Algoritmo di Mimimizzazione della Traccia per il Problema agli Autovalori Generalizzato

Codici sorgente e sperimentazioni al calcolatore

Giuseppe Lombardi

November 26, 2018

Indice

1 Problema

In questo lavoro verrà presentato un metodo iterativo che ci permetterà di ottenere buone approssimazioni su alcuni dei più piccoli autovalori e corrispettivi autovettori del *problema generalizzato agli autovalori*:

$$Ax = \lambda Bx \tag{1}$$

Dove:

- x vettore di lunghezza n
- λ uno scalare
- A e B matrici simmetriche $n \times n$, con B definita positiva
- solitamente A e B sono molto grandi e sparse.

Definizione 1. *Una matrice Y $n \times p$ forma una* ***sezione*** *del problema agli autovalori $Ax = \lambda Bx$ se valgono:*

$$Y^t AY = \Sigma \qquad e \qquad Y^t BY = I_p$$

con $\Sigma = diag(\sigma_1, ..., \sigma_p)$.

Il *metodo di minimizzazione della traccia* calcola i p piú piccoli autovalori e i corrispettivi autovettori sfruttando la proprietà di riduzione della traccia. Il nostro approccio è quello di trovare una sequenza di iterazioni $Y_{k+1} = F(Y_k)$ tale che:

- Y_k e Y_{k+1} sono una *sezione* del problema
- $tr(Y_{k+1}^t AY_{k+1}) < tr(Y_k^t AY_k)$
- $F(Y_k)$ è scelto in modo tale che la convergenza globale del processo sia assicurata.

Trattiamo il problema come un problema di minimizzazione quadratica

$$\text{Minimizza } tr(Y^t AY) \quad \text{soggetto ai vincoli } Y^t BY = I_p.$$

2 Algoritmo di base

Data Y_k una matrice $n \times p$ che approssima i p autovettori corrispondenti ai p più piccoli autovalori del problema cioè tale che:

$$Y_k^t A Y_k = \Sigma_k = diag(\sigma_1^{(k)}, ..., \sigma_p^{(k)})$$

$$Y_k^t B Y_k = I_p$$

vogliamo costruire una matrice Y_{k+1} della forma

$$Y_{k+1} = (Y_k + \Delta_k) S_k$$

dove S_k e Δ_k sono scelte in modo che:

- $Y_{k+1}^t A Y_{k+1} = \Sigma_{k+1} = diag(\sigma_1^{(k+1)}, ..., \sigma_p^{(k+1)})$
- $Y_{k+1}^t B Y_{k+1} = I_p$
- $tr(Y_{k+1}^t A Y_{k+1}) < tr(Y_k^t A Y_k)$

3 Codici sorgente ausiliari

```
function [A, B, condS] = generaAB(n, distribuzione)
%input:
%       n               dimensione delle due matrici quadrate da generare
%       distribuzione   distribuzione degli autovalori
%output:
%       A, B            matrici random, dense, simmetriche, definite positive
%                       tali che il problema generalizzato agli autovalori
%                       A * x = lambda * B * x, ha autovalori distribuzione
%       condS           numero di condizionamento della matrice random S

%genera matrice densa nxn
S = rand(n);
%fattorizzazione QR di S, Q e' una matrice ortogonale
[Q, R] = qr(S);
%A e' una matrice con autovalori distribuzione
A = Q * diag(distribuzione) * Q';
A = S * A * S';
B = S * S';
condS = cond(S);
endfunction

%Gli autovalori di A distribuzione sono anche autovalori per il problema
%generalizzato poiche'
%B^(-1)*A=(S')^(-1)*S^(-1)*S*A*S'=(S')^(-1)*A*(S)^(-1)
%quindi B^(-1)*A e' simile ad A
```

```
function Y = creasezione(A, B, p)
%input:
%       A, B  matrici dense, simmetriche e definite positive nxn
%       p     numero di autovalori che vogliamo approssimare
%output:
%       Y     sezione per il problema A*x = lambda*B*x, che sara' usata
%             nel passo iniziale

n = size(A)(1);
%estraggo sottomatrice di testa pxp da A
A = A(1 : p, 1 : p);
%estraggo sottomatrice di testa pxp da B
B = B(1 : p, 1 : p);
%calcolo matrice di autovettori X e di autovalori D relative alla
%matrice B^(-0.5)*A*B^(-0.5)
[X, D] = eig( B ^ (-0.5) * A * B ^ (-0.5) );
Y = B ^ (-0.5) * X;
%creo matrice che ha la prima parte uguale a Y e poi tutti zero
Y = [Y; zeros(n - p, p)];
endfunction
```

```
function [A, B] = spgeneraAB(n, d)
%input:
%       n     dimensione delle due matrici quadrate da generare
%       d     densita' della matrice sparsa (percentuale di non zeri)
%output:
%       A, B  matrici random, sparse, simmetriche, definite positive
%             per il problema generalizzato agli autovalori
%             A * x = lambda * B * x.

%genera matrice sparsa nxn con densita' d in notazione compressa
A = sprand(n, n, d);
B = sprand(n, n, d);
%rendo le matrici A e B simmetriche
A = A + A';
B = B + B';
alfa = norm(A, inf);
beta = norm(B, inf);
%rendo gli autovalori di A e di B tutti positivi
A = A + alfa * speye(n);
B = B + beta * speye(n);
endfunction



%data una matrice A la successione ||A^k||^(1/k) e' decrescente e
%converge dall' alto al
%raggio spettrale di A (autovalore di modulo massimo di A)
```

```
function Y = spcreasezione(A, B, p)
%input:
%       A, B  matrici simmetriche definite positive, sparse nxn
%       p     numero di autovalori che vogliamo approssimare
%output:
%       Y     sezione per il problema A*x = lambda*B*x, che sara' usata
%             nel passo iniziale

n = size(A)(1);
%estraggo sottomatrice di testa pxp da A
A = A(1 : p, 1 : p);
%estraggo sottomatrice di testa pxp da B
B = B(1 : p, 1 : p);
%calcolo matrice di autovettori X e di autovalori D relative alla
%matrice B^(-0.5)*A*B^(-0.5)
[X, D] = eig( full(B) ^ (-0.5) * full(A) * full(B) ^ (-0.5) );
Y = B ^ (-0.5) * X;
%creo matrice che ha la prima parte uguale a Y e poi tutti zero
Y = [Y; zeros(n - p, p)];
endfunction
```

4 Implementazione del metodo

Ci sono due tipi di implementazioni del *metodo di minimizzazione della traccia*, che si distinguono su come è costruita la matrice Δ_k.

4.1 Δ^k tramite passo di discesa rapido

```
function [Yf, sigma, err, tr, iter] = tracmn1(A, B, Y, p, tol, maxit)
%input:
%       A, B  matrici simmetriche definite positive
%       Y     matrice nxp di rango p, sezione per il problema
%             A*x = lambda*B*X
%       p     numero di autovalori che vogliamo approssimare
%       tol   tolleranza per il criterio d' arresto
%       maxit massimo numero di iterazioni
%output:
%       Yf    matrice le cui colonne sono le approssimazioni degli
%             autovettori relativi agli autovalori trovati
%       sigma vettore le cui componenti sono le approssimazioni degli
%             autovalori
%       err   vettore la cui componente k e' il residuo relativo
%             || A * Y_k - B * Y_k * SIGMA_k ||
%       tr    vettore la cui componente k indica la traccia della
%             matrice diagonale SIGMA_k (ovvero ci da' la somma
%             delle approssimazioni k-esime degli autovalori)
%       iter  numero di iterazioni eseguite

%stabiliamo un numero massimo di iterazioni

n = size(A)(1);
termina = false;
%SIGMA_1 matrice diagonale pxp che approssima gli autovalori
SIGMA = Y' * A * Y;
%calcolo il residuo relativo al primo passo
err(1) = norm(A * Y - B * Y * SIGMA, 'inf');
%calcolo la traccia di SIGMA_1
tr(1) = trace(SIGMA);
k = 2;
%blocco che calcola ad ogni iterazione la sezione Y_k+1
while !termina && k < maxit
        %P_k e' la proiezione del sottospazio generato dalle colonne
        %di B*Y_k
        P = B * Y * inv(Y' * B * B * Y) * Y' * B;
        C = Y' * A * (eye(n) - P) * A * Y;
        E = Y' * A * (eye(n) - P) * A * (eye(n) - P) * A * Y;
```

```
        %costruisco W_k matrice diagonale di ordine p
        W = zeros(p);
        for i = 1 : p
                if E(i, i) != 0
                        W(i, i) = - C(i, i) / E(i, i);
                endif
        endfor
        Z = A * Y * W;
        %calcolo DELTA_k
        DELTA = (eye(n) - P) * Z;
        %calcolo Y cappuccio
        Ycap = Y + DELTA;
        %decomposizione spettrale di Ycap'*B*Ycap matrice definita
        %positiva, D matrice diagonale pxp e U matrice ortogonale
        [U, D] = eig(Ycap' * B * Ycap);
        D = D .^ 0.5;
        %decomposizione spettrale di D^(-1)*U'*Ycap'*A*Ycap*U*D^(-1),
        %SIGMA matrice diagonale pxp e V matrice ortogonale
        [V, SIGMA] = eig(inv(D)* U' * Ycap' * A * Ycap * U * inv(D));
        %calcolo S_k
        S = U * inv(D) * V;
        %calcolo Y_k+1, DELTA_k e S_k sono stati scelti in modo che
        %anche Y_k+1 sia anch' essa una sezione del problema
        %e in modo che la traccia sia strettamente decrescente
        Y = Ycap * S;
        %calcolo il residuo relativo al passo k-esimo
        err(k) = norm(A * Y - B * Y * SIGMA, 'inf');
        %calcolo la traccia di SIGMA_k
        tr(k) = trace(SIGMA);
        %se il miglioramento dell' approssimazione e' trascurabile
        %possiamo interrompere il ciclo
        if err(k) <= tol * err(1)
                termina = true;
        endif
        k = k + 1;
endwhile
iter = k - 1;
%creo il vettore con le approssimazioni degli autovalori
sigma = zeros(1,p);
for i = 1 : p
        sigma(i) = SIGMA(i, i);
endfor
Yf = Y;
endfunction
```

