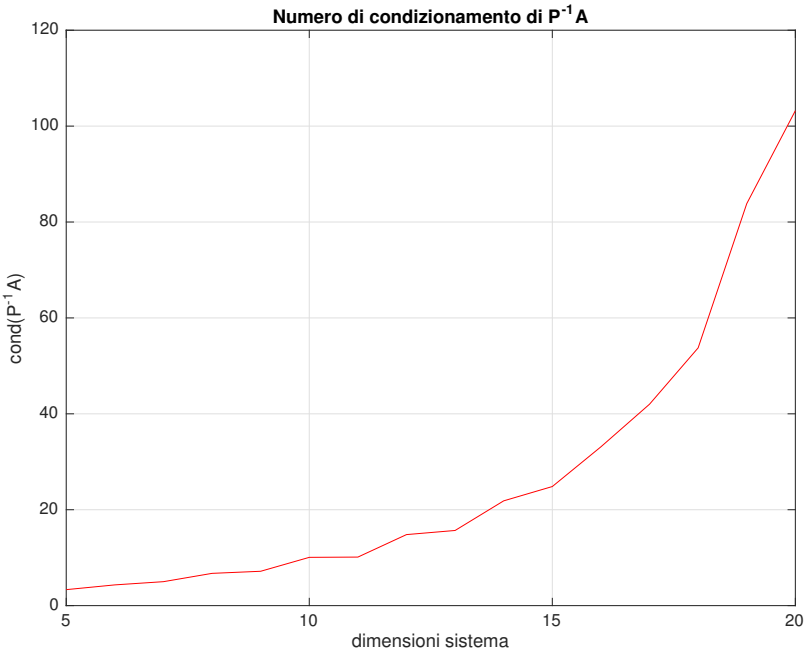

Figura 9: Rappresentazione grafica del numero di condizionamento di $P^{-1}A$ al variare delle dimensioni del sistema.

5. I tre grafici che permettono di confrontare le iterazioni dei vari metodi sono ottenuti con i comandi

```
figure(3);
semilogy( N, Itnp, 'r', N, Itp, 'b',N, Itcg, 'g-')
grid on
title('metodo del gradiente coniugato ...
VS gradiente VS gradiente precondizionato')
xlabel('dimensioni sistema');
ylabel('iterazioni')
legend( 'gradiente','gradiente precondizionato', ...
'gradiente coniugato', 'Location', 'Northwest')
```

Il grafico risultante è mostrato in Figura 10.

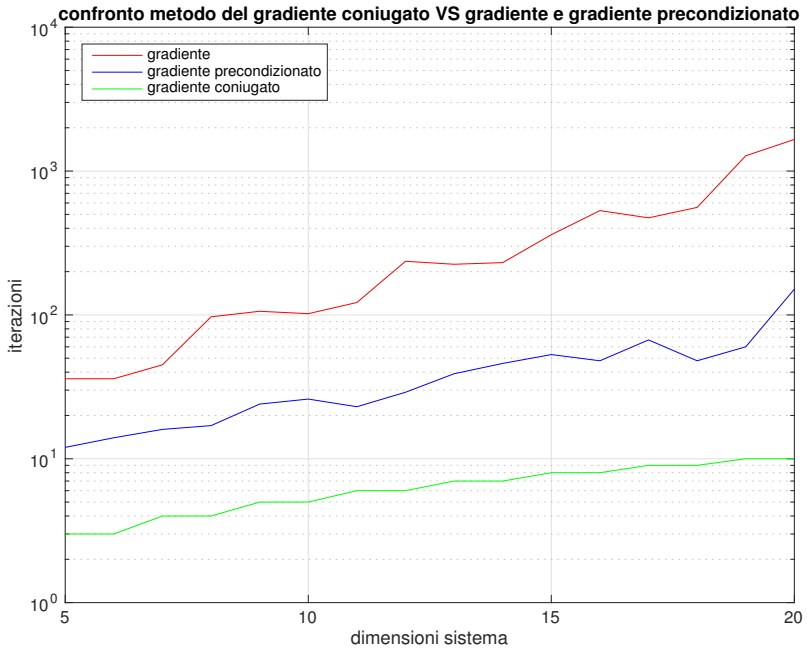

Figura 10: Confronto grafico tra il numero di iterazioni impiegate dal metodo del gradiente, del gradiente precondizionato e del gradiente coniugato per risolvere il sitema lineare $A\mathbf{x} = \mathbf{b}$ al variare delle dimensioni del sistema.

