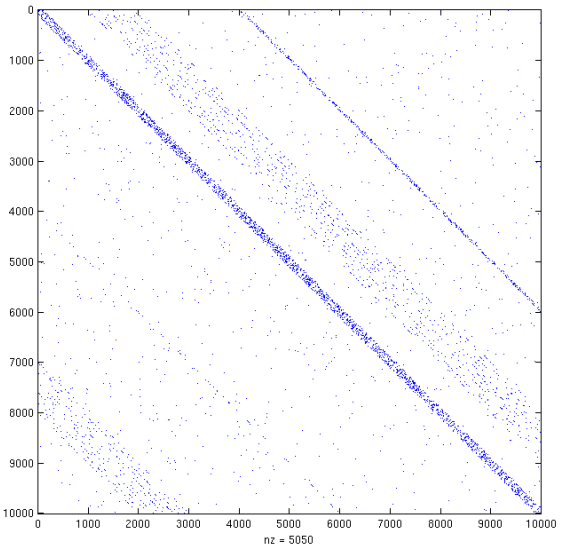
Seminario di metodi di approssimazione

Metodi iterativi per la risoluzione di sistemi lineari sparsi

Mele Giampaolo

4 agosto 2012



Sommario

Verranno presentati metodi iterativi noti come *metodi dei sottospazi di Krylov* i quali permettono di risolvere iterativamente un sistema lineare la cui matrice è sparsa.

Indice

1 Introduzione

La computazione di matrici sparse di per se è un argomento molto affascinante e ha applicazioni in tantissimi campi differenti. Gli esempi più immediati sono quelli legati alla teoria dei grafi, infatti determinare la distanza tra due nodi o la presenza di comunità equivale a risolvere un problema di algebra lineare, inoltre se il grafo è sparso (ovvero se ci sono n nodi ci sono $O(n)$ archi) allora tale problema in algebra lineare sarà definito da una matrice sparsa. Un primo esempio potrebbe essere il grafo di un *social network* come Facebook, non solo, anche le reti p2p sono strutturate come un grafo e chiaramente entrambi questi problemi hanno natura sparsa e grossa dimensione. Tali problemi sono però definiti da matrici simmetriche e in questi casi fortunati si può fare una trattazione leggermente diversa da quella fatta in queste pagine (più semplice ed efficiente). Daltronde in molti altri problemi si incontrano matrici sparse non simmetriche, la maggior parte di tali problemi nasce dalla risoluzione numerica di equazioni differenziali alle derivate parziali, in particolare dalla discretizzazione di operatori differenziali utilizzando il metodo delle differenze finite o il metodo degli elementi finiti. Ad ogni modo il problema che si affronterà sarà quello della risoluzione di sistemi lineari $Ax = b$ dove A è una matrice sparsa sulla quale facciamo l'unica ipotesi che sia invertibile (cosa che succede nei casi sopracitati), si evita la trattazione del caso singolare o malcondizionato, si presentano piuttosto i principali metodi di risuluzione di tali problemi con delle ingenue sperimentazioni numeriche (si può far decisamente di meglio) a supporto delle argomentazioni esposte. Tutto il materiale è stato preso dal libro *Iterative Methods for Sparse Linear Systems* di Yousef Saad, in particolare dai capitoli 6 e 7. Altro materiale è stato reperito su internet. La sperimentazione numerica è stata svolta completamente in matlab.

2 Prime definizioni

2.1 Risoluzione per proiezione

L'obbiettivo è risolvere un sistema lineare della forma

$$Ax = b \qquad A \in \mathbb{R}^{n \times n}$$

Definizione 2.1 (Risoluzione per proiezione)**.** Fissati due sottospazi K_m e L_m di dimensione m e una stima iniziale della soluzione x_0 allora si prende come stima della soluzione del sistema lineare x_m tale che

- $x_m \in x_0 + K_m$
- $b - Ax_m \perp L_m$ (*condizione di Petrov-Galerkin*)

Si vuole ora capire come è fatto concretamente un metodo di risoluzione per proiezione. Si è detto che la soluzione approssimata deve avere la forma

$$x_m = x_0 + y_m \qquad \text{con} \quad y_m \in L_m$$

Sia dunque $v_1, \ldots, v_m$ una base di K_m e $w_1, \ldots, w_m$ una base di L_m allora è possibile esprimere

$$y_m = \sum_{i=1}^{m} z_i v_i = V_m z$$

Inoltre il residuo m-esimo è definito come

$$\begin{aligned} r_m &= b - Ax_m \\ &= r_0 - Ay_m \end{aligned}$$

e questo deve esser ortogonale a L_m per Petrov-Galerkin. Scrivendo esplicitamente la condizione di ortogonalità si ha

$$(w_i, r_0 - Ay_m) = 0 \qquad \text{per} \quad i = 1, \dots, m$$

Sostituendo y_m, è possibile riscrivere tutto in forma matriciale come

$$W_m^T(r_0 - AV_m z) = 0$$

Posto che la matrice $W_m^T A V_m$ sia invertibile, è possibile scrivere i coefficienti come

$$z = (W_m^T A V_m)^{-1} W_m^T r_0$$

da cui si ricava che la soluzione approssimata è la seguente

$$x_m = x_0 + V_m (W_m^T A V_m)^{-1} W_m^T r_0$$

Definizione 2.2 (Metodi di Krylov)**.** Un metodo di Krylov è una risoluzione per proiezione dove, fissata la stima iniziale x_0, si definisce $r_0 = b - Ax_0$ e si sceglie come spazio K_m il sottospazio $K_m(A, r_0)$ (sottospazio di Krylov) definito come

$$K_m(A, r_0) = \text{span}\left\{r_0, Ar_0, A^2 r_0, \dots, A^{m-1} r_0\right\}$$

I metodi di Krylov differiscono in base alla scelta dello spazio L_m.

Si analizzeranno solo due casi: $L_m = K_m$ (metodo di Arnoldi) ed $L_m = AK_m$ (GMRES), in realtà sono possibili anche altre scelte che non saranno esaminate.

Osservazione 2.1. L'approssimazione data da un metodo di Krylov è della forma

$$A^{-1} b \approx x_m = x_0 + q_{m-1}(A) r_0$$

Dove q_{m-1} è un polinomio di grado al più $m - 1$.

2.2 Sottospazi di Krylov

Verrano di seguito caratterizzati i sottospazi della forma

$$K_m(A, v) = \text{span}\left\{v, Av, A^2 v, \dots, A^{m-1} v\right\}$$

Questi sono detti sottospazi di Krylov e spesso saranno indicati solo con K_m.

Osservazione 2.2. Una prima caratterizzazione banale è la seguente

$$x \in K_m \iff \exists \;\; p \;\; \text{polinomio} \quad \text{t.c.} \quad x = p(A)v$$

Dove il grado di p non può eccedere $m - 1$.

Definizione 2.3. Fissato un vettore v, il suo polinomio minimo rispetto A è il polinomio monico di grado più basso tale che $p(A)v = 0$, per il teorema di Cayley-Hamilton il grado di tale polinomio è minore di $n-1$, tale grado è detto *grado di v rispetto A*.