4.2 Δ^k come iterata sottospazio ottimale

```
function [Yf, sigma, err, tr, iter] = tracmn2(A, B, Y, p, tol,...
gamma, maxit)
%input:
%       A, B    matrici simmetriche definite positive
%       Y       matrice nxp di rango p, sezione per il problema
%               A*x = lambda*B*X
%       p       numero di autovalori che vogliamo approssimare
%       tol     tolleranza per il criterio di arresto
%       gamma   scelta del criterio di arresto per il metodo del
%               gradiente coniugato (CG), (gamma < 1)
%       maxit   massimo numero di iterazioni
%output:
%       Yf      matrice le cui colonne sono le approssimazioni degli
%               autovettori relativi agli autovalori trovati
%       sigma   vettore le cui componenti sono le approssimazioni
%               degli autovalori
%       err     vettore la cui componente k e' il residuo relativo
%               || A * Y_k - B * Y_k * SIGMA_k ||
%       tr      vettore la cui componente k indica la traccia della
%               matrice diagonale SIGMA_k (ovvero ci da' la somma
%               delle approssimazioni k-esime degli autovalori)
%       iter    numero di iterazioni eseguite

n = size(A)(1);
%calcola SIGMA_1
SIGMA = Y' * A * Y;
%calcolo il residuo relativo al primo passo
err(1) = norm(A * Y - B * Y * SIGMA, 'inf');
%calcolo la traccia di SIGMA_1
tr(1) = trace(SIGMA);
i = 1;
termina = false;
%blocco che calcola ad ogni iterazione la sezione Y_i+1
while !termina && i < maxit
        %calcolo della fattorizzazione QR di B*Y (B*Y ha rango p)
        [Q, R] = qr(B * Y);
        %Q2 matrice nxn-p
        Q2 = Q(:, p + 1 : n);
        %inizializzazione della correzione i-esima DELTA_i
        %(matrice nxp)
        DELTA = zeros(n, p);
        %risoluzione dei p sistemi per calcolare le p colonne di
        %DELTA_i, tramite CG
```

```
for j = 1 : p
        %colonna j-esima di Y
        y = Y(:, j);
        %primo passo del CG per la soluzione del j-esimo
        %sistema, g_0 vinene inizializzato a 0
        g = zeros(n - p, 1);
        %calcolo r_0, residuo relativo, anche esso di
        %lunghezza n-p
        r = (Q2' * A * y) - (Q2' * A * Q2 * g);
        %calcola q_0 direzione di decrescita
        q = r;
        %calcola il residuo iniziale in norma 2
        residuoStart = norm(r, 2);
        k = 1;
        terminaCG = false;
        %calcola ad ogni iterazione g_k+1 tramite il metodo CG
        while !terminaCG
                %scelta ottimale per a_k
                a = (r' * r) / (q' * Q2' * A * Q2 * q);
                g = g + (a * q);
                %calcola r_k+1
                r1 = r - (a * Q2' * A * Q2 *q);
                b = (r1' * r1) / (r' * r);
                %calcola il nuovo q (q_k+1)
                q = r1 + b * q;
                %aggiorna il residuo relativo
                r = r1;
                %calcola il nuovo residuo in norma 2
                residuo = norm(r, 2);
                %se il residuo (in norma 2) si e' ridotto di un
                %fattore gamma terminiamo
                if residuo <= gamma * residuoStart
                        terminaCG = true;
                endif
                k = k + 1;
        endwhile
        %colonna j-esima della correzione i-esima DELTA_i
        DELTA(:, j) = Q2 * g;
endfor
Ycap = Y - DELTA;
%decomposizione spettrale di Ycap'*B*Ycap matrice definita
%positiva, D matrice diagonale pxp e U matrice ortogonale
[U, D] = eig(Ycap' * B * Ycap);
D = D .^ 0.5;
%decomposizione spettrale di D^(-1)*U'*Ycap'*A*Ycap*U*D^(-1),
```

```
        %SIGMA matrice diagonale pxp e V matrice ortogonale
        [V, SIGMA] = eig(inv(D)* U' * Ycap' * A * Ycap * U * inv(D));
        %calcolo S_i
        S = U * inv(D) * V;
        %calcolo Y_i+1, DELTA_i e S_i sono stati scelti in modo che
        %anche Y_i+1 sia anch' essa una sezione del problema
        %e in modo che la traccia sia strettamente decrescente
        Y = Ycap * S;
        %calcolo il residuo relativo k-esimo
        err(i) = norm(A * Y - B * Y * SIGMA, 'inf');
        %calcolo la traccia di SIGMA_k
        tr(i) = trace(SIGMA);
        %se il miglioramento dell' approssimazione e' trascurabile
        %possiamo interrompere il ciclo
        if err(i) <= tol * err(1)
                termina = true;
        endif
        i = i + 1;
endwhile
iter = i - 1;
%creo il vettore con le approssimazioni degli autovalori
sigma = zeros(1, p);
for i = 1 : p
        sigma(i) = SIGMA(i, i);
endfor
Yf = Y;
endfunction
```

4.2.1 Aspetti computazionali: Shifting

L' algoritmo di minimizzazione della traccia, dove Δ_k è scelto tramite iterata sottospazio minimale, può essere migliorato tramite una tecnica di *shifting*.
Il metodo di minimizzazione consiste nel risolvere

$$(A - \sigma_j^{(k)} B)x_j = \bar{\lambda}_j^{(k)} B x_j \qquad \text{per } j = 1, ..., p.$$

Questi problemi:

- hanno gli stessi autovettori del problema $Ax = \lambda Bx$
- hanno autovalori $\bar{\lambda}_j^{(k)} = \lambda_j - \sigma_j^{(k)}$ dove λ_j è il j-esimo autovalore del problema e $\sigma_j^{(k)}$ è il parametro di spostamento, approssimazione dell' autovalore λ_j al passo k.

Osserviamo che, ai fini dell' euristica:

- l' efficenza dell' algoritmo dipende in modo cruciale dalla strategia dello *shifting* impiegata; se spostiamo la colonna j attraverso σ_j troppo tardi l' algoritmo diventa inefficiente nel senso che prendiamo alcuni passi a un tasso di convergenza più lento (lineare) quando invece è possibile un tasso di convergenza cubico. Dall' altra parte se spostiamo troppo presto, la funzione obiettivo con la A aumenta invece di diminuire e la convergenza globale è persa. Dunque il passo **??** della strategia cerca di prevenire tale perdita di convergenza globale mentre il passo **??** evita inutili ritardi di spostamento;
- questa strategia risulta essere ben bilanciata cioè evita eccessive iterazioni del CG e di calcolare troppe sezioni;
- il metodo blocca una colonna di Y quando raggiunge la convergenza;
- la convergenza è ottenuta quando le coppie (autovalore, autovettore) hanno un residuo relativo piccolo e in tal caso terminiamo il processo di iterazione;
- una colonna di Y è accettata come valida approssimazione di un autovettore se i d_j calcolati con il CG sono al livello della precisione di macchina (eps).

```
function [Yf, sigma, err, tr, iter] = tracmnshifting(A, B, Y, p,...
start, tol, gamma, maxit)
%input:
%       A, B    matrici simmetriche definite positive nxn
%       Y       matrice nxp di rango p, sezione per il problema
%               A*x = lambda*B*X
%       p       numero di autovalori che vogliamo approssimare
%       start   iterazione in cui si applica lo shifting
%       tol     tolleranza per il criterio di arresto
%       gamma   scelta del criterio di arresto per il metodo del
%               gradiente coniugato (CG), (gamma < 1)
%       maxit   massimo numero di iterazioni
%output:
%       Yf      matrice le cui colonne sono le approssimazioni degli
%               autovettori relativi agli autovalori trovati
%       sigma   vettore le cui componenti sono le approssimazioni
%               degli autovalori
%       err     vettore la cui componente k e' il residuo relativo
%               || A * Y_k - B * Y_k * SIGMA_k ||
%       tr      vettore la cui componente k indica la traccia della
%               matrice diagonale SIGMA_k (ovvero ci da' la somma
%               delle approssimazioni k-esime degli autovalori)
%       iter    numero di iterazioni eseguite

n = size(A)(1);
Aold = A;
Ashift = zeros(n, n, p);
for j = 1 : p
        Ashift(:, :, j) = Aold;
endfor
%calcola SIGMA_1 matrice diag pxp
SIGMA = Y' * A * Y;
%calcolo il residuo relativo al primo passo
err(1) = norm(A * Y - B * Y * SIGMA, 'inf');
%calcolo il residuo delle rispettive colonne
colonna(1, :) = max(abs(A * Y - B * Y * SIGMA));
%calcolo la traccia di SIGMA_1
tr(1) = trace(SIGMA);
% bloccata vettore la cui componente j-esima mi indica se la j-esima
%(autovettore, autovalore) ha raggiunto la condizione d' arresto
bloccata = zeros(1, p);
i = 1;
termina = false;
%blocco che calcola ad ogni iterazione la sezione Y_i+1
while !termina && i < maxit
```

```
%calcolo della fattorizzazione QR di B*Y (B*Y ha rango p)
[Q, R] = qr(B * Y);
%Q2 matrice nxn-p
Q2 = Q(:,p + 1 : n);
%inizializzazione della correzione i-esima DELTA_i
%(matrice nxp)
DELTA = zeros(n, p);
%risoluzione dei p sistemi per calcolare le p
%colonne di DELTA_i, tramite CG
for j = 1 : p
        %applico il CG solo per le colonne di Y per cui non
        %vale ancora la condizione d' arresto del residuo
        if bloccata(j) == 0
                %all' iterazione start applichiamo lo shifting
                if i == start
                        if SIGMA(1, 1) < 0
                                for j = 2 : p
                                        SIGMA(j, j) = SIGMA(1, 1);
                                endfor
                        endif
                        %nella componente j del vettore Ashift
                        %memorizzo la matrice A shiftata che sara'
                        %utilizzata per il calcolo della colonna
                        %j-esima di Y
                        Ashift(:, :, j) = Aold - SIGMA(j, j) * B;
                endif
                if i >= start
                        A = Ashift(:, :, j);
                endif
                %colonna j-esima di Y
                y = Y(:, j);
                %primo passo del CG per la soluzione del
                %j-esimo sistema, g_0 vinene inizializzato a 0
                g = zeros(n - p, 1);
                %calcolo r_0, residuo relativo, anche esso di
                %lunghezza n-p
                r = (Q2' * A * y) - (Q2' * A * Q2 * g);
                %calcola q_0 direzione di decrescita
                q = r;
                %calcola il residuo iniziale in norma 2
                residuoStart = norm(r, 2);
                terminaCG = false;
                %calcola ad ogni iterazione g_k+1 tramite il
                %metodo CG
                while !terminaCG
```

```
                %scelta ottimale per a_k
                a = (r' * r) / (q' * Q2' * A * Q2 * q);
                g = g + (a * q);
                %calcola r_k+1
                r1 = r - (a * Q2' * A * Q2 *q);
                b = (r1' * r1) / (r' * r);
                %calcola il nuovo q (q_k+1)
                q = r1 + b * q;
                %aggiorna il residuo relativo
                r = r1;
                %calcola il nuovo residuo in norma 2
                residuo = norm(r, 2);
                %se il residuo (in norma 2) si e'
                %ridotto di un
                %fattore gamma terminiamo
                if residuo <= gamma * residuoStart
                        terminaCG = true;
                endif
            endwhile
            %colonna j-esima della correzione i-esima
            %DELTA_i
            DELTA(:, j) = Q2 * g;
        endif
    endfor
    Ycap = Y - DELTA;
    %decomposizione spettrale di Ycap'*B*Ycap matrice definita
    %positiva, D matrice diagonale p x p e
    %U matrice ortogonale
    [U, D] = eig(Ycap' * B * Ycap);
    D = D .^ 0.5;
    %decomposizione spettrale di D^(-1)*U'*Ycap'*A*Ycap*U*D^(-1),
    %SIGMA matrice diagonale p x p e
    %V matrice ortogonale
    [V, SIGMA] = eig(inv(D) * U' * Ycap' * Aold * Ycap * U * inv(D));
    %calcolo S_i
    S = U * inv(D) * V;
    %calcolo Y_i+1, DELTA_i e S_i sono stati scelti in modo che
    %anche Y_i+1 sia anch' essa una sezione del problema
    %e in modo che la traccia sia strettamente decrescente
    Y = Ycap * S;
    %calcolo il residuo k-esimo
    err(i) = norm(Aold * Y - B * Y * SIGMA, 'inf');
    %calcolo il residuo k-esimo delle rispettive colonne
    colonna(i, :) = max(abs(Aold * Y - B * Y * SIGMA));
    %calcolo la traccia di SIGMA_k
```

```
        tr(i) = trace(SIGMA);
        for j = 1 : p
                %controlla se la coppia (autovettore, autovalore)
                %j-esima ha raggiunto la condizione d' arresto.
                %In caso affermativo blocca la colonna j-esima di Y
                if bloccata(j) == 0 && i > 1
                        if colonna(i, j) <= tol * colonna(1, j) %||...
                        %colonna(i, j) > 1e1 * colonna(i - 1, j)
                                bloccata(j) = 1;
                        endif
                endif
        endfor
        if bloccata * ones(p, 1) == p
                termina = true;
        endif
        i = i + 1;
endwhile
iter = i - 1;
%creo il vettore con le approssimazioni degli autovalori
sigma = diag(SIGMA)';
Yf = Y;
endfunction
```

5 Problemi, con matrici dense, sperimentati al calcolatore

Ho testato i due codici **tracmn1** e **tracmn2** su un insieme test di cinque problemi. Le matrici A e B sono state scelte in modo tale che gli autovalori del problema (1) hanno la distribuzione mostrata nella tabella 5:

Problema	Distribuzione	n	p
1	$(0, 10, 20, 30, 31, ..., 36)$	10	3
2	$(1, 1.001, ..., 1.004, 49.981, 49.892, ..., 50)$	25	3
3	$(1, 1.001, ..., 1.004, 49.981, 49.892, ..., 50)$	25	6
4	$(1, 1.5, 2, 2.5, 5, 5.5, 6, 6.5, 9, 9.5, ..., 24.5)$	40	8
5	$(-3, -1, 1, 3, 5, ..., 35)$	20	4

dove n è la dimensione delle matrici A e B, p è il numero di autovalori e rispettivi autovettori da approssimare.
Ho generato le matrici A e B, usando la function **generaAB** e il vettore **distribuzione**. La function **generaAB** crea le matrici dense A e B a partire dalla matrice con valori random S. Analizzeremo i nostri due algoritmi **tracmn1** e **tracmn1** anche al variare del numero di condizionamento di S (condS).

```
[A, B, condS] = generaAB(n, distribuzione);
```

La sezione di partenza Y_0, è stata generata tramite il comando:

```
Y = creasezione(A, B, p);
```

In **tracmn2**, per il calcolo delle colonne della matrice correzione Δ ho usato come condizione d' arresto:

- il residuo diminuisce di un fattore **gamma**.

Negli esperimenti ho posto $gamma = 10^{-7}$.
Sia in **tracmn1** che in **tracmn2**, per il calcolo di Y, matrice approssimazione degli autovettori, e di Σ, matrice approssimazione dei rispettivi autovalori, ho usato due condizioni d' arresto:

1. il residuo diminuisce di un fattore **tol**:

$$\|AY_k - BY_k\Sigma_k\|_\infty \leq tol\|AY_0 - BY_0\Sigma_0\|_\infty,$$

2. il numero di iterazioni eseguite è maggiore di **maxit**.

Il calcolo termina quando una delle due condizioni viene verificata.
Negli esperimenti ho eseguito **tracmn1** e **tracmn2** con **maxit** $= 1000$.

5.1 Esperimento 1

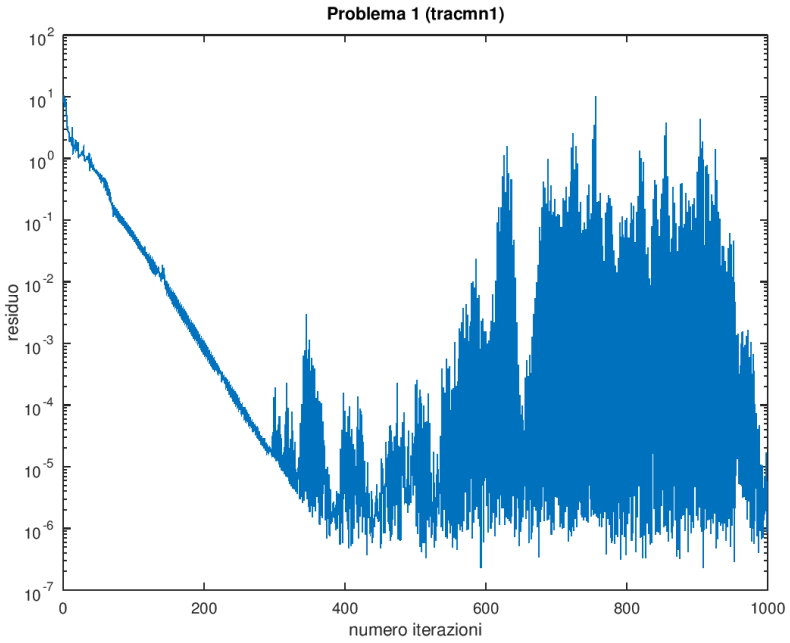
Analizziamo la risoluzione del problema 1 prima con **tracmn1** poi con **tracmn2**, osservando l' andamento del residuo in funzione del numero di iterazioni.