Esercizio 4.1

La matrice A, il termine noto $\mathbf{b}$ e il vettore $\mathbf{x}$ possono essere costruiti tramite i seguenti comandi:

```
n = 300;
A = diag( 6 * ones( n, 1 ) ) + diag( 1 * ones( n - 2, 1 ), -2 ) ...
    + diag( 1 * ones( n - 2, 1 ), 2 ) + diag( -2 * ones( n - 1, 1 ), -1 ) ...
    + diag( -2 * ones( n - 1, 1 ), 1 );
x = ones( n, 1 );
b = A * x;
```

1. La condizione necessaria e sufficiente per la convergenza dei metodi di Jacobi e Gauss–Seidel per ogni $\mathbf{x}^{(0)} \in \mathbb{R}^n$ è che il raggio spettrale delle corrispondenti matrici di iterazioni sia strettamente inferiore ad uno.

   La matrice di iterazione e il raggio spettrale si calcolano a partire dalle definizioni:

   $$B = I - P^{-1}A,$$

   $$\rho(B) = \max_{i=1,\dots,n} |\lambda_i(B)|,$$

   dove $P = D$, la matrice diagonale estratta da A, per il metodo di Jacobi e $P = D - E$, la matrice triangolare inferiore estratta da A, per il metodo di Gauss-Seidel.

Queste quantità possono essere calcolate in Matlab® attraverso i seguenti comandi:

```
D = diag( diag( A ) );
BJ = eye( n ) - D \ A;
rhoBJ = max( abs( eig( BJ ) ) );
    rhoBJ =
        0.9999

BGS = eye( n ) - tril( A ) \ A;
rhoGS = max( abs( eig( BGS ) ) );
    rhoGS =
        0.3535
```

Dal calcolo del raggio spettrale delle matrici si può concludere che sia il metodo di Jacobi che quello di Gauss-Seidel convergono, in quanto gli autovalori massimi risultano in modulo strettamente minori di 1.

2. Dalla teoria sappiamo che un raggio spettrale in modulo più basso accelera la convergenza del metodo. In particolare,

$$\rho(B_{GS}) < \rho(B_J),$$

quindi la convergenza del metodo di Gauss-Seidel sarà più rapida e quantificata da

$$R_{GS} = -\log(\rho(B_{GS})) > R_J = -\log(\rho(B_J)).$$

In conclusione, il metodo di Gauss-Seidel richiederà un numero di iterazioni inferiore, per un dato livello di accuratezza, rispetto al metodo di Jacobi.

3. Per risolvere il sistema è necessario richiamare la funzione Matlab® `gs.m`, dopo aver definito tutti i parametri di ingresso richiesti:

```
toll = 1e-6;
nmax = 1000;
x0 = b;
[ xk, k ] = gs( A, b, x0, toll, nmax );
```

In questo modo è possibile stampare il numero di iterazioni k effettuate, il valore dell'errore relativo $e_{rel}^{(N_{it})}$ ed il valore del residuo normalizzato $r_{norm}^{(N_{it})}$ corrispondente:

```
k =
    13
res_norm = norm( b - A * xk ) / norm( b )
    7.3717e-07
err_rel = norm( x - xk ) / norm( x )
    6.1157e-07
```

Infine, sfruttando la definizione della stima a posteriori dell'errore, possiamo calcolare una stima dell'errore ottenuto a partire dal residuo normalizzato $r_{norm}^{(N_{it})}$:

```
err_stim = cond(A) * res_norm
    2.9482e-06
```

Si verifica quindi che l'errore relativo ottenuto è effettivamente inferiore all'errore stimato tramite la stima a posteriori dell'errore.