Proposizione 2.1. Sia μ il grado di v (rispetto A), allora K_μ è invariante sotto l'azione di A e di conseguenza $K_m = k_\mu$ per ogni $m \geq \mu$

Osservazione 2.3. La dimensione degli spazi K_m è chiaramente non decrescente e in generale non è detto sia m, basti prendere $m \geq n$ e usare il teorema di Cayley-Hamilton. La proposizione seguente da una caratterizzazione della dimensione.

Proposizione 2.2. Un sottospazio di Krylov K_m ha dimensione m se e soltato se il grado di v rispetto A è m. Vale in generale che

$$\dim(K_m) = \min\{m, \deg(v)\}$$

NB: si ricorda che la notazione completa è $K_m(A, v)$

Definizione 2.4 (proiettore)**.** Un proiettore P è un'applicazione lineare tale che $P^2 = P$ (quindi l'immagine è invariante)

Proposizione 2.3. Sia Q_m un proiettore in K_m (ovvero che ha come immagine un sottospazio di K_m) e si A_m la sezione di A in K_m ovvero $A_m = Q_m A|_{K_m}$ allora per ogni polinomio di grado al più $m-1$ vale

$$q(A)v = q(A_m)v$$

E per ogni polinomio di grado al più m

$$Q_m q(A)v = q(A_m)v$$

2.3 Convergenza dei metodi per proiezione

La classe dei metodi per proiezione è certamente convergente, infatti si minimizza il residuo su uno spazio di dimensione crescente , questi metodi diventano metodi diretti per $m = n$ (*propriprietà di terminazione finita*). In generale si ha che

- $\|r_k\|_2$ è una successione decrescente
- $\|r_n\|_2 = 0$

In generale però questo vale se si considera tutto in aritmetica esatta.

Nel caso dei metodi di Krylov si può dire di più sulla convergenza, infatti per le osservazioni appena fatte si ha che l'approssimazione avrà la forma $p_k(A)x_0$ dove p_k è un polinomio di grado al più k. Per seplicità si suppone che A sia diagonalizzabile (un'analisi più dettagliata è possibile ma decisamente più complessa). Sia dunque $A = VDV^{-1}$, allora

$$\begin{aligned}\|r_k\|_2 &= \min_{p_k} \|Vp_k(D)V^{-1}\|_2 \|r_0\|_2 \\ &\leq \operatorname{cond}(V) \min_{p_k} \|p_k(D)\|_2 \|r_0\|_2\end{aligned}$$

Quindi si ottiene

$$\frac{\|r_k\|_2}{\|r_0\|_2} \leq \text{cond}(V) \min_{p_k} \max_{1 \leq i \leq m} |p_k(\lambda_i)|$$

Si osserva subito che per il teorema di Cayley-Hamilton il residuo è nullo per $k = m$ (cosa che già era nota). Daltronde si osserva anche che la convergenza del metodo dipende dal condizionamento di V, pertanto possono presentarsi situazioni in cui il problema è malcondizionato e, anche se teoricamente dovrebbe esserci convergenza, numericamente i metodi falliscono o richiedono m molto grande, cosa che come si vedrà rende impraticabili questi metodi. E' possibile utilizzare tecniche di precondizionamento ma la vera difficoltà è determinare dei precondizionatori efficienti. Per far ciò spesso si sfrutta la natura del problema se ad esempio nasce dalla risuluzione di una equazione alle derivate parziali. Esistono anche precondizionatori generali ma per introdurre tale teoria bisognerebbe parlare di fattorizzazioni LU incomplete e tasso di fill-in. Si rimanda alla referenza principale per l'approfondimento di tali questioni.

3 Metodo di Arnoldi

Il metodo di Arnoldi determina una base ortogonale del sottospazio di Krylov K_m e a partire da questo si riusciranno a risolvere sistemi lineari. Esistono quattro varianti pricipali di questo algoritmo, in questo contesto si mostreranno solo due di queste varianti, ovvero la versione GS (Gram-Schmidt) e la versione MGS (Modified Gram-Schmidt), la prima è la più intuitiva ma numericamente non stabile, la seconda è equivalente alla prima ma numericamente più stabile.

Algorithm 1 GS

Si sceglie un vettore v_1 di norma unitaria
for $j = 1, 2, \ldots, m$ **do**
  for $i = 1, \ldots, j$ **do**
    calcolo $h_{i,j} = (Av_j, v_i)$
  end for
  calcolo $w_j := Av_j - \sum_{i=1}^{j} h_{i,j} v_i$
  $h_{j+1,j} = \|w_j\|_2$
  if $h_{j+1,j} = 0$ **then**
    stop
  else
    $v_{j+1} = w_j / h_{j+1,j}$
  end if
end for

Algorithm 2 MGS

Si sceglie un vettore v_1 di norma unitaria
for $j = 1, 2, \ldots, m$ **do**
  calcolo $w_j := Av_j$
  for $i = 1, \ldots, j$ **do**
    calcolo $h_{i,j} = (w_j, v_i)$
    calcolo $w_j = w_j - h_{i,j}v_i$
  end for
  $h_{j+1,j} = \|w_j\|_2$
  if $h_{j+1,j} = 0$ **then**
    stop
  else
    $v_{j+1} = w_j/h_{j+1,j}$
  end if
end for

Osservazione 3.1. Il costo computazionale di GS e di MGS è $2m^2n$ mentre la memoria richiesta è $(m+1)n$

Ad ogni modo queste due varianti non sono le migliori ma per non appesantire troppo tutta l'argomentazione ci sono sufficienti dato che l'obbiettivo è l'utilizzo di Arnoldi per la risoluzione dei sistemi lineari.

Proposizione 3.1. Se l'algoritmo 1 (o equivalente il 2) non si arresta dopo m passi allora i vettori $v_1, \ldots, v_m$ formano una base ortonormale del sottospazio di Krylov

$$K_m = \left\{v_1, Av_1, \ldots, A^{m-1}v_1\right\}$$

Proposizione 3.2. Si consideri la matrice $V_m \in \mathbb{R}^{n\times m}$ che ha per vettori colonna $v_1, \ldots, v_m$ (base ortonormale di K_m, calcolata ad esempio con l'algoritmo 1 o 2) e sia $\bar{H}_m \in \mathbb{R}^{(m+1)\times m}$ la matrice (di Hessenberg) i cui elementi sono i $h_{i,j} = (Av_j, v_i)$ e sia H_m la matrice ottenuta da $\bar{H}_m$ eliminando l'ultima riga. Allora valgono le seguenti relazioni

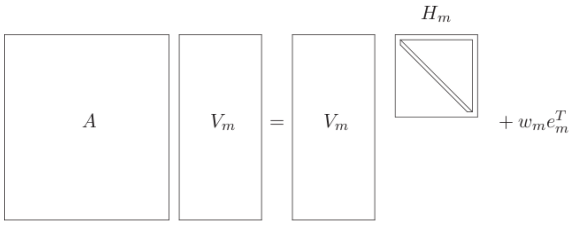
$$\begin{aligned} AV_m &= V_m H_m + w_m e_m^T \\ &= V_{m+1}\bar{H}_m \\ V_m^T A V_m &= H_m \end{aligned}$$

Dimostrazione. La dimostrazione segue banalmente dall'algoritmo 1 o 2, infatti per come sono state costruite le matrici vale

$$Av_j = \sum_{i=1}^{j+1} h_{i,j}v_i \qquad j = 1, \ldots, m$$

e per costruzione i vettori v_i sono ortonormali, quindi segue banalmente la tesi. □

Osservazione 3.2. L'azione di A su V_m da $V_m H_m$ più una correzione di rango 1.