5.1.1 tracmn1

Il Problema 1 risolto con **tracmn1** non dà buoni risultati. Infatti il programma termina poichè eccede il mumero massimo **maxit** di iterazioni eseguite dall' algoritmo, senza verificare il criterio d' arresto sul residuo. Il residuo, come si può vedere dall' immagine, decresce lentamente dopo un certo numero di iterazioni.

	tol	maxit	condS	iterazioni	tempo(s)
tracmn1	$10e-12$	1000	40.889	maxit	0.60122

Qui sotto il grafico del residuo in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:

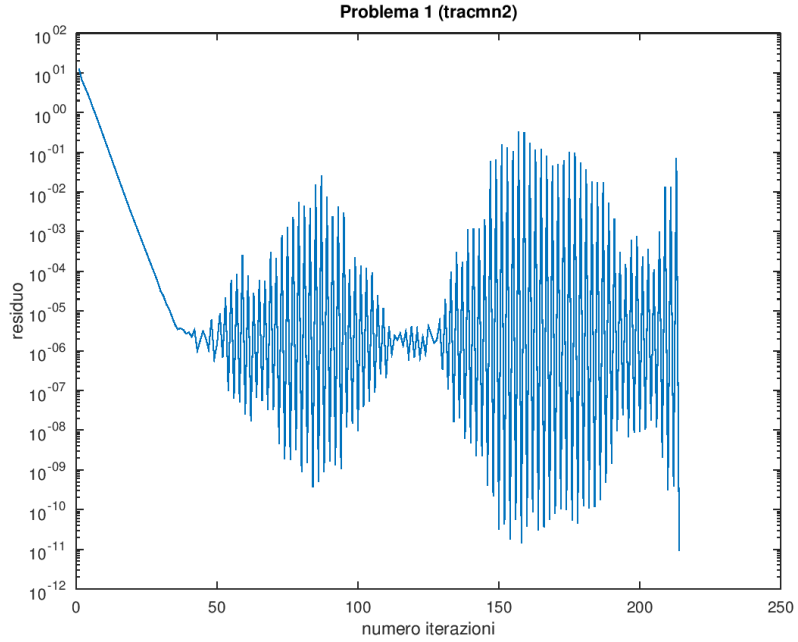


5.1.2 tracmn2

Il Problema 1 risolto con **tracmn2** dà i seguenti risultati:

	tol	gamma	maxit	condS	iterazioni	tempo(s)
tracmn2	$10e-12$	$10e-7$	1000	99.137	214	0.46318

Qui sotto il grafico del residuo in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:



5.2 Esperimento 2

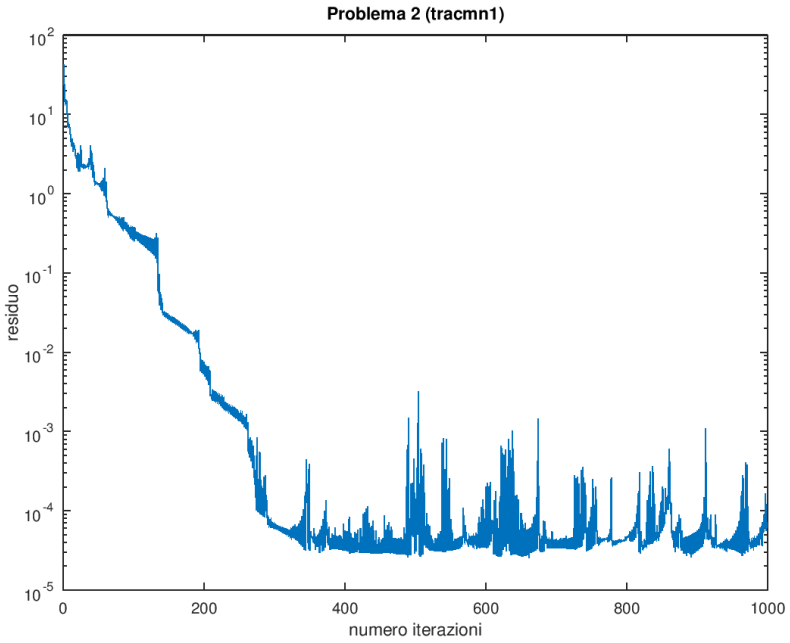
Analizziamo la risoluzione del problema 2 prima con **tracmn1** poi con **tracmn2**, osservando l' andamento del residuo in funzione del numero di iterazioni.

5.2.1 tracmn1

Il Problema 1 risolto con **tracmn1** non dà buoni risultati. Infatti il programma termina poichè eccede il mumero massimo **maxit** di iterazioni eseguite dall' algoritmo, senza verificare il criterio d' arresto sul residuo. Il residuo, come si può vedere dall' immagine, decresce lentamente dopo un certo numero di iterazioni.

	tol	maxit	condS	iterazioni	tempo(s)
tracmn1	$10e-12$	1000	532.16	maxit	0.67484

Qui sotto il grafico del residuo in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:

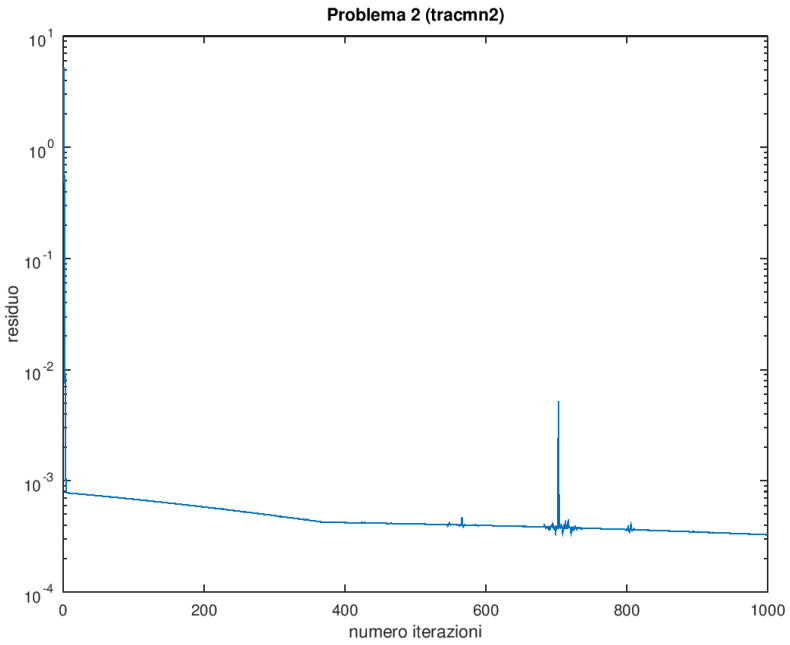


5.2.2 tracmn2

Il Problema 2 risolto con **tracmn2** dà i seguenti risultati.

	tol	gamma	maxit	condS	iterazioni	tempo(s)
tracmn2	$10e-12$	$10e-7$	1000	88.773	maxit	9.2566

Qui sotto il grafico del residuo in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:



5.3 Esperimento 3

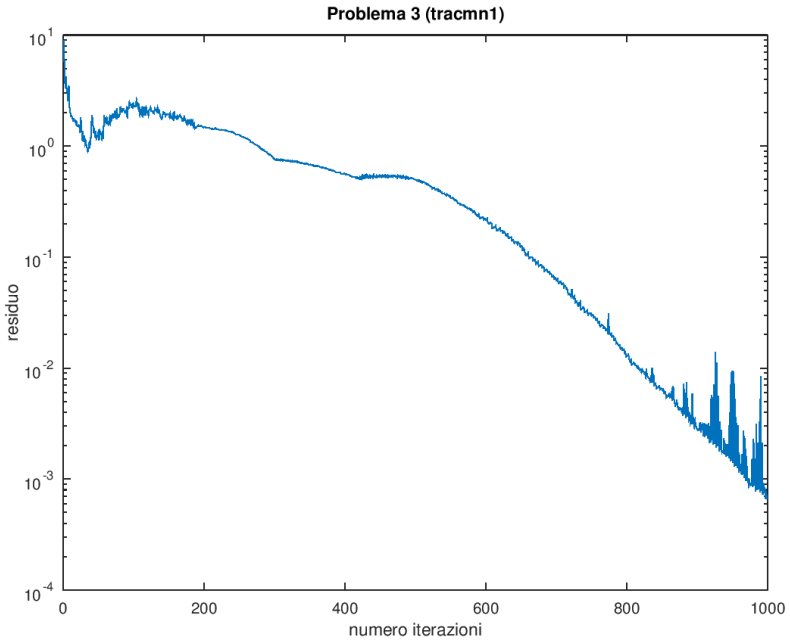
Analizziamo la risoluzione del problema 3 prima con **tracmn1** poi con **tracmn2**, osservando l' andamento del residuo in funzione del numero di iterazioni.

5.3.1 tracmn1

Il Problema 1 risolto con **tracmn1** non dà buoni risultati. Infatti il programma termina poichè eccede il mumero massimo **maxit** di iterazioni eseguite dall' algoritmo, senza verificare il criterio d' arresto sul residuo. Il residuo, come si può vedere dall' immagine, decresce lentamente dopo un certo numero di iterazioni.

	tol	maxit	condS	iterazioni	tempo(s)
tracmn1	$10e-12$	1000	139.46	maxit	0.73368

Qui sotto il grafico del residuo in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:

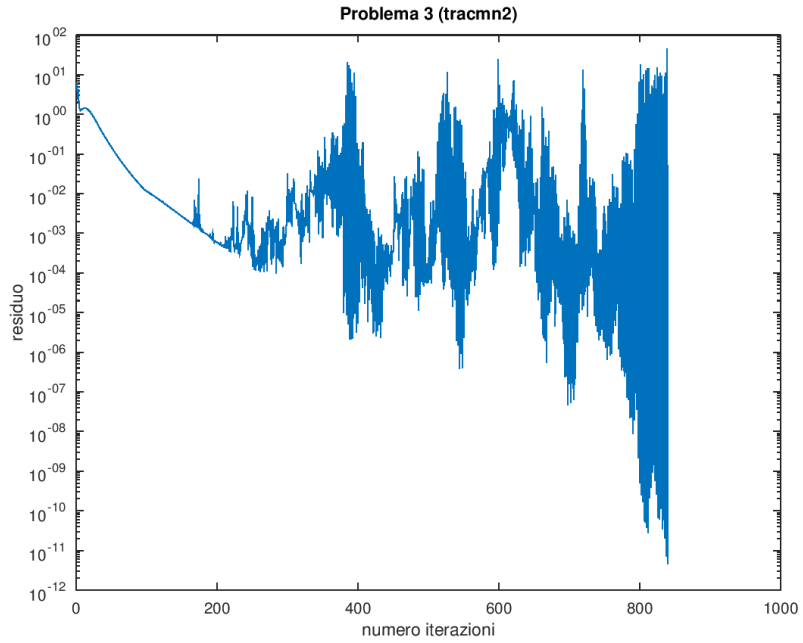


5.3.2 tracmn2

Il Problema 3 risolto con **tracmn2** dà ottimi risultati.