4. Sfruttando la teoria ed in particolare la stima dell'errore

$$\|\mathbf{e}^{(k)}\|_A \leq \left(\frac{K(A)-1}{K(A)+1}\right)^k \|\mathbf{e}^{(0)}\|_A, \qquad k \geq 0,$$

è possibile stimare l'errore in norma A ($\mathbf{v} \in \mathbb{R}^n$, $||\mathbf{v}||_A = \sqrt{\mathbf{v}^T A \mathbf{v}}$) dopo 20 iterazioni del metodo con i seguenti comandi:

```
d = ( cond(A) - 1 ) / ( cond( A ) + 1 );
err_g_k = d^20 * sqrt( ( x - x0 )' * A * ( x - x0 ) )
    0.0038
```

5. Il valore di β, per il quale il metodo del gradiente precondizionato converge più rapidamente alla soluzione per ogni scelta del dato iniziale, è quello che minimizza il fattore di abbattimento dell'errore per ogni iterazione

$$d = \frac{K(P^{-1}A)-1}{K(P^{-1}A)+1},$$

con

$$K(P^{-1}A) = \frac{\lambda_{max}(P^{-1}A)}{\lambda_{min}(P^{-1}A)}.$$

Infatti, per il metodo del gradiente precondizionato, vale la seguente stima dell'errore:

$$\|\mathbf{e}^{(k)}\|_A \leq d^k \|\mathbf{e}^{(0)}\|_A, \qquad k \geq 0.$$

Tale valore d può essere calcolato e visualizzato attraverso i seguenti comandi:

```
betav = 2 : 1: 5;
dv = [];
for beta = betav
    P = diag( beta * ones( n, 1 ) ) ...
        + diag( - 1 * ones( n - 1, 1 ), 1 ) ...
        + diag( - 1 * ones( n - 1, 1 ), -1 );
    cPA = max(eig(P\A))/min(eig(P\A));
    dPA = ( cPA - 1 ) / ( cPA + 1 );
    dv = [ dv, dPA ];
end
plot( betav, dv )
```

Si nota il risultato riportato in Figura 11, da cui si evince che il valore minimo di d corrisponde a $\beta = 4$. Ovvero, la convergenza più rapida, per ogni scelta di $\mathbf{x}^{(0)}$, si ottiene per $\beta = 4$.

6. Il problema può essere risolto attraverso il metodo del gradiente coniugato attraverso la chiamata alla funzione Matlab® `pcg`:

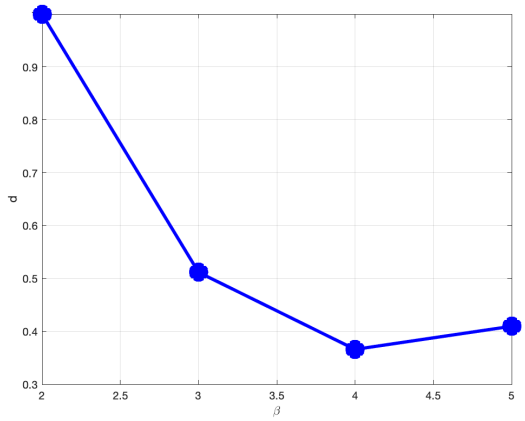

Figura 11: Valore di d vs. β.

```
xpcg = pcg(A,b,1e-6,1000,[],[],b);
    err_g_k =
        0.0038
    pcg converged at iteration 12 to a solution with relative residual 8e-07
```

tramite la quale viene stampato il numero di iterazioni N_{it} effettuate dal metodo.

Infine, l'errore $\|\mathbf{x}^{(N_{it})} - \mathbf{x}\|_A$ corrispondente può essere calcolato utilizzando la seguente istruzione:

```
err = sqrt( (xpcg-x)'*A*(xpcg-x))
    2.2694e-05
```