Proposizione 3.3. L'agoritmo 1 o 2 si arrestano al passo j-esimo (ovvero $h_{j+1,j}$) se il polinomio minimo di v_1 rispetto A ha grado j e in questo caso lo spazio K_j è invariante sotto l'azione di A.

A questo punto è possibile mostrare i primi metodi di risoluzione per sistemi lineari sparsi.

3.1 Metodo di ortogonalizzazione completa (FOM)

Il metodo di ortogonalizzazione completa, che sarà indicato con l'acronimo FOM (*Full Orthogonalization Method*) si basa esattamente su quanto detto all'inizio, ovvero si risolve il sistema lineare per proiezione scegliendo $L_m = K_m$.

Dato x_0 una stima iniziale della soluzione allora si pone $r_0 = b - Ax_0$, $K_m = K_m(A, r_0)$ e si cerca un'approssimazione della soluzione x_m nello spazio affine $x_0 + K_m$ imponendo la condizione di Galerkin

$$b - Ax_m \perp K_m$$

Allora si pone $v_1 = r_0/\|r_0\|_2$ e $\beta = \|r_0\|_2$ allora l'approssimazione sarà data da

$$\begin{aligned} x_m &= x_0 + V_m y_m \\ y_m &= H_m^{-1}(\beta e_1) \end{aligned}$$

Per capire questa formula è sufficiente ricordare come è fatta una soluzione approssimata calcolata con un metodo per proiezione (si veda sezione precedente), osservare che in questo caso $V_m = W_m$ e che $r_0 = \beta v_1$, inoltre si usa che $V_m^T v_1 = e_1$ per ortonormalità della base. Usando queste osservazioni, svolgendo il calcolo si trova esattamente la formula appena scritta. Un'altra propriprietà interessante del FOM è che si ha un controllo del residuo automatico, infatti con un semplice calcolo si mostra che

$$b - Ax_m = -h_{m+1,m} e_m^T y_m v_{m+1}$$

e quindi

$$\|b - Ax_m\|_2 = h_{m+1,m} |e_m^T y_m|$$

3.2 Sperimentazione numerica su FOM

Di seguito c'è l'implementazione in matlab del FOM.

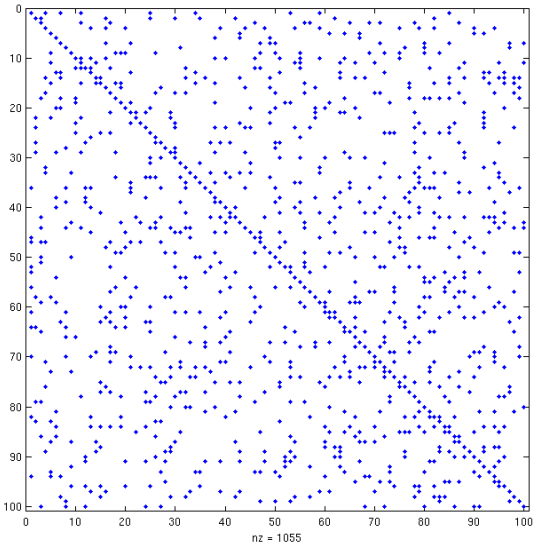
Listing 1: FOM

```
function xm = FOM(A, b, x0, m)
% implementazione matlab di FOM

% inizializzazione
n=size(A,1);
```

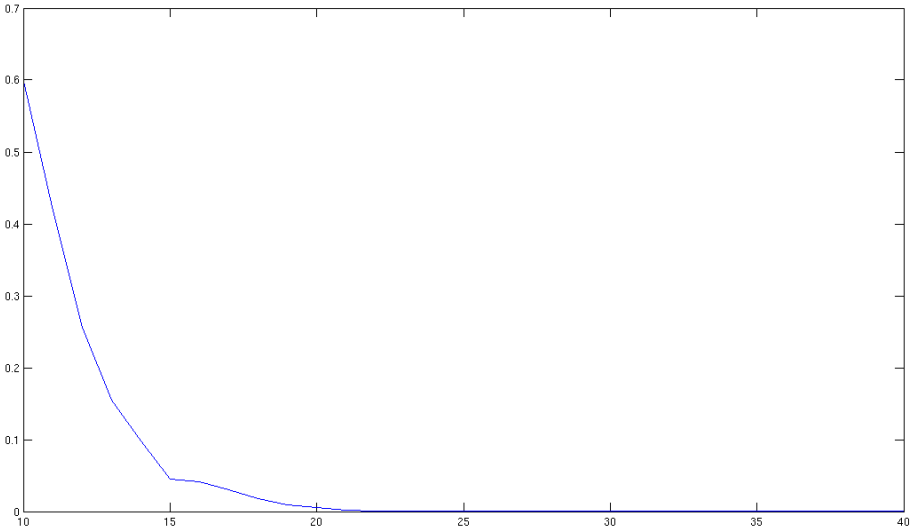
```
r0= b - A*x0;
beta=norm(r0);
V(:,1)=r0/beta;
H=0;

for j=1:1:m
    wj=A*V(:,j);



    for i=1:j
        H(i,j)=wj'*V(:,i);
        wj=wj-H(i,j)*V(:,i);
    end

    H(j+1,j)=norm(wj);

    if (H(j+1,j)==0)
        'breakdown happened'
        m=j;
        ym=[beta;zeros(m-1,1)];
        H=H(1:m,1:m);
        ym=inv(H)*ym;
        V=V(1:n,1:m);
        xm=x0+V*ym
    else

        V(:,j+1)=wj/H(j+1,j);
    end
end


ym=[beta;zeros(m-1,1)];
H=H(1:m,1:m);
if(abs(det(H))<10^(-10))
    xm=zeros(n,1);
else
    ym=H\ym;
    V=V(1:n,1:m);
    xm=x0+V*ym;
end



end
```

Esempio 3.1. Si consideri una matrice $A \in \mathbb{R}^{100\times 100}$ con elementi casuali e una sparsità del 10% (ovvero solo il 10% dei suoi elementi sono non nulli), inoltre per assicurare la non singolarità si impone la

forte dominanza diagonale pomendo nella posizione (i, i) della matrice la somma dei valori assoluti degli elementi sulla riga i-esima più 1.