	tol	gamma	maxit	condS	iterazioni	tempo(s)
tracmn2	$10e-12$	$10e-7$	1000	101.67	840	2.0976

Qui sotto il grafico del residuo in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:



5.4 Esperimento 4

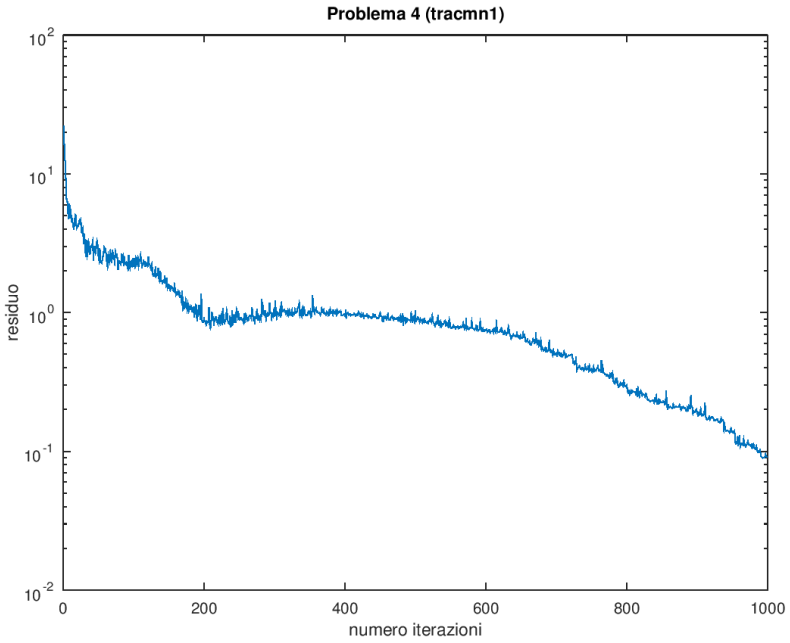
Analizziamo la risoluzione del problema 4 prima con **tracmn1** poi con **tracmn2**, osservando l' andamento del residuo in funzione del numero di iterazioni.

5.4.1 tracmn1

Il Problema 4 risolto con **tracmn1** non dà buoni risultati. Infatti il programma termina poichè eccede il mumero massimo **maxit** di iterazioni eseguite dall' algoritmo, senza verificare il criterio d' arresto sul residuo. Il residuo, come si può vedere dall' immagine, decresce lentamente dopo un certo numero di iterazioni.

	tol	maxit	condS	iterazioni	tempo(s)
tracmn1	$10e-12$	1000	361.36	maxit	1.1492

Qui sotto il grafico del residuo in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:

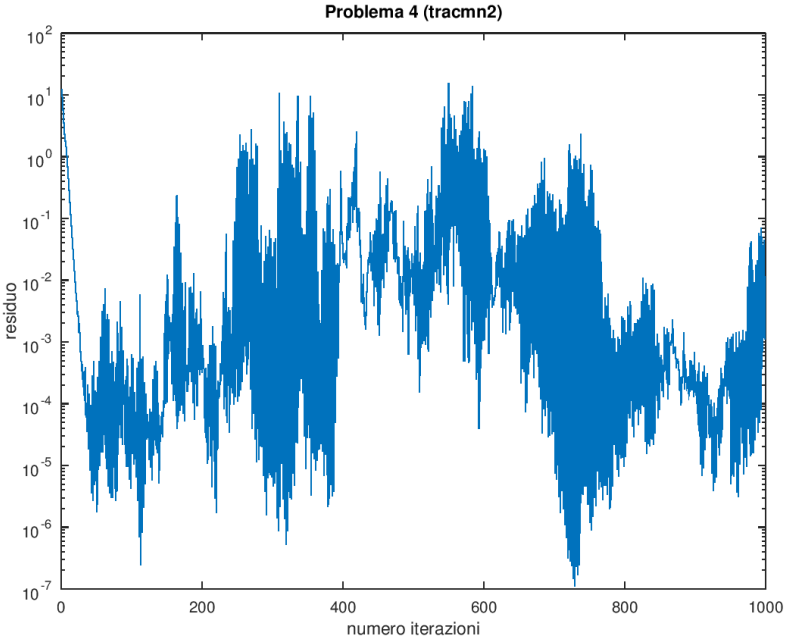


5.4.2 tracmn2

Il Problema 4 risolto con **tracmn2** dà i seguenti risultati.

	tol	gamma	maxit	condS	iterazioni	tempo(s)
tracmn2	$10e-12$	$10e-7$	1000	992.46	maxit	84.035

Qui sotto il grafico del residuo in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:



5.5 Esperimento 5

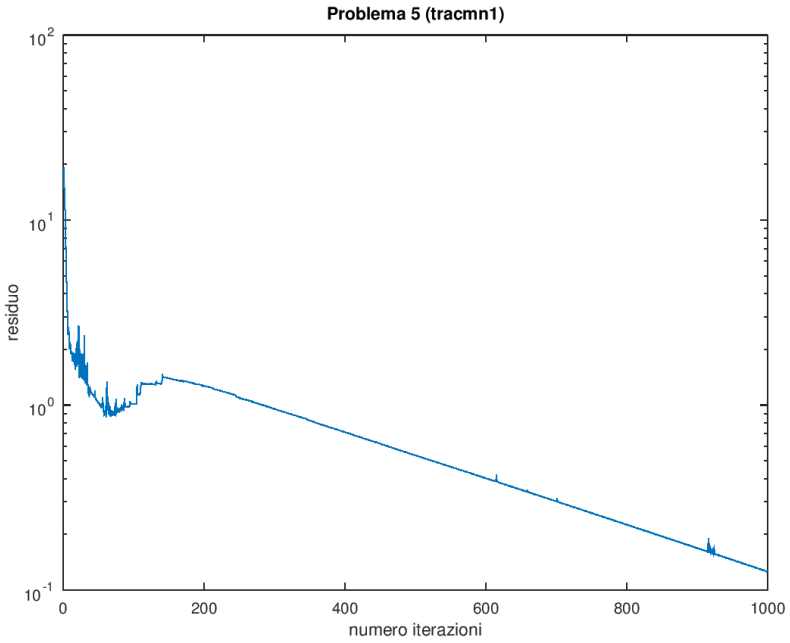
Analizziamo la risoluzione del problema 5 prima con **tracmn1** poi con **tracmn2**, osservando l' andamento del residuo in funzione del numero di iterazioni.

5.5.1 tracmn1

Il Problema 1 risolto con **tracmn1** non dà buoni risultati. Infatti il programma termina poichè eccede il mumero massimo **maxit** di iterazioni eseguite dall' algoritmo, senza verificare il criterio d' arresto sul residuo. Il residuo, come si può vedere dall' immagine, decresce lentamente dopo un certo numero di iterazioni.

	tol	maxit	condS	iterazioni	tempo(s)
tracmn1	$10e-12$	1000	109.54	maxit	0.65190

Qui sotto il grafico del residuo in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:

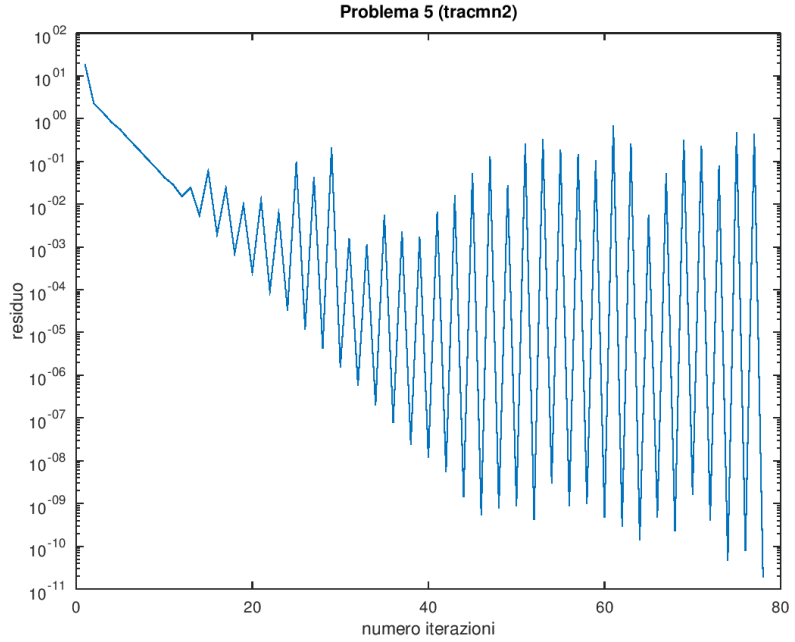


5.5.2 tracmn2

Il Problema 5 risolto con **tracmn2** dà i seguenti risultati.

	tol	gamma	maxit	condS	iterazioni	tempo(s)
tracmn2	$10e-12$	$10e-7$	1000	544.49	78	0.62367

Qui sotto il grafico del residuo in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:



6 Problemi, con matrici sparse, sperimentati al calcolatore

Problema	n	p	d_A	d_B
6	50	5	0.20560	0.20640
7	100	5	0.19760	0.19840

dove n è la dimensione delle matrici A e B, p è il numero di autovalori e rispettivi autovettori da approssimare e d_A e d_B sono le densità delle matrici sparse A e B.
Ho generato le matrici A e B, usando la function **spgeneraAB**.

```
[A, B, condS] = spgeneraAB(n, d);
```

La sezione di partenza Y_0, è stata generata tramite il comando:

```
Y = spcreasezione(A, B, p);
```

Poichè la dimensione n delle matrici A e B è abbastanza grande, per ottimizzare l' algoritmo uso l' euristica descritta nel paragrafo **??**. tale euristica usa la tecnica dello shifting ed é implementata dal codice **tracmnshifting**. Per il calcolo delle colonne della matrice correzione Δ ho usato come condizione d' arresto:

- il residuo diminuisce di un fattore **gamma**, questa volta posto uguale alla precisione di macchina *eps*.