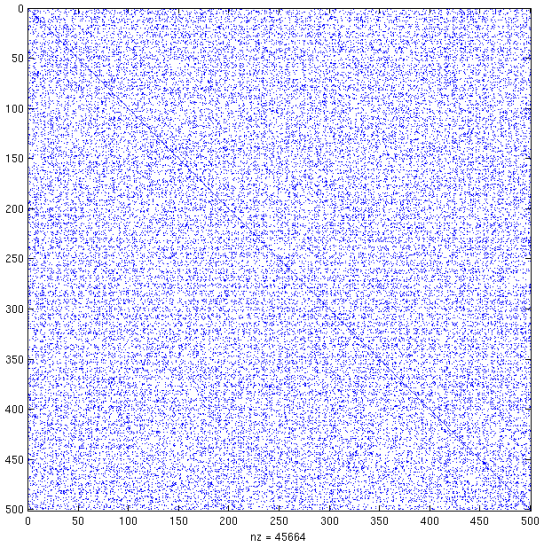


In tal caso usando il FOM si ha che l'errore decresce esponenzialmente



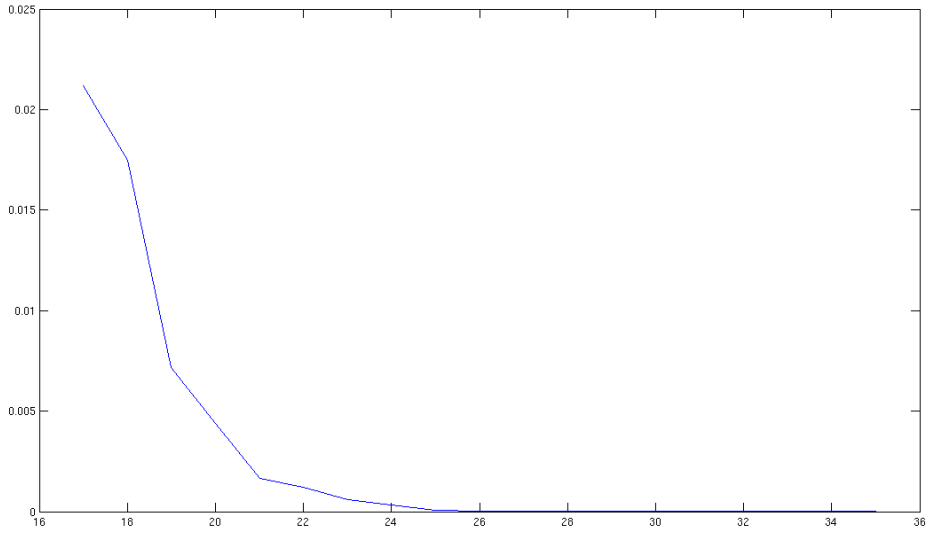
m	$\|Ax - b\|_2$
5	8.024719454927496
10	$6.011679642212282e-01$
15	$4.560460283575539e-02$
20	$5.144164659678912e-03$
25	$1.391573488500990e-04$
30	$5.712172046984290e-06$
35	$9.168713084658108e-11$
40	$1.250658762108303e-12$

Esempio 3.2. Si consideri $A \in \mathbb{R}^{500\times 500}$ con una densità del 20%



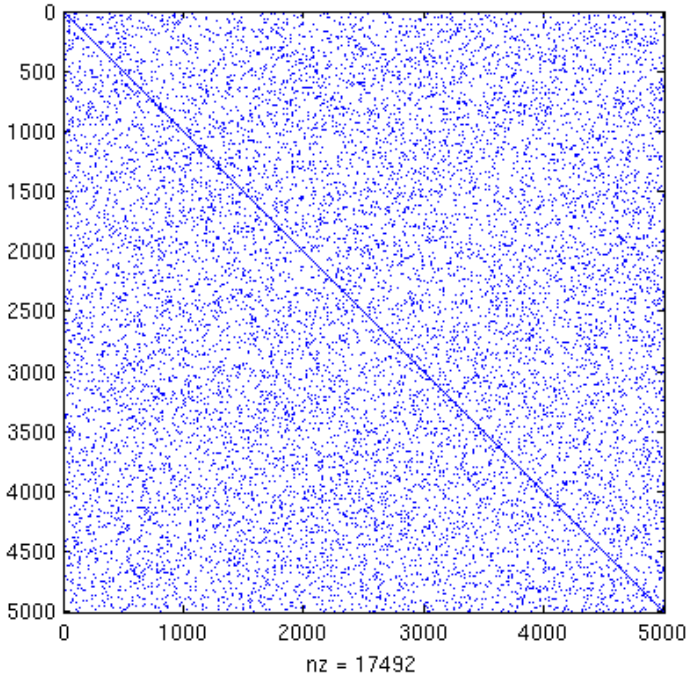
In tal caso l'errore decresce come prima in modo esponenziale

In tal caso usando il FOM si ha che l'errore decresce esponenzialmente

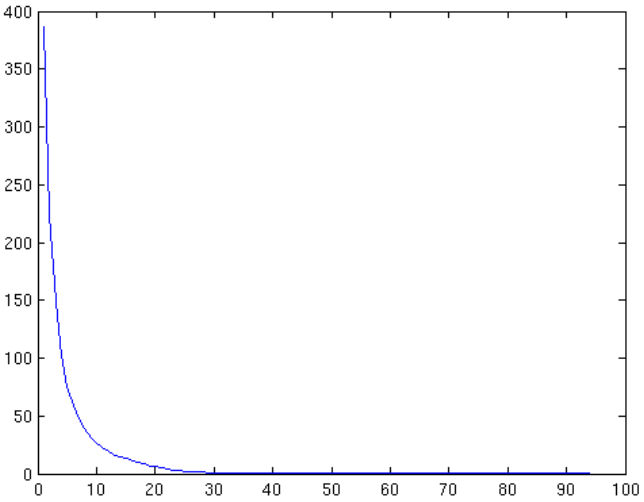


Già per $m = 30$ si ha che $\|Ax - b\| = 3.407057040608546e - 06$

Esempio 3.3. Intuitivamente quello che ci si aspetta è che al crescere di m la precisione dell'approssimazione aumenta (si deve comunque considerare che l'approssimazione ottenuta dipende anche dal punto iniziale) quindi ci si chiede qual'è l'm ottimale, o per lo meno risulta chiaro che tanto più grande è m tanto più aumenta il costo computazionale e la memoria, in particolare se si aumenta la dimensione della matrice e si diminuisce la spasità si può osservare che m cresce notevolmente. Si supponga di volere un'approssimazione dell'ordine di 10^{-6}, se si considera $A \in \mathbb{R}^{5000\times 5000}$ con una densità del 0.05% allora



Per avere tale precisione è stato necessario $m = 94$



Se si prova a scegliere n ancora più grande l'algoritmo diventa inutile dato che m diventa anh'esso più grande, quindi è necessario raffinare ulteriormente il FOM ed introdurre una prima variante.

3.3 Metodo di ortogonalizzazione completa con Restart (RFOM)

La prima variante è denominata RFOM (Restarted Full Orthogonalization Method)

Algorithm 3 MGS

Si sceglie un vettore x_0 approssimazione iniziale
Si pone $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$ e $v_1 = r_0/\beta$
Si genera la matrice H_m con l'algoritmo di Arnoldi partendo dal vettore v_1
Calcolare $y_m = H_m^{-1}\beta e_1$ e $x_m = x_0 + V_m y_m$
if $\|b - Ax_m\|_2 <$ tol **then**
  stop
else
  $x_0 = x_m$ e ricominciare
end if

Quindi l'idea è di usare FOM per avvicinarsi alla soluzione con m piccolo e ricominciare FOM dall'approssimazione ottenuta col passo precedente.