Per il calcolo di Y, matrice approssimazione degli autovettori, e di Σ, matrice approssimazione dei rispettivi autovalori, ho usato due condizioni d' arresto:

1. il residuo diminuisce di un fattore **tol**:

$$\|AY_k - BY_k\Sigma_k\|_\infty \leq tol\|AY_0 - BY_0\Sigma_0\|_\infty,$$

2. il numero di iterazioni eseguite è maggiore di **maxit**.

Il calcolo termina quando una delle due condizioni viene verificata.
Negli esperimenti ho eseguito **tracmnshifting** con **maxit** = 1000.

6.1 Esperimento 6

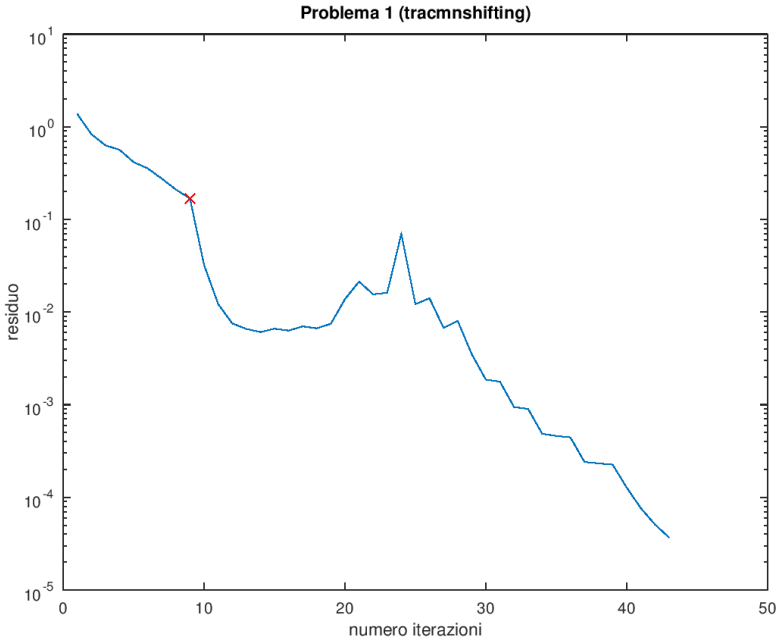
Analizzo la risoluzione del problema 6 con **tracmnshifting**, osservando l' andamento del residuo in funzione del numero di iterazioni. Uso un approccio euristico facendo variare l' iterazione **start** in cui si esugue lo *shifting*; analizzando il residuo, infine, ricavo la strategia migliore per lo *shifting*.

	start	tol	gamma	maxit	iterazioni	tempo(s)
tracmnshifting	10	$10e-6$	*eps*	1000	43	1.5043
tracmnshifting	30	$10e-6$	*eps*	1000	41	1.2675
tracmnshifting	50	$10e-6$	*eps*	1000	61	1.5043
tracmnshifting	100	$10e-6$	*eps*	1000	102	1.7245

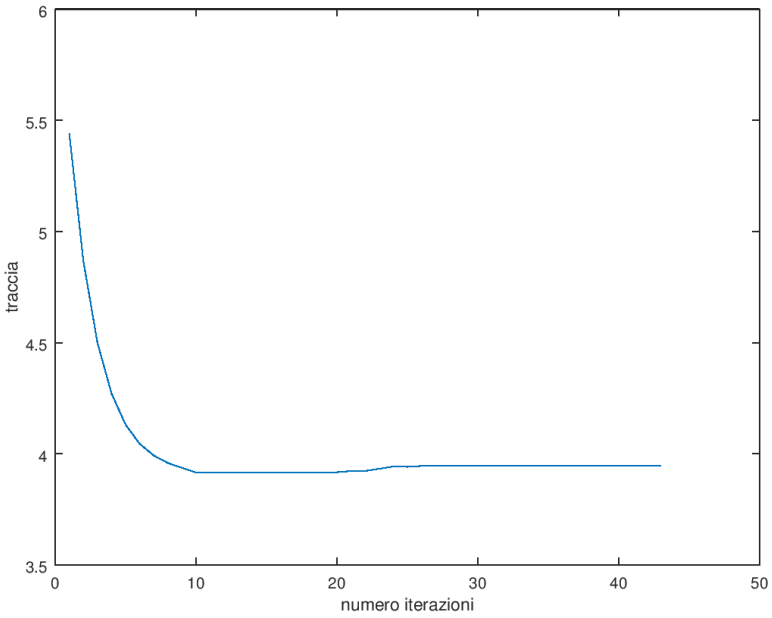
6.1.1 start=10

	Approssimazione autovalori				
eig	0.73190	0.75230	0.77241	0.79058	0.82225
tracmn2	0.73190	0.75230	0.77241	0.51373	0.82225

Qui sotto il grafico del residuo in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:



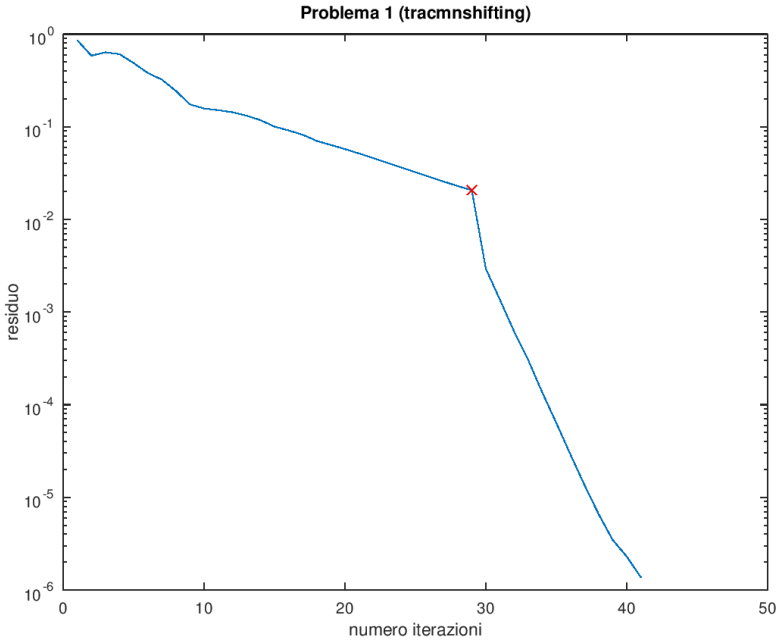
Qui sotto il grafico di $tr(\Sigma_k)$ in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:



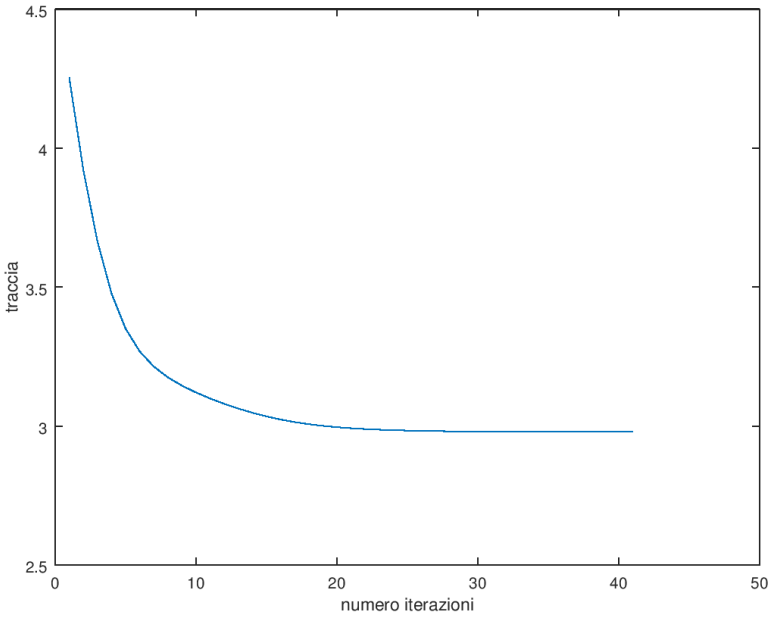
6.1.2 start=30

	Approssimazione autovalori				
eig	0.53054	0.57684	0.60634	0.61517	0.64750
tracmn2	0.53054	0.57684	0.60634	0.61517	0.64750

Qui sotto il grafico del residuo in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:



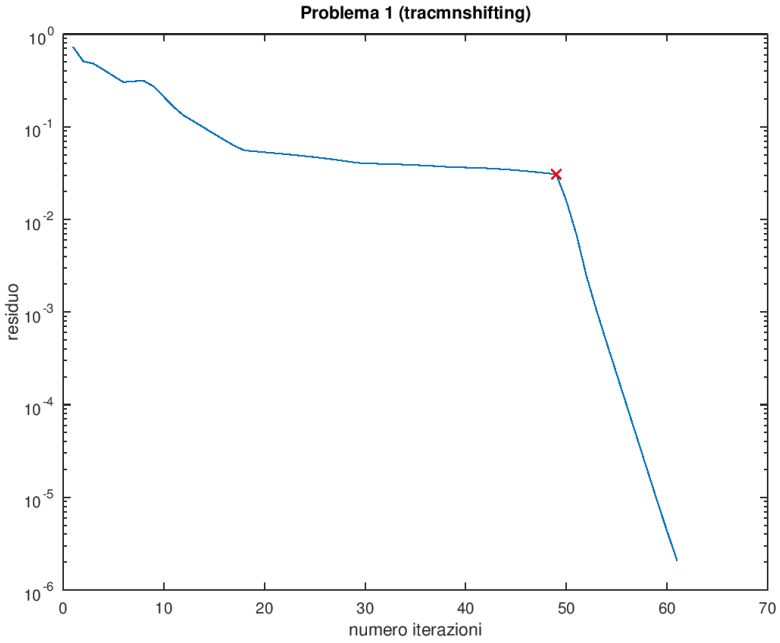
Qui sotto il grafico di $tr(\Sigma_k)$ in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:



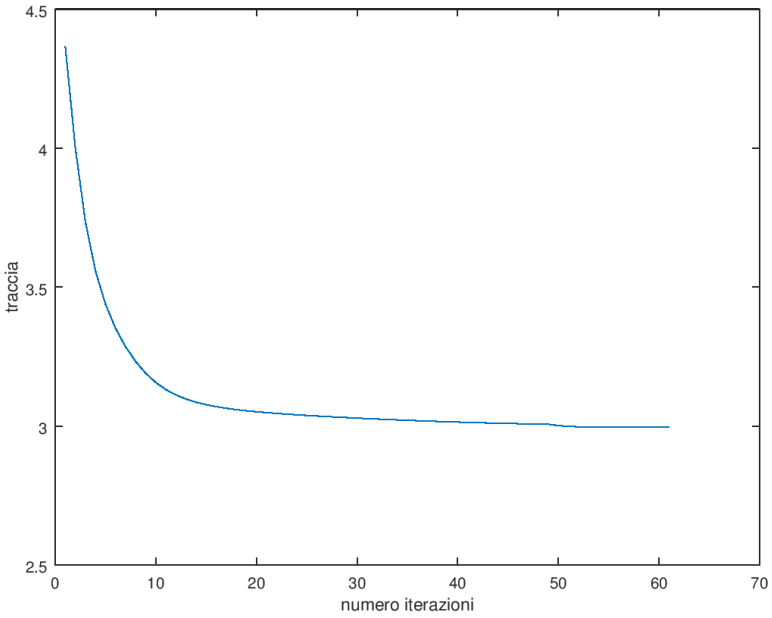
6.1.3 start=50

	Approssimazione autovalori				
eig	0.53501	0.57280	0.61021	0.63219	0.64689
tracmn2	0.53501	0.57280	0.61021	0.63219	0.64689

Qui sotto il grafico del residuo in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:



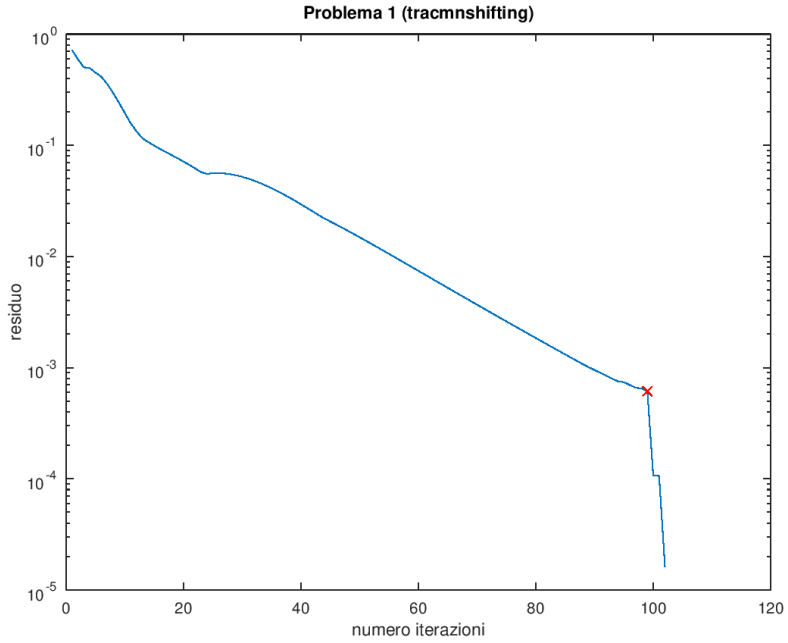
Qui sotto il grafico di $tr(\Sigma_k)$ in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:



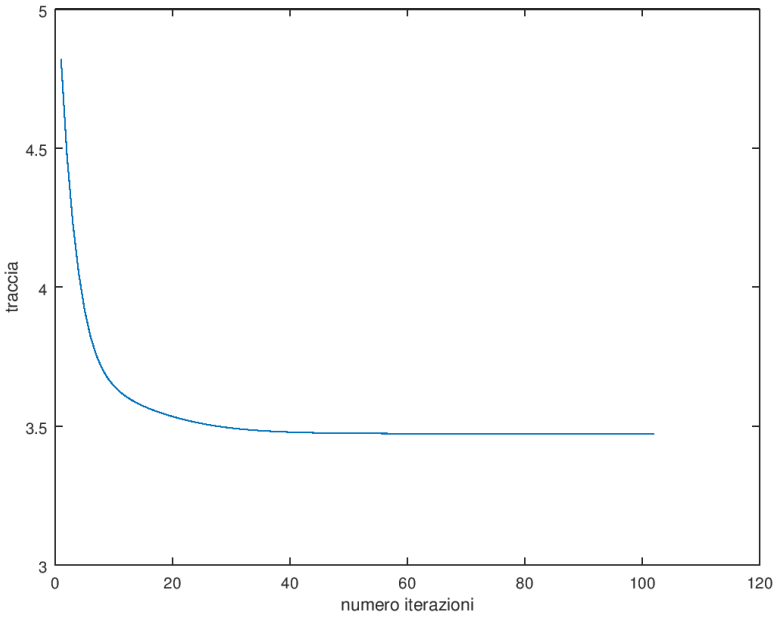
6.1.4 start=100

	Approssimazione autovalori				
eig	0.62020	0.67250	0.71571	0.72545	0.73850
tracmn2	0.62020	0.67250	0.71571	0.72545	0.73850

Qui sotto il grafico del residuo in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:



Qui sotto il grafico di $tr(\Sigma_k)$ in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:



6.2 Esperimento 7

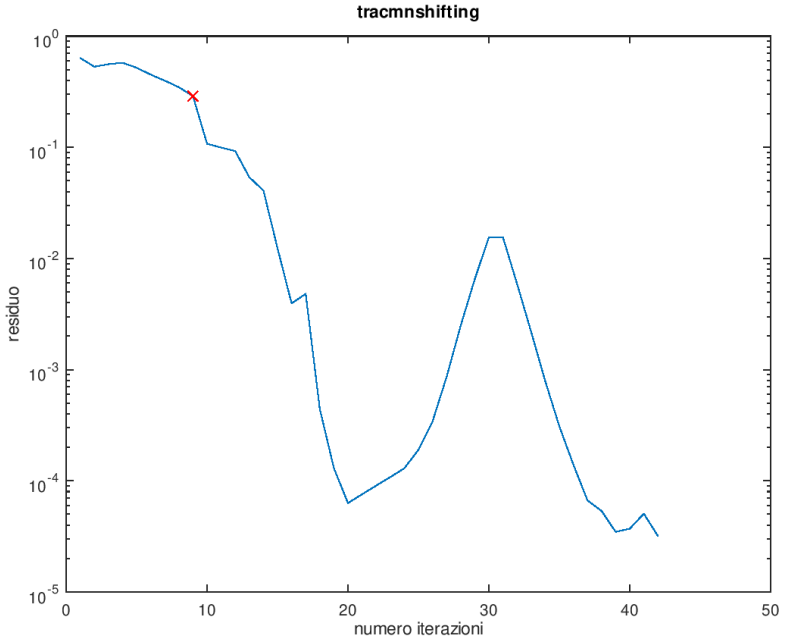
Analizzo la risoluzione del problema 7 con **tracmnshifting**, osservando l' andamento del residuo in funzione del numero di iterazioni. Uso un approccio euristico facendo variare l' iterazione **start** in cui si esugue lo *shifting*; analizzando il residuo, infine, ricavo la strategia migliore per lo *shifting*.

	start	tol	gamma	maxit	iterazioni	tempo(s)
tracmnshifting	10	$10e-6$	*eps*	1000	42	18.595
tracmnshifting	30	$10e-6$	*eps*	1000	39	9.9919
tracmnshifting	50	$10e-6$	*eps*	1000	56	11.692
tracmnshifting	100	$10e-6$	*eps*	1000	102	17.260

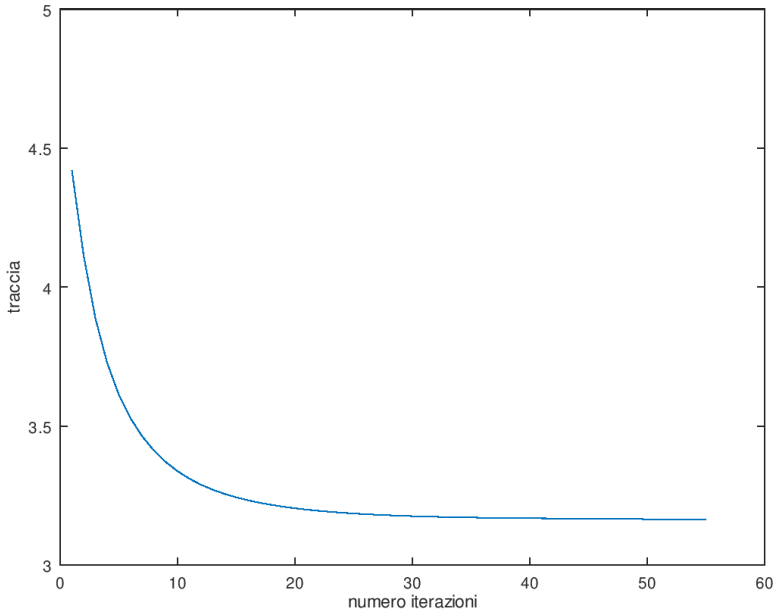
6.2.1 start=10

	Approssimazione autovalori				
eig	0.63779	0.64192	0.66185	0.66925	0.67673
tracmn2	0.63779	0.64192	0.66185	0.66925	0.67673