3.4 Sperimentazione numerica su RFOM

Di seguito c'è l'implementazione in matlab di RFOM

Listing 2: RFOM

```
function x = RFOM( A, b, x0, m, numit )
% implementazione matlab di RFOM

x=x0;
for i=1:1:numit
```

```
x=FOM(A, b, x, m);

end

end
```

Esempio 3.4. Se ad esempio si considera $A \in \mathbb{R}^{10000\times 10000}$ con densità del 0.05% e si supponga di volere un'approssimazione della soluzione con un ordine 10^{-6}, il problema è inattaccabile con FOM ma con il RFOM sono sufficienti 75 iterazioni con $m = 3$.

3.5 Metodo di ortogonalizzazione incompleta (IOM)

L'idea è di non completare il processo di ortogonalizzazione, ovvero ogni vettore v_j sarà ortogonale solo ai k vettori precedenti, ovvero ai v_{j-k+i} per $i = 0, \dots, k-1$. Per il resto l'algoritmo è esattamente lostesso. Di seguico c'è l'algoritmo di ortogonalizzazione incompleta (IO).

Algorithm 4 IO

Si sceglie un vettore v_1 di norma unitaria
for $j = 1, 2, \dots, m$ **do**
  calcolo $w_j := Av_j$
  for $i = max(1, j-k+1), \dots, j$ **do**
    calcolo $h_{i,j} = (w_j, v_i)$
    calcolo $w_j = w_j - h_{i,j}v_i$
  end for
  $h_{j+1,j} = \|w_j\|_2$
  if $h_{j+1,j} = 0$ **then**
    stop
  else
    $v_{j+1} = w_j/h_{j+1,j}$
  end if
end for

Dopo aver eseguito IO, come nel caso precedente è sufficiente calcolare $y_m = H_m^{-1}\beta e_1$ e $x_m = x_0 + V_m y_m$.

Osservazione 3.3. Per calcolare H_m ed eseguire il calcolo dell'ortogonalizzazione incompleta non è necessario memorizzare tutti i $v_1, \dots, v_m$ ma è sufficiente memorizzarne k per volta, questo da un notevole risparmio computazionale, ma bisogna tenere a mente che ai fini del calcolo di x_m è necessario conoscere V_m, quindi l'idea potrebbe essere di costruire la matrice H_m, calcolare y_m e per il calcolo del prodotto $V_m y_m$ riavviare l'algoritmo e ricalcolare nuovamente i v_i e via via eseguire i prodotti per calcolare $V_m y_m$. Questo abbassa il costo computazionale e quello che non era possibile fare con FOM lo si può con IOM, si eviterà di soffermarsi troppo su questa variante dato che si può migliorare come si vedrà nel prossimo paragrafo.

3.6 Sperimentazione numerica su IOM

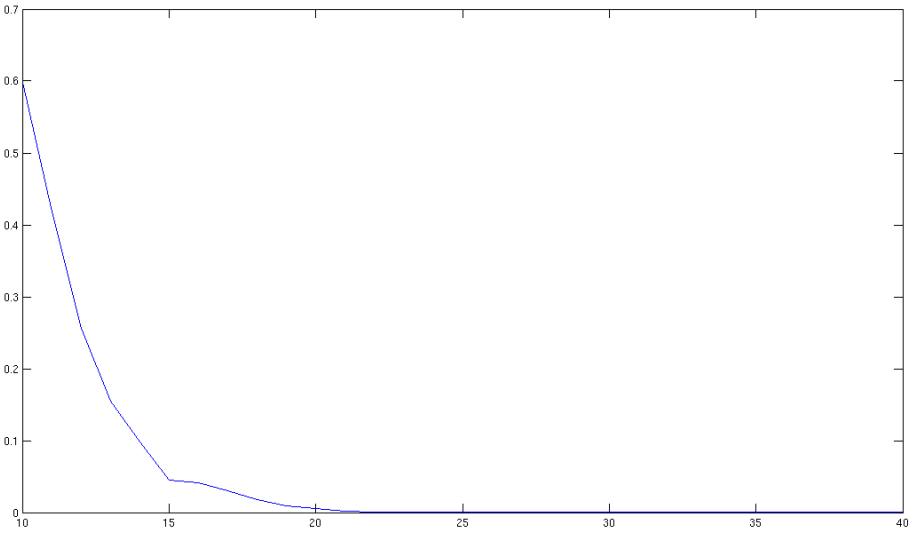
Si mostra un'implementazione in matlab ingenua, non si tiene conto per nulla dell'osservazione appena fatta, si memorizzano tutti i $v_1, \ldots, v_m$, in tal caso si abbassa il costo computazionale rispetto FOM ma il problema della memoria resta.

Listing 3: IOM

```
function xm = IOM( A, b, x0, m, max_ort )
%IOM Summary of this function goes here
% Implementazione in matlab di IOM



% inizializzazione
n=size(A,1);
r0= b - A*x0;
beta=norm(r0);
V(:,1)=r0/beta;
H=0;

for j=1:1:m
    wj=A*V(:,j);

    for i=max(j-max_ort+1,1):1:j
        H(i,j)=wj'*V(:,i);
        wj=wj-H(i,j)*V(:,i);
    end

    H(j+1,j)=norm(wj);

    if (H(j+1,j)==0)
        'breakdown happened'
        m=j;
        ym=[beta;zeros(m-1,1)];
        H=H(1:m,1:m);
        ym=inv(H)*ym;
        V=V(1:n,1:m);
        xm=x0+V*ym
    else

        V(:,j+1)=wj/H(j+1,j);
    end
end

ym=[beta;zeros(m-1,1)];
H=H(1:m,1:m);
if(abs(det(H))<10^(-10))
    xm=zeros(n,1);
else
```

```
    ym=H\ym;
    V=V(1:n,1:m);
    xm=x0+V*ym;
end

end
```

Esempio 3.5. Come nell'esempio 3.1 si considera $A \in \mathbb{R}^{100\times 100}$ con una sparsità del 10%, allora si pone $k = 3$ (si ortogonalizza solo rispetto ai 3 vettori precedenti), in tal caso si osserva che l'errore decresce esponenzialmente



In particolare dopo 44 iterazioni si ottiene un errore dell'ordine di 10^{-8}.

3.7 Metodo di ortogonalizzazione incompleta diretta (DIOM)

Si consideri l'IOM, la matrice di Hessenberg H_m ottenuta tramite l'ortogonalizzazione incompleta è una matrice a banda, si supponga $m = 5, k = 3$ allora

$$H_m = \begin{pmatrix} h_{1,1} & h_{1,2} & h_{1,3} & & \\ h_{2,1} & h_{2,2} & h_{2,3} & h_{2,4} & \\ & h_{3,2} & h_{3,3} & h_{3,4} & h_{3,5} \\ & & h_{4,3} & h_{4,4} & h_{4,5} \\ & & & h_{5,4} & h_{5,5} \end{pmatrix}$$

Si consideri dunque la fattorizzazione LU della matrice H_m, quindi $H_m = L_m U_m$, dunque L_m sarà una matrice bigiagonale ed U_m sarà una matrice triangolare banda.

$$H_m = \begin{pmatrix} 1 & & & & \\ l_{2,1} & 1 & & & \\ & l_{3,2} & 1 & & \\ & & l_{4,3} & 1 & \\ & & & l_{5,4} & 1 \end{pmatrix} \begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} & & \\ & u_{2,2} & u_{2,3} & u_{2,4} & \\ & & u_{3,3} & u_{3,4} & u_{3,5} \\ & & & u_{4,4} & u_{4,5} \\ & & & & u_{5,5} \end{pmatrix}$$

Quindi la soluzione approssimata sarà data da

$$x_m = x_0 + V_m U_m^{-1} L_m^{-1}(\beta e_1)$$

Allora si definisce

$$P_m := V_m U_m^{-1}$$

e

$$z_m := L_m^{-1}(\beta e_1)$$

Quindi l'approssimazione sarà data da

$$x_m = x_0 + P_m z_m$$

A questo punto si osserva che grazie alla struttura di U_m e P_m, queste possono essere aggiornate in modo molto semplice. Infatti si ha che $P_m U_m = V_m$, leggendo questa espressione sulla sola ultima colonna si ottiene

$$\sum_{i=m-k+1}^{m} u_{i,m} p_i = v_m$$

Da questa si ottiene

$$p_m = \frac{1}{u_{m,m}} \left[v_m - \sum_{i=m-k+1}^{m-1} u_{i,m} p_i \right]$$

Inoltre, grazie alla struttura di L_m si ha che

$$z_m = \begin{pmatrix} z_{m-1} \\ \zeta_m \end{pmatrix}$$

dove

$$\zeta_m = -l_{m,m-1} \zeta_{m-1}$$

Quindi, mettendo tutto insieme si ottiene

$$\begin{aligned} x_m &= x_0 + (P_{m-1}, p_m) \begin{pmatrix} z_{m-1} \\ \zeta_m \end{pmatrix} \\ &= x_0 + P_{m-1} z_{m-1} + \zeta_m p_m \end{aligned}$$

Daltronde si ha che

$$x_0 + P_{m-1} z_{m-1} = x_{m-1}$$

Allora segue che l'approssimazione x_m può esser aggiornata ad ogni passo

$$x_m = x_{m-1} + \zeta_m p_m$$

E' finalmente possibile esprimere l'algoritmo di ortogonalizzazione incompleta diretta (DIOM).

Algorithm 5 DIOM

Si sceglie una stima iniziale x_0 e si calcola $r_0 = b - Ax_0$, $\beta := \|r_0\|_2$ e $v_1 = r_0/\beta$
for $m = 1, 2, \ldots,$ convergenza **do**
  calcolo a partire da H_{m-1} calcolo H_m (seguendo l'algoritmo IO)
  (*) Aggiorno le matrici L_m, U_m
  calcolo $w_j = w_j - h_{i,j}v_i$
  if $u_{m,m} = 0$ **then**
    stop
  else
    if $m = 1$ **then**
      $\zeta_m = \beta$
    else
      $\zeta_m = -l_{m,m-1}\zeta_{m-1}$
    end if
    $p_m = u_{m,m}^{-1}(v_m - \sum_{i=m-k+1}^{m-1})u_{i,m}p_i$
    $x_m = x_{m-1} + \zeta_m p_m$
  end if
end for

Come si può intuire bisogna argomentare (*), non è infatti ovvio come aggiornare le matrici L_m ed U_m, daltronde bastano delle semplici osservazioni. Per L_m il conto è immediato, infatti vale la relazione $H_m = L_m U_m$ ed $H_{m-1} = L_{m-1}U_{m-1}$, supponendo di conoscere H_m, L_{m-1}, U_{m-1} allora si trova subito la relazione

$$l_{m,m-1}u_{m-1,m-1} = h_{m,m-1}$$

Quindi si riesce a costruire L_m, per determinare U_m invece si usa $U_m = L_m^{-1}H_m$, inoltre data la struttura triangolare di U_m è sufficiente calcolare solo la colonna m-esima. Daltronde è semplice invertire L_m, infatti si può mostrare per induzione che

$$L_m^{-1} = (\tilde{l}_{i,j})_{i,j=1}^m \qquad \text{con} \qquad \tilde{l}_{i,j} = (-1)^{i+1}\prod_{k=i}^{j} l_{k+1,k}$$

Proposizione 3.4. Vale

$$\begin{aligned}\|b - Ax_m\|_2 &= h_{m+1.m}|e_m^T y_m| \\ &= h_{m+1,m}\frac{|\zeta_m|}{|u_{m,m}|}\end{aligned}$$

Proposizione 3.5. IOM e DIOM sono equivalenti al processo di proiezione in K_m imponendo l'ortogonalità allo spazio

$$L_m = \text{span}\{z_1, z_2, \ldots, z_m\}$$

dove

$$z_i = v_i - (v_i, v_{m+1})v_{m+1} \qquad \text{per} \qquad i = 1, \ldots, m$$

Quest'ultimo risultato garantisce la convergenza di IOM e DIOM.

4 GMRES

Come già anticipato il GMRES (Generalized Minimum Residual Method) è un metodo per preoiezione dove si pone $K = K_m$ e $L = AK_m$.

4.1 Algoritmo di base

Si inizia con la descrizione di base del GMRES per poi presentare qualche variante. Si osserva che se $x \in x_0 + K_m$ allora

$$x = x_0 + V_m y \qquad \text{per qualche } y \in \mathbb{R}^m$$

Allora si definisce

$$\begin{aligned} J(y) &= \|b - Ax\|_2 \\ &= \|b - A(x_0 + V_m y)\|_2 \end{aligned}$$

Allora sfruttando le relazioni tra V_m ed A citate nella sezione precedente si ha

$$\begin{aligned} b - Ax &= b - A(x_0 + V_m y) \\ &= r_0 - AV_m y \\ &= \beta v_1 - V_{m+1}\bar{H}_m y \\ &= V_{m+1}(\beta e_1 - \bar{H}_m y) \end{aligned}$$

Dato che le colonne di V_{m+1} sono ortonormali allora

$$\begin{aligned} J(y) &= \|b - A(x_0 + V_m y)\|_2 \\ &= \|\beta e_1 - \bar{H}_m y\|_2 \end{aligned}$$

Dato che l'obbiettivo è determinare $x \in x_0 + K_m$ che minimizza $\|b - Ax\|_2$ allora per quanto appena visto si ha che l'approssimazione è data da

$$\begin{aligned} x_m &= x_0 + V_m y_m \qquad \text{dove} \\ y_m &= \min_{y \in \mathbb{R}^m} \|\beta e_1 - \bar{H}_m y\|_2 \end{aligned}$$

Determinare l'y_m in genere non è troppo costoso dato che si sceglie m piccolo, quindi si tratta di risolvere un problema ai minimi quadrati in bassa dimensione.

Algorithm 6 GMRES

Si sceglie una stima iniziale x_0 e si calcola $r_0 = b - Ax_0$, $\beta := \|r_0\|_2$ e $v_1 = r_0/\beta$
for $j = 1, \ldots, m$ **do**
  $w_j = Av_j$
  for $i = 1, \ldots, j$ **do**
    $h_{i,j} = (w_j, v_i)$
    $w_j = w_j - h_{i,j}v_i$
  end for
  $h_{j+1,j} = \|w_j\|_2$
  if $h_{j+1,j} = 0$ **then**
    Stop
  else
    $v_{j+1} = w_j/h_{j+1,j}$
  end if
end for
$\bar{H} = (h_{i,j})_{1 \leq i \leq m+1, 1 \leq j \leq m}$
Risolvere il problema ai minimi quadrati $\|\beta e_1 - \bar{H}y\|_2$
$x_m = x_0 + V_m y_m$

Osservazione 4.1. Un primo difetto dell'algoritmo è quello che non fornisce passo per passo l'errore, quindi non si conosce quando buona è l'approssimazione x_m a meno di non calcolare ad ogni passo $\|b - Ax_m\|_2$, daltronde si può risolvere questo problema con delle semplici osservazioni. Tutto dipende da come viene risolto il problema ai minimi quadrati, se si utlizzano le rotazioni di Givens è possibile, con qualche conto, avere una stima del residuo nel problema ai minimi quadrati e con qualche altro conto questa fornisce una stima nell'errore ad ogni passo, ad ogni modo per tutti i dettagli tecnici si rimanda alla referenza principale.

4.2 Sperimentazione numerica su GMRES

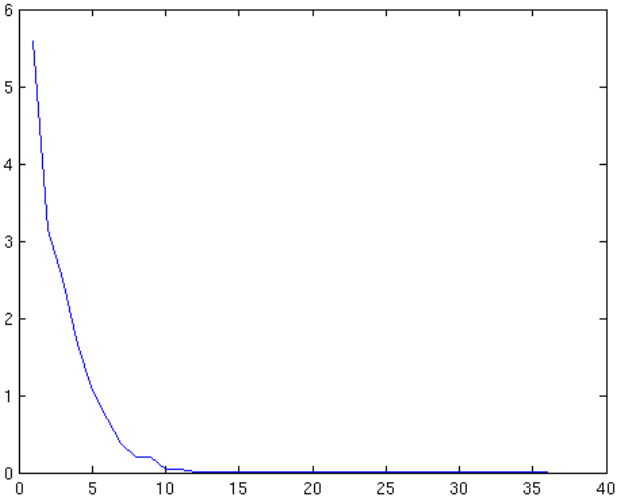
Di seguito c'è l'implementazione in matlab del GMRES

Listing 4: GMRES

```
function xm = GMRES( A, b, x0, m )

% implementazione matlab di GMRES

n=size(A,1)

r0=b-A*x0;
beta=norm(r0);
V(:,1)=r0/beta;

for j=1:1:m
    wj=A*V(:,j);
```

```
    for i=1:1:j
        H(i,j)=wj'*V(:,i);
        wj=wj-H(i,j)*V(:,i);
    end

    H(j+1,j)=norm(wj);

    V(:,j+1)=wj/H(j+1,j);

end

ym=lsqr(H,[beta;zeros(m,1)],0,1000);

V=V(1:n,1:m);

xm=x0+V*ym;

end
```

Esempio 4.1. Si considera lo stesso esempio esaminato con il FOM, sia $A \in \mathbb{R}^{100\times 100}$ con una sparsità del 10% allora anche questa volta l'errore decresce esponenzialmente



m	$\|Ax-b\|_2$
5	5.085520882343868
10	$4.077795664386280e-01$
15	$7.811617245478521e-03$
20	$6.092949161539471e-04$
25	$1.925114102011914e-05$
30	$2.668101141093672e-07$
35	$6.930037285762242e-09$
40	$1.148568647261571e-10$

Da questo primo esempio sembrerebbe che ci vogliano più iterazioni con GMRES rispetto FOM.

Esempio 4.2. Si considera ora $A \in \mathbb{R}^{5000\times5000}$ con una sparsità del 0.05%, se si vuole un'approssimazione dell'ordine di 10^{-6} allora è sufficiente $m = 79$ (per il FOM invece $m = 94$).

4.3 GMRES con Restart (RGMRES)

Similmente al metodo di Arnoldi quando m diventa troppo grande il $GMRES$ è impraticabile, quindi l'idea è come prima fare m passi con la versione base del GMRES per avvicinarsi alla soluzione e ricominciare da capo scegliendo come approssimazione iniziale l'ultima trovata.

Algorithm 7 RGMRES

Si calcola $r_0 = b - Ax_0$, $\beta := \|r_0\|_2$ e $v_1 = r_0/\beta$
Si genera una base di Arnoldi e la matrice $\bar{H}_m$ usando l'algoritmo di Arnoldi partendo da v_1
Si risolve il problema ai minimi quadrati $\bar{H}_m y_m = \beta e_1$ e si pone $x_m = x_0 + V_m y_m$
if Il residuo è sufficientemente piccolo **then**
  Stop
else
  Si ricomincia con $x_0 = x_m$
end if

4.4 Sperimentazione numerica su RGMRES

Di seguito c'è il codice matlab per l' RGMRES

Listing 5: RGMRES

```
function x = RGMRES( A, b, x0, m, numit )
% implementazione matlab di RGMRES

x=x0;
for i=1:1:numit

x=GMRES(A, b, x, m);

end

end
```

Esempio 4.3. Si consideri $A \in \mathbb{R}^{10000\times10000}$ con una sparsità del 0.05%, se si vuole una precisione dell'ordine di 10^{-6}, con $m = 3$ sono necessarie circa 72 iterazioni.

4.5 Cenni sul Quasi GMRES (QGMRES) e sul Quasi GMRES Diretto (DQGMRES)

E' possibile ripetere le stesse e identiche argomentazioni fatte per IOM e DIOM per il GMRES, infatti è sufficiente fare una ortogonalizzazione incompleta ed eseguire il GMRES in modo usuale. In analogia al

DIOM si può osservare che è possibile risparmiare oltre che computazione anche memoria facendo delle osservazioni su $\bar{H}_m$. Si evita di ripetere il tutto dato che l'idea di base è esattamente la stessa usata nel IOM e DIOM, ad ogni modo per ulteriori dettagli si rimanda alla referenza principale.

5 Metodo di Biortogonalizzazione di Lanczos

I metodi di Arnoldi e i GMRES sono detti metodi di proiezione ortogonale a causa della condizione di Petrov-Galerkin, se si rimuove questa condizione si hanno i metodi per proiezione obliqua, uno di questi è ad esempio il metodo di Lanczos.

5.1 Algoritmo di biortogonalizzazione di Lanczos

Definizione 5.1. Dati due sottospazi V e W di dimensione m, due basi $\{v_1, \dots, v_m\}$ e $\{w_1, \dots, w_m\}$ (rispettivamente di V e W) si dicono biortogonali se $(v_i, w_j) = \delta_{i,j}$

L'obbiettivo è quello di calcolare una coppia di basi biortogonali dei seguenti sottospazi di Krylov

$$\begin{aligned} K_m(A, v_1) &= \text{span}\left\{v_1, Av_1, \dots, A^{m-1}v_1\right\} \\ K_m(A^T, w_1) &= \text{span}\left\{w_1, A^Tw_1, \dots, (A^T)^{m-1}w_1\right\} \end{aligned}$$

Algorithm 8 Biortogonalizzazione di Lanczos

Scegliere due vettori v_1 e v_2 tali che $(v_1, w_1) = 1$
Si pone $\beta_1 = \delta_1 = 0$ e $w_0 = v_0 = 0$
for $j = 1, \dots, m$ **do**
  $\alpha_j = (Av_j, w_j)$
  $\hat{v}_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1}$
  $\hat{w}_{j+1} = A^T w_j - \alpha_j w_j - \delta_j w_{j-1}$
  $\delta_{j+1} = |(\hat{v}_{j+1}, \hat{w}_{j+1})|^{1/2}$
  if $\delta_{j+1} = 0$ **then**
    Stop
  end if
  $\beta_{j+1} = (\hat{v}_{j+1}, \hat{w}_{j+1})/\delta_{j+1}$
  $w_{j+1} = \hat{w}_{j+1}/\beta_{j+1}$
  $v_{j+1} = \hat{v}_{j+1}/\delta_{j+1}$
end for

A questo punto si considera la matrice

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \delta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \delta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & \delta_m & \alpha_m \end{pmatrix}$$

Proposizione 5.1. Se l'algoritmo termina con successo allora i vettori $v_i \in K_m(A, v_1)$ e $w_j \in K_m(A^T, w_1)$ e le due basi sono biortogonali. Inoltre valgono le seguenti relazioni

$$\begin{aligned} AV_m &= V_m T_m + \delta_{m+1} v_{m+1} e_m^T \\ A^T W_m &= W_m T_m^T + \beta_{m+1} w_{m+1} e_m^T \\ W_m^T A V_m &= T_m \end{aligned}$$

5.2 Metodo di Lanczos per la risoluzione di sistemi lineari

A questo punto è possibile risolvere un sistema lineare utilizzando i risultati enunciati sin ora.

Algorithm 9 Metodo di Lanczos

Si calcola $r_0 = b - Ax_0$ e $\beta = \|r_0\|_2$

Si pone $v_1 = r_0/\beta$ e w_1 tale che $(v_1, w_1) = 1$

Si generano i vettori $v_1, \ldots, v_m$ e $w_1, \ldots, w_m$ e la matrice T_m utilizzando il metodo di biortogonalizzazione di Lanczos

Si calcola $y_m = T_m^{-1}(\beta e_1)$ e $x_m = x_0 + V_m y_m$

Inoltre per il residuo vale

$$\|b - Ax_j\|_2 = |\delta_{j+1} e_j^T y_j| \|v_{j+1}\|_2$$

5.3 Sperimentazione numerica sul metodo di Lanczos

Di seguito c'è l'implementazione in matlab

Listing 6: BIORT

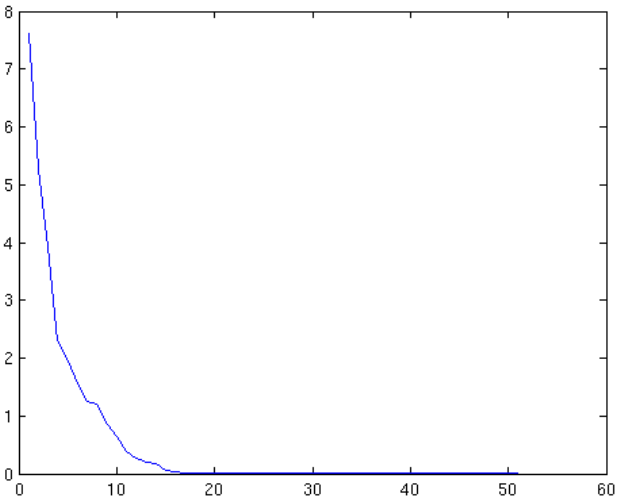
```
function [ alpha, beta, delta , v] = BIORT( A, m, v1, w1)
%BIORT Summary of this function goes here
% implementazione matlab di BIORT

n=size(A,1);

beta(1)=0;
delta(1)=0;

beta(2)=0;
delta(2)=0;

w(:,1)=zeros(n,1);
v(:,1)=zeros(n,1);

w(:,2)=w1;
v(:,2)=v1;

for j=2:1:m+1

   alpha(j)=(A*v(:,j))'*w(:,j);
```

```
    aux1=A*v(:,j)-alpha(j)*v(:,j)-beta(j)*v(:,j-1);
    aux2=(A')*w(:,j)-alpha(j)*w(:,j)-delta(j)*w(:,j-1);

    delta(j+1)=sqrt(abs((aux1'*aux2)));
    beta(j+1)=(aux1'*aux2)/delta(j+1);
    w(:,j+1)=aux2/beta(j+1);
    v(:,j+1)=aux1/delta(j+1);

end

v=v(:,2:m+1);
alpha=alpha(2:m+1);
beta=beta(3:m+1);
delta=delta(3:m+1);

end
```

Listing 7: LANCZOS

```
function xm = LANCZOS(A, b, x0, m)
%LANCZOS Summary of this function goes here
% implementazione matlab di LANCZOS

n=size(A,1);

r0=b-A*x0;
aux=norm(r0);

v1=r0/aux;
w1=rand(n,1);
w1=w1/(v1'*w1);

[alpha,beta,delta, v]=BIORT(A,m,v1,w1);

T=diag(alpha)+diag(beta,1)+diag(delta,-1);

ym=T\([aux;zeros(m-1,1)]);
xm=x0+v*ym;

end
```

Esempio 5.1. Si consideri come sempre $A \in \mathbb{R}^{100\times 100}$ con una densità del 10%, anche in questo caso l'errore decresce esponenzialmente



m	$\|Ax - b\|_2$
5	7.065326812822373
10	1.603750665730710
15	$3.004002605116352e-01$
20	$3.662663067288966e-02$
25	$1.676281920601705e-03$
30	$1.103868948629327e-04$
35	$3.207716999908961e-06$
40	$7.514080356615319e-08$
45	$8.181026843615434e-10$
50	$2.331463497119234e-11$
55	$1.273142051279545e-13$

Esempio 5.2. Con $A \in \mathbb{R}^{5000\times5000}$ con una sparsità del 0.05% si trova che sono sufficienti 95 iterazioni per avere una stima della soluzione dell'ordine di 10^{-6}.

6 Conclusioni

Sono state presentate solo le idee generali di alcuni dei metodi più utilizzati per risolvere iterativamente sistemi lineari sparsi. Sono molti gli aspetti interessanti legati alla computazione delle matrici sparse, pultroppo non è possibile una trattazione completa in poche pagine, inoltre andando avanti il tutto diventa estremamente tecnico. Uno degli aspetti più interessanti è che nonostante la tecnologia vada avanti e la capacità di calcolo aumenti in continuazione, risulta impossible risolvere in modo diretto questi problemi, quindi costantemente sarà necessario studiare e raffinare gli algoritmi attualmente esistenti e cercarne di nuovi.