Qui sotto il grafico del residuo in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:



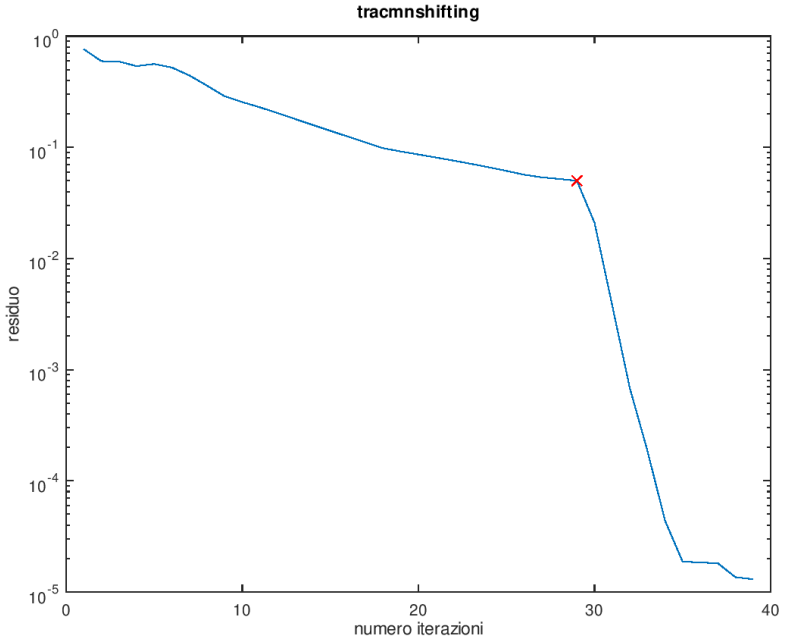
Qui sotto il grafico di $tr(\Sigma_k)$ in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:



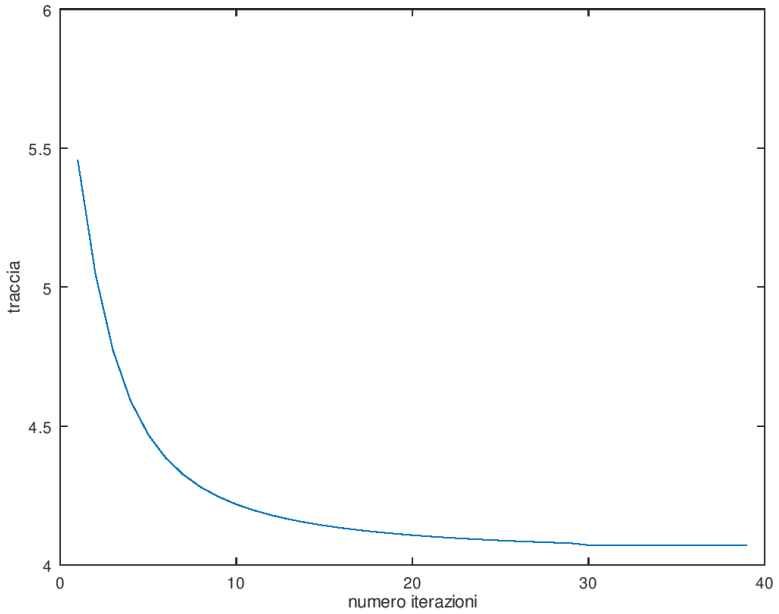
6.2.2 start=30

	Approssimazione autovalori				
eig	0.77815	0.80097	0.81768	0.81936	0.83507
tracmn2	0.77815	0.80097	0.81768	0.81936	0.83507

Qui sotto il grafico del residuo in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:



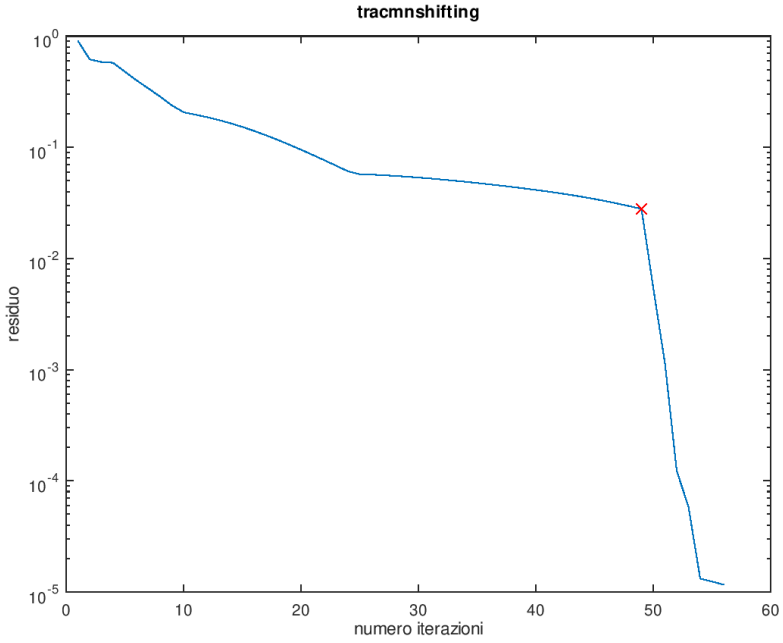
Qui sotto il grafico di $tr(\Sigma_k)$ in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:



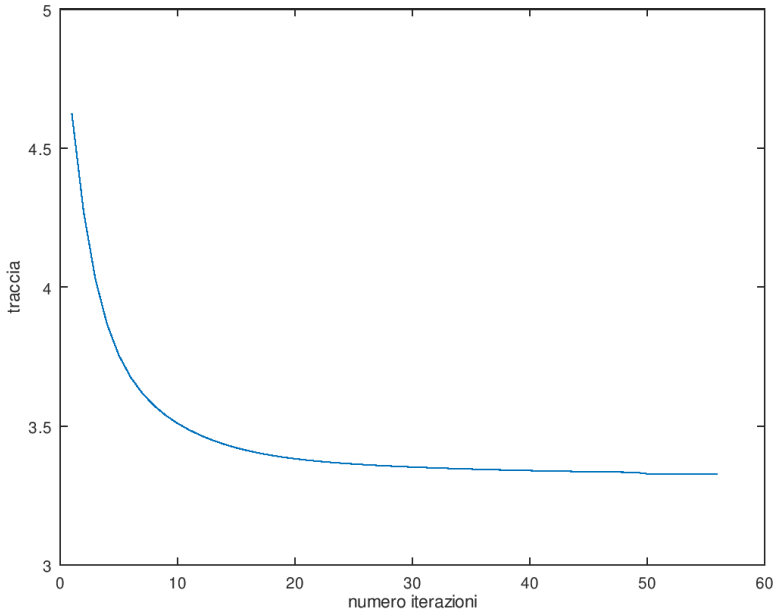
6.2.3 start=50

	Approssimazione autovalori				
eig	0.64832	0.65752	0.66645	0.67428	0.68097
tracmn2	0.64832	0.65752	0.66645	0.67428	0.68097

Qui sotto il grafico del residuo in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:



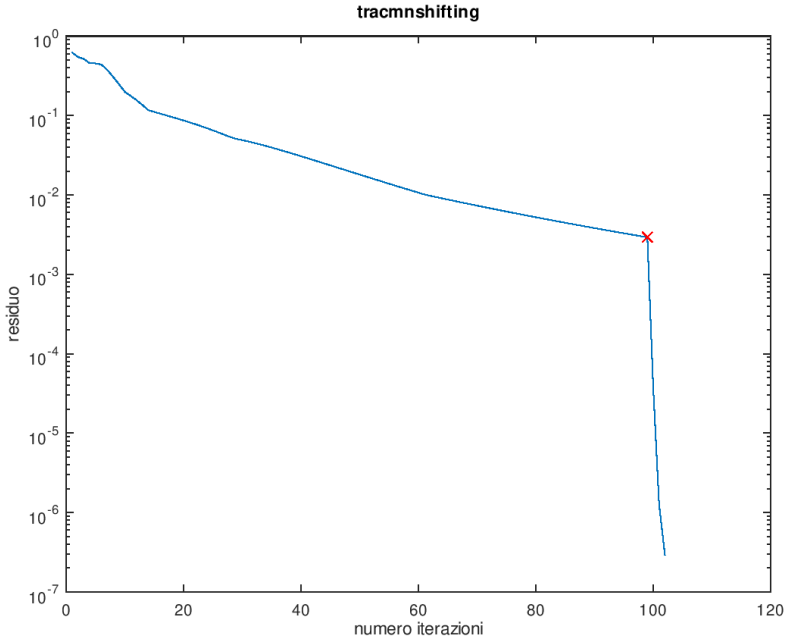
Qui sotto il grafico di $tr(\Sigma_k)$ in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:



6.2.4 start=100

	Approssimazione autovalori				
eig	0.56980	0.59596	0.60737	0.61675	0.62322
tracmn2	0.56980	0.59596	0.60737	0.61675	0.62322

Qui sotto il grafico del residuo in funzione del numero di iterazioni nell' intervallo $[1, 1000]$:



Qui sotto il grafico di $tr(\Sigma_k)$ in funzione del numero di iterazioni nell' intervallo $[1, 1000]$: