GMRES e BICGSTAB: due metodi iterativi per risolvere sistemi lineari di grandi dimensioni

Elena Gaburro

Università degli studi di Verona
Master's Degree in Mathematics and Applications

29 aprile 2013

Consideriamo un sistema lineare $Ax = b$, dove

- $A \in \mathbb{R}^{n \times n}$ è la matrice dei coefficienti: la supponiamo non singolare
- $b \in \mathbb{R}^n$ è il vettore termine noto
- $x \in \mathbb{R}^n$ è il vettore incognito

Per risolverlo esistono diverse tecniche:

- Regola di Cramer: costo computazionale $(n+1)!$, si usa solo per matrici di dimensioni molto piccole.
- Metodi Diretti: Sono quelli in cui si cerca di fattorizzare la matrice A nel prodotto di matrici triangolari per le quali la risoluzione del sistema è più rapida. Esempi: fattorizzazione *LU* per una matrice qualsiasi, fattorizzazione di *Cholesky* per una matrice simmetrica definita positiva, algoritmo di Thompson per matrici tridiagonali ...

E se la matrice è SPARSA...

Introduzione

Se la matrice A è sparsa è possibile memorizzare i suoi elementi con un costo limitato: $O(n)$ invece di $O(n^2)$.
In generale però, se una matrice è sparsa, non è detto che lo siano anche i suoi fattori e quindi non è consigliabile usare metodi che prevedano la fattorizzazione della matrice.

Si usano quindi i metodi iterativi

A partire da una soluzione approssimata, questi metodi ne modificano, ad ogni iterazione, una o più componenti, con lo scopo di minimizzare la norma del residuo.
(Se x_m è la soluzione al passo m si vuole minimizzare $Ax_m - b$).

Esempi base di metodi iterativi sono: Jacobi, Gauss Saidel, Sor (ma...)

I Metodi Iterativi

La maggior parte delle tecniche iterative per risolvere sistemi lineari di grandi dimensioni sono basati su metodi proiettivi, i quali estraggono un'approssimazione della soluzione da un particolare sottospazio di $\mathbb{R}^n$.

- Chiamiamo $\mathcal{K}$ il sottospazio di dimensione m dove cerchiamo la soluzione approssimante.
- Per determinarla in modo univoco dovremo imporre m vincoli: in generale imporremo che il residuo $b - Ax$ sia ortogonale a m vettori linearmente indipendenti, i quali definiscono un altro sottospazio di dimensione m, che indicheremo con $\mathcal{L}$.
- Le condizioni imposte in questo modo sono dette di Petrov-Galerkin
- I metodi proiettivi si differenziano tra loro per la scelta di $\mathcal{K}$ e $\mathcal{L}$: in particolare si parla di metodi proiettivi ortogonali se $\mathcal{K} = \mathcal{L}$ e di quelli obliqui se $\mathcal{L}$ è diverso da $\mathcal{K}$.

Se poi vogliamo cercare una nuova soluzione approssimante $\tilde{x}$ a partire da un'approssimazione precedente x_0 dobbiamo introdurre qualche modifica:

- Cerchiamo $\tilde{x}$ nello spazio affine $x_0 + \mathcal{K}$ in modo che $b - A\tilde{x} \perp \mathcal{L}$
- Poi scrivendo $\tilde{x}$ come $x_0 + \delta$, con $r_0 = b - Ax_0$ il residuo iniziale, la condizione di ortogonalità diventa

$$b - A(x_0 + \delta) \perp \mathcal{L}$$
$$r_0 - A\delta \perp \mathcal{L}$$

- La soluzione si trova quindi risolvendo

$$\boxed{\begin{aligned} &\tilde{x} = x_0 + \delta \quad \delta \in \mathcal{K} \\ &(r_0 - A\delta, \omega) = 0 \quad \forall \omega \in \mathcal{L} \end{aligned}}$$

Siano $V = [v_1, \ldots, v_m]$ e $W = [w_1, \ldots, w_m]$ matrici $n \times m$ le cui colonne formano rispettivamente una base di $\mathcal{K}$ e di $\mathcal{L}$:

- la soluzione approssimante può essere scritta come $\tilde{x} = x_0 + Vy$
- la condizione di ortogonalità diventa $(W^T AV)y = W^T r_0$
- quindi

$$\tilde{x} = x_0 + V(W^T AV)^{-1} W^T r_0$$

Note:

- In molti algoritmi la matrice $W^T AV$ non deve essere calcolata realmente, ma è disponibile come prodotto matrice-vettore
- La matrice $W^T AV$ non è singolare se e solo se nessun vettore del sottospazio $A\mathcal{K}$ è ortogonale a $\mathcal{L}$

Nel caso in cui $\mathcal{L} = A\mathcal{K}$ si ottiene facilmente il seguente risultato

$\tilde{x}$ è il risultato di un metodo proiettivo su $\mathcal{K}$ ortogonalmente ad $\mathcal{L}$, con x_0 l'approssimazione iniziale, se e solo se minimizza la norma euclidea del residuo $b - Ax$, cioè

$$R(\tilde{x}) = \min_{x \in x_0 + \mathcal{K}} R(x)$$

dove $R(x) = \|b - Ax\|_2$.

Il metodo *GMRES* si ottiene proprio usando $\mathcal{L} = A\mathcal{K}$, dove $\mathcal{K}$ è un sottospazio di Krylov.

Sottospazi di Krylov

Definizione (Sottospazi di Krylov)

Il sottospazio di Krylov di dimensione m $\mathcal{K}_m(A, r_0)$ è lo spazio generato da vettori della forma $p(A)r_0$, dove p è un polinomio, cioè generato da $\{r_0, Ar_0, A^2 r_0, \ldots, A^{m-1} r_0\}$.

Però questa base di $\mathcal{K}$ è numericamente instabile: al crescere di m infatti i vettori tendono a diventare linearmente dipendenti.

Per questo è necessario estrarre una base ortonormale: per trovarla usiamo il Metodo di Arnoldi

Si basa sull'algoritmo di Gram-Schmidt Modificato (usiamo direttamente quello modificato perchè risulta più affidabile nel contenere errori dovuti alle cancellazioni numeriche)

Metodo di Arnoldi

1: Scegli un vettore v_1 t.c. $\|v_1\|_2 = 1$
2: **for** $j = 1, 2, \ldots, m$ **do**
3: Calcola $w_j := Av_j$
4: **for** $i = 1, 2, \ldots, j$ **do**
5: $h_{ij} := (w_j, v_i)$
6: $w_j := w_j - h_{ij} v_i$
7: **end for**
8: $h_{j+1,j} = \|w_j\|_2$.
9: **if** $h_{j+1,j} = 0$ **then** STOP
10: **end if**
11: $v_{j+1} = w_j / h_{j+1,j}$
12: **end for**

COSTO

riga 3: prodotto matrice-vettore (m volte)

riga 5 e 6: 2 prodotti ($\frac{m^2}{2}$ volte)

totale:

m (1 mat-vet) + $\frac{m^2}{2}$ (2 prod)

Dall'algoritmo si ricava la matrice V_m, base ortonormale di $\mathcal{K}_m$ e la matrice di Hessenberg $\bar{H}_m$ $(m+1)$ x m.

GMRES

Il GMRES (Generalised Minimum Residual Method) è un metodo proiettivo dove

- $\mathcal{K} = \mathcal{K}_m$ è il sottospazio di *Krylov* di dimensione m
- $\mathcal{L}_m = A\mathcal{K}_m$

L' algoritmo:

- prima costruisce una base ortonormale di $\mathcal{K}_m$, V_m, attraverso il metodo di Arnoldi
- in modo che la soluzione cercata possa essere scritta come $\tilde{x} = x_0 + V_m y_m$.
- Poi calcola y_m in modo che minimizzi il residuo $b - A\tilde{x}$.

Si può mostrare che $b - A\tilde{x} = V_{m+1}(\beta e_1 - \bar{H}_m y_m)$, con $\beta = \|r_0\|_2$.
Quindi, ricordando che V_{m+1} è una matrice ortogonale, si cerca y_m in modo che minimizzi

$$\| \beta e_1 - \bar{H}_m y_m \|_2$$

GMRES

1: Calcola $r_0 = b - Ax_0$, $\beta := \|r_0\|_2$ e $v_1 := r_0/\beta$
2: **for** $j = 1, 2, \ldots, m$ **do**
3: Calcola $w_j := Av_j$
4: **for** $i = 1, 2, \ldots, j$ **do**
5: $h_{ij} := (w_j, v_i)$
6: $w_j := w_j - h_{ij} v_i$
7: **end for**
8: $h_{j+1,j} = \|w_j\|_2$.
9: **if** $h_{j+1,j} = 0$ **then** poni $m := j$ e vai alla 13
10: **end if**
11: $v_{j+1} = w_j / h_{j+1,j}$
12: **end for**
13: Definisci $\bar{H}_m = h_{ij}$
14: Calcola y_m t.c. minimizzi $\| \beta e_1 - \bar{H}_m y_m \|_2$
15: $x_m = x_0 + V_m y_m$

Miglioramento: Calcolare il residuo ad intervalli regolari

In questo modo l'algoritmo può terminare non appena $\mathcal{K}$ raggiunge la dimensione sufficiente per la tolleranza richiesta.

Costi:

Il costo dell'algoritmo è quello dovuto al metodo di Arnoldi (infatti poichè m, in generale, è piccolo rispetto alle dimensioni della matrice, risolvere il problema di minimo per trovare y_m non è dispendioso).
Quindi una stima del costo in tempo si può ottenere da

$$m(\text{mat-vet}) + m^2(\text{prod})$$

Inoltre, bisogna notare che per calcolare una nuova soluzione è necessario memorizzare tutti gli m vettori della base.

Quindi GMRES diventa troppo costoso ed inefficiente se m cresce troppo! Una soluzione è il restarting

Consiste nell'arrestare l'algoritmo quando viene raggiunto un certo m, calcolare x_m e ricominciare l'algoritmo con come approssimazione iniziale l'x_m precedente.

1: Calcola $r_0 = b - Ax_0$, $\beta := \|r_0\|_2$ e $v_1 := r_0/\beta$
2: Genera $\bar{H}_m$ e V_m con il metodo di Arnoldi
3: Calcola y_m t.c. minimizzi $\| \beta e_1 - \bar{H}_m y_m \|_2$ e $x_m = x_0 + V_m y_m$
4: **if** $\|x_m\|_2 <$ tolleranza **then** STOP
5: **end if** Poni $x_0 = x_m$ e vai alla 1

Vediamo come il restarting, pur facendo aumentare il numero delle iterazioni, riduce considerevolmente il tempo impiegato per trovare la soluzione.

GMRES

- Matrice fd3d 144000x144000 non simmetrica
- Senza restarting: converge in 169 iterazioni in circa 52 secondi

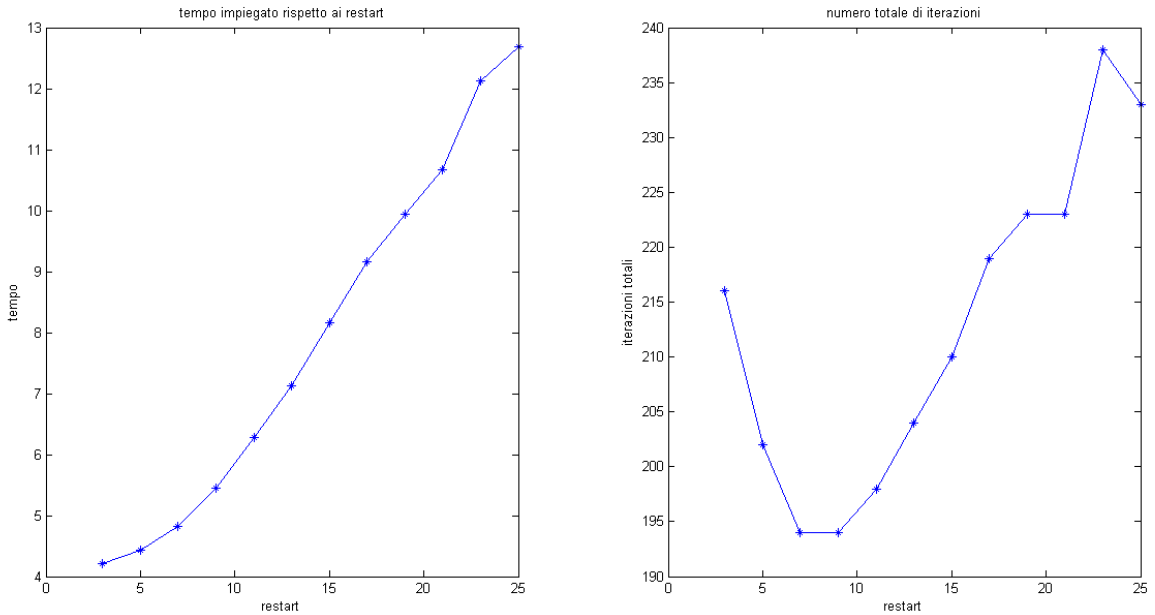


Figura : Risultati ottenuti facendo un restarting ogni $m = 3, 5, 7 \ldots, 25$. Ad esempio con $m = 3$ il tempo impiegato diminuisce, nonostante le iterazioni aumentino e, per $m = 7, o, 9$ il tempo aumenta nonostante le iterazioni diminuiscano.

GMRES

- Comunque non è sempre detto che il restarting migliori le prestazioni:
  infatti, mentre è garantito che GMRES converga in n passi, non è detto che il metodo non sia soggetto a delle stagnazioni usando questa tecnica.

- Studiamo quindi un altro metodo, con cui poi confronteremo le prestazioni:
  in particolare vorremmo che
  - non fosse necessario memorizzare tutti i vettori della base per calcolare la soluzione
  - non fossero necessari due cicli annidati.

vs BICGSTAB: Biortogonalizzazione di Lanczos

Un'altra classe di metodi proiettivi si basa, invece che sul metodo di ortogonalizzazione di Arnoldi, sull'algoritmo di biortogonalizzazione di Lanczos:
il principale vantaggio di questo metodo è che richiede di memorizare solo un numero ridotto di vettori.

Biortogonalizzazione di Lanczos

La biortogonalizzazione di Lanczos produce una coppia di basi ortogonali per i sottospazi

$$\mathcal{K}_m(A, v_1) = \text{span}\,\{v_1, Av_1, \ldots, A^{m-1}v_1\}$$
$$\mathcal{K}_m(A^T, w_1) = \text{span}\,\{w_1, A^Tw_1, \ldots, (A^T)^{m-1}w_1\}$$

vs BICGSTAB: Biortogonalizzazione di Lanczos

1: Scegli v_1 e w_1 t.c. $(v_1, w_1) = 1$
2: Poni $\beta_1 = \delta_1 = 0, w_0 = v_0 = 0$
3: **for** $j = 1, 2, \ldots, m$ **do**
4: $\quad \alpha_j = (Av_j, w_j)$
5: $\quad \hat{v}_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1}$
6: $\quad \hat{w}_{j+1} = A^T w_j - \alpha_j w_j - \delta_j w_{j-1}$
7: $\quad \delta_{j+1} = |(\hat{v}_{j+1}, \hat{w}_{j+1})|^{1/2}$
8: $\quad$ **if** $\delta_{j+1} = 0$ **then** STOP
9: $\quad$ **end if**
10: $\quad \beta_{j+1} = (\hat{v}_{j+1}, \hat{w}_{j+1})/\delta_{j+1}$
11: $\quad w_{j+1} = \hat{w}_{j+1}/\beta_{j+1}$
12: $\quad v_{j+1} = \hat{v}_{j+1}/\delta_{j+1}$
13: **end for**

COSTO e NOTE

Ci sono 2 prodotti matrice-vettore (righe 5 e 6) e vari prodotti (diciamo 10)

totale:

m (2 mat-vet + 10 prod)

C'è un solo ciclo.
Serve sia A sia A^T.

vs BICGSTAB:

Algoritmo BCG

Un algoritmo che si basa sulla biortogonalizzazione di Lanczos senza introdurre modifiche è il BCG (Biconjugate Gradient).

- Tramite questo algoritmo si risolvono sia $Ax = b$ sia $A^T x^* = b^*$
- E' efficiente se è davvero necessario risolvere entrambi i sistemi
- Diventa eccessivamente costoso se lo si usa solo per risolvere $Ax = b$.
  Infatti talvolta la matrice A è disponibile nella forma matrice-vettore (per esempio se rappresenta lo Jacobiano di una funzione quando si sta eseguendo il metodo di Newton) e in tal caso non è possibile ricavare anche A^T nella stessa forma.

Si cercano quindi delle modifiche all'algoritmo dette Transpose-Free Variants che non facciano uso della matrice trasposta.

vs BICGSTAB:

Transpose-Free Variants

Una prima soluzione è il CGS (Conjugate Gradient Squared)

- Scrive il residuo attraverso un polinomio $\varphi_j(A)$ elevato al quadrato.
- Questa scelta, però, in caso di convergeza irregolare, provoca un aumento negli errori di arrotondamento e anche possibili overflow.

Ciò porta a preferire un altro metodo:

BICGSTAB (Biconjugate Gradient Stabilized)

E' basato sullo scrivere il residuo come prodotto di due polinomi $\psi_j(A)\varphi_j(A)$.

Prima di vedere l'algoritmo BICGSTAB e di confrontarlo con il GMRES facciamo vedere dei casi in cui è più veloce e stabile di CGS e BICG.

Confronto: BICG, CGS, BICGSTAB

- Matrice fd3d 9000x9000 non simmetrica
- Confronto tempi di esecuzione:
  BICG: 0.186s CGS: 0.078s BICGSTAB: 0.0457s
- Confronto iterazioni:

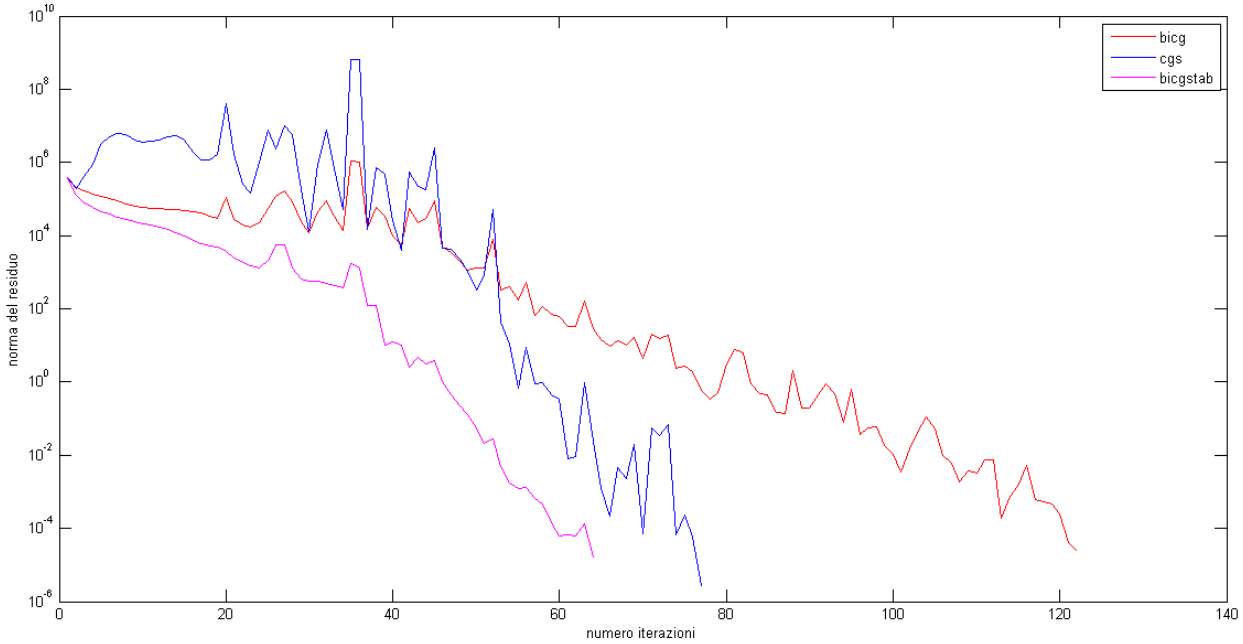
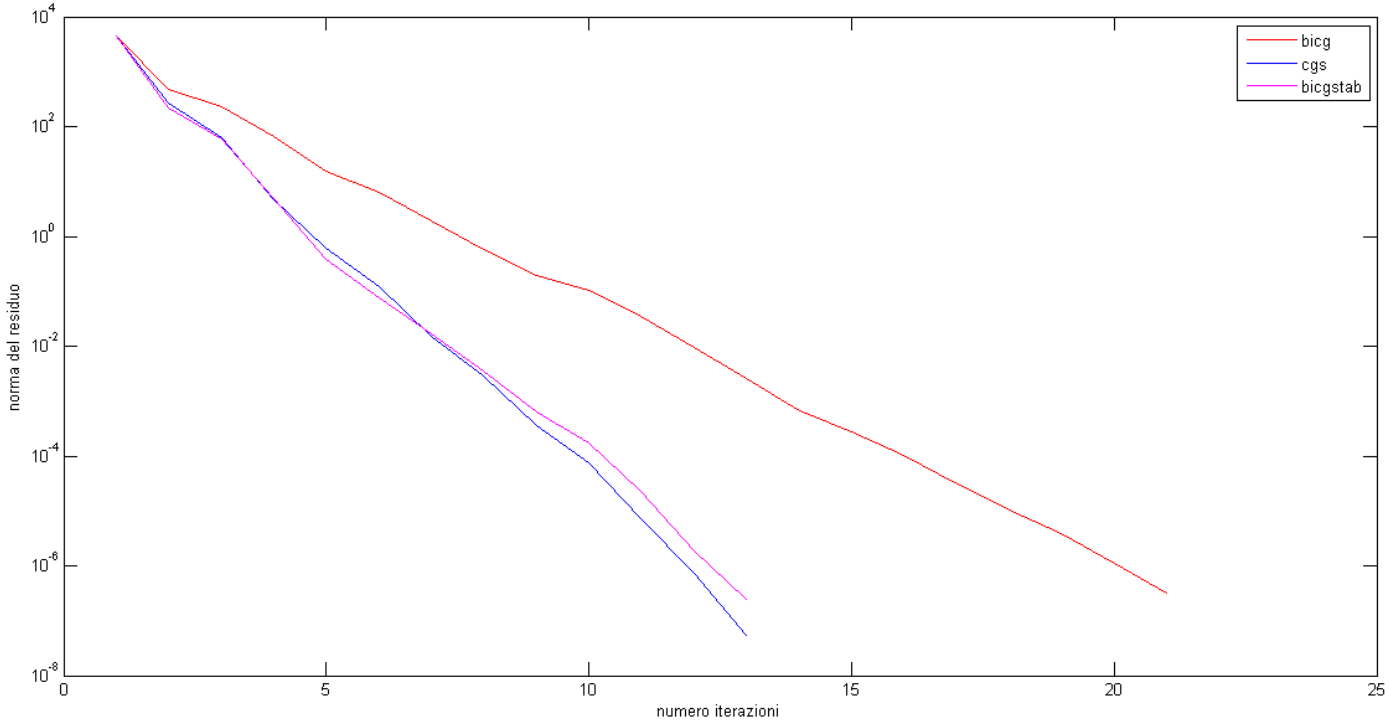


Figura : Si nota che l'andamento del residuo nel metodo CGS è molto instabile.

Confronto: BICG, CGS, BICGSTAB

- Matrice can_mod 350x350 simmetrica (7% elementi diversi da zero)
- Confronto tempi di esecuzione:
  BICG: 0.070s CGS: 0.047s BICGSTAB: 0.046s
- Confronto iterazioni:



BICGSTAB: Algoritmo

1: Calcola $\mathbf{r_0} := \mathbf{b} - A\mathbf{x_0}$ e scegli $\mathbf{r_0^*}$ arbitrariamente
2: Poni $\mathbf{p_0} := \mathbf{r_0}$
3: **for** $j = 0, 1, \ldots$ fino a convergenza **do**
4: $\quad \alpha_j := (\mathbf{r_j}, \mathbf{r_0^*})/(A\mathbf{p_j}, \mathbf{r_0^*})$
5: $\quad \mathbf{s_j} := \mathbf{r_j} - \alpha_j A\mathbf{p_j}$
6: $\quad \omega_j := (A\mathbf{s_j}, \mathbf{s_j})/(A\mathbf{s_j}, A\mathbf{s_j})$
7: $\quad \mathbf{x_{j+1}} := \mathbf{x_j} + \alpha_j \mathbf{p_j} + \omega_j \mathbf{s_j}$
8: $\quad \mathbf{r_{j+1}} := \mathbf{s_j} - \omega_j A\mathbf{s_j}$
9: $\quad \beta_j := \frac{(\mathbf{r_{j+1}}, \mathbf{r_0^*})}{(\mathbf{r_j}, \mathbf{r_0^*})} \cdot \frac{\alpha_j}{\omega_j}$
10: $\quad \mathbf{p_{j+1}} := \mathbf{r_{j+1}} + \beta_j(\mathbf{p_j} - \omega_j A\mathbf{p_j})$
11: **end for**

COSTO e NOTE

Ci sono 2 prodotti matrice-vettore ($A\mathbf{p_j}$ e $A\mathbf{s_j}$) e vari prodotti (diciamo 10)

1 solo ciclo

totale:

m (2 mat-vet + 10 prod)

Confronto: GMRES e BICGSTAB

Premessa

Non è possibile in generale dire quale dei due metodi sia il più efficiente!

Comunque, in base alle stime dei costi computazionali fatti, cioè

- GMRES: $m(\text{ mat-vet }) + m^2(\text{ prod })$
- BICGSTAB: $m\,(\,2 \text{ mat-vet } + 10 \text{ prod }\,)$

Possiamo formulare un'ipotesi:

Al diminuire del numero di elementi diversi da zero presenti nella matrice l'algoritmo BICGSTAB dovrebbe essere più rapido:
infatti nonostante nel BICGSTAB vi siano 2 prodotti matrice-vettore questi dovrebbero avere un costo limitato in relazione ai prodotti eseguiti m^2 volte e all'elevato costo di memorizzazione del GMRES

Confronto: GMRES e BICGSTAB

L'idea quindi è quella di testare i metodi su matrici con un numero di elementi diversi da zero via via crescenti ed esaminare

- il tempo impiegato
- il numero di iterazioni necessarie (per vedere se emergono altri elementi)

Usiamo matrici sparse costruite con `sprand(n,n,densità)`

- Problema: sono spesso mal condizionate
- Si deve quindi usare un precondizionatore: i risultati dipendono molto dall'accuratezza del precondizionatore usato

Confronto: GMRES e BICGSTAB

- matrice: 1000 x 1000 sparsa, random, precondizionatore: `ilu(A,struct('type','ilutp','droptol',0.0005))`
- Test eseguito su 4 matrici per ogni indice di sparsità 0.01 : 0.02 : 0.25

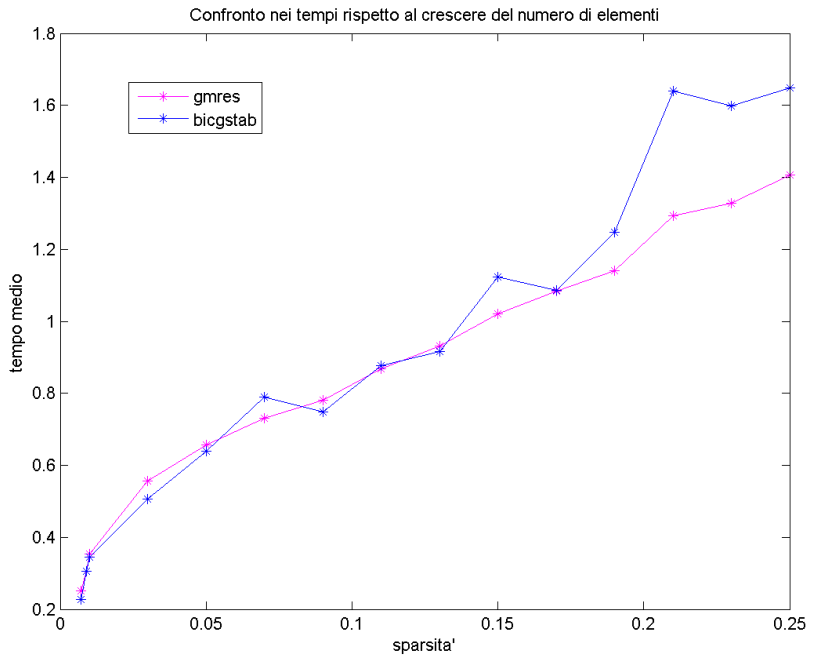


Figura : Al crescere della sparsità il BICGSTAB diventa più lento

Confronto: GMRES e BICGSTAB

matrice fd3d 9000 x 9000

- 0.07% elementi diversi da zero
- BICGSTAB: 0.0634s
  GMRES: 0.5289s

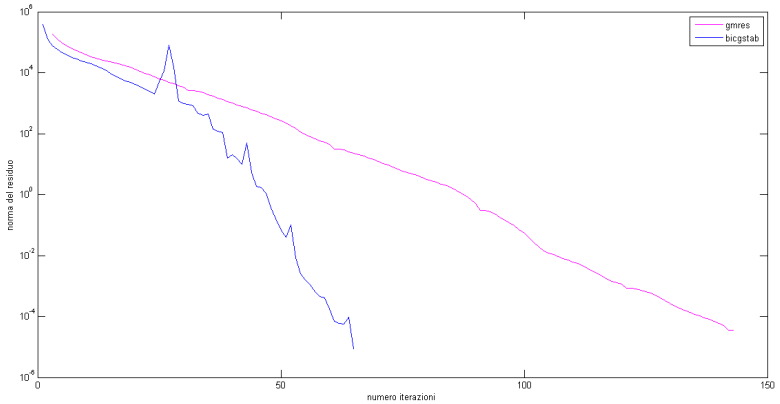


Figura : Bicgstab è più veloce e ci mette meno iterazioni

matrice: gallery('neumann',1600)

- 0.3% elementi diversi da zero
- BICGSTAB: 0.0197s
  GMRES: 0.1492s

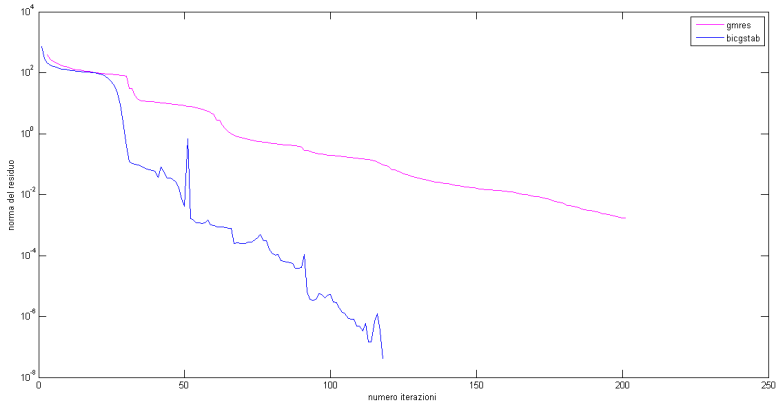


Figura : Bicgstab è più veloce e ci mette meno iterazioni

Confronto: GMRES e BICGSTAB

matrice can_mod.mat 350 x 350

- 7% elementi diversi da zero
- BICGSTAB: 0.0311s
  GMRES: 0.0614s

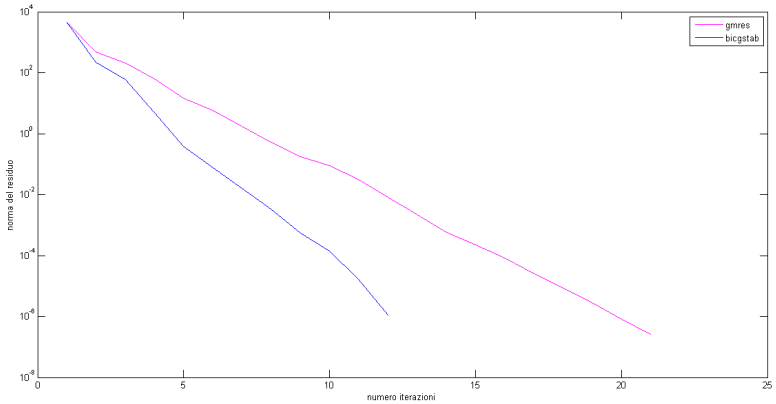


Figura : Bicgstab è più veloce e ci mette meno iterazioni

matrice: gallery('wilk',21)

- 13.6% elementi diversi da zero
- BICGSTAB: 0.0352s
  GMRES: 0.0690s

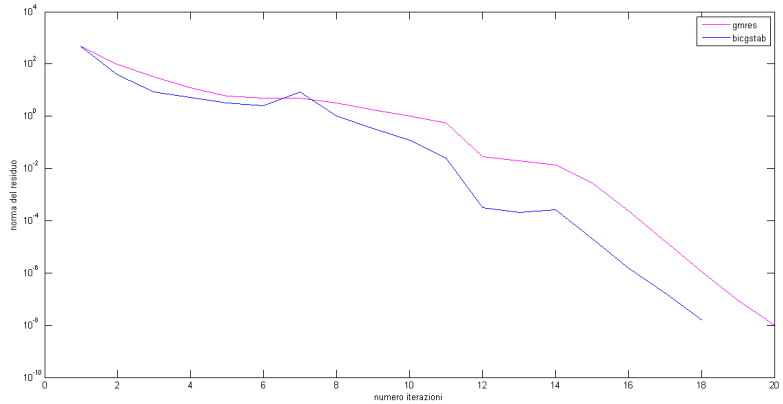


Figura : Bicgstab è ancora più veloce ma il numero di iterazioni è simile

Confronto: GMRES e BICGSTAB

matrice gallery('toeppen',12)

- 29.2% elementi diversi da zero
- BICGSTAB: 0.0122s
  GMRES: 0.0184s

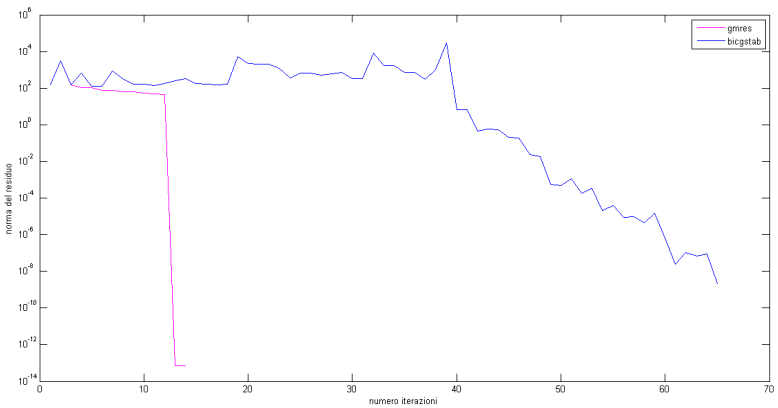


Figura : Bicgstab è più veloce ma ci mette più iterazioni

- Il tempo impiegato dall'algoritmo BICGSTAB, in questi test, è sempre inferiore.
- GMRES fa meno iterazioni (nonostante impieghi più tempo) di BICGSTAB al crescrere del numero di elementi diversi da zero.
- Ad influenzare il costo del GMRES, oltre ai prodotti eseguiti m^2 volte, potrebbe essere l'elevato numero di elementi che è necessario memorizzare.

Confronto: GMRES e BICGSTAB

Però a volte si ottengono anche risultati che non rispettano la previsione:

- matrice: gallery('kahan',45,20)
- 51.1% elementi diversi da zero
- BICGSTAB: 0.0363s GMRES: 0.0757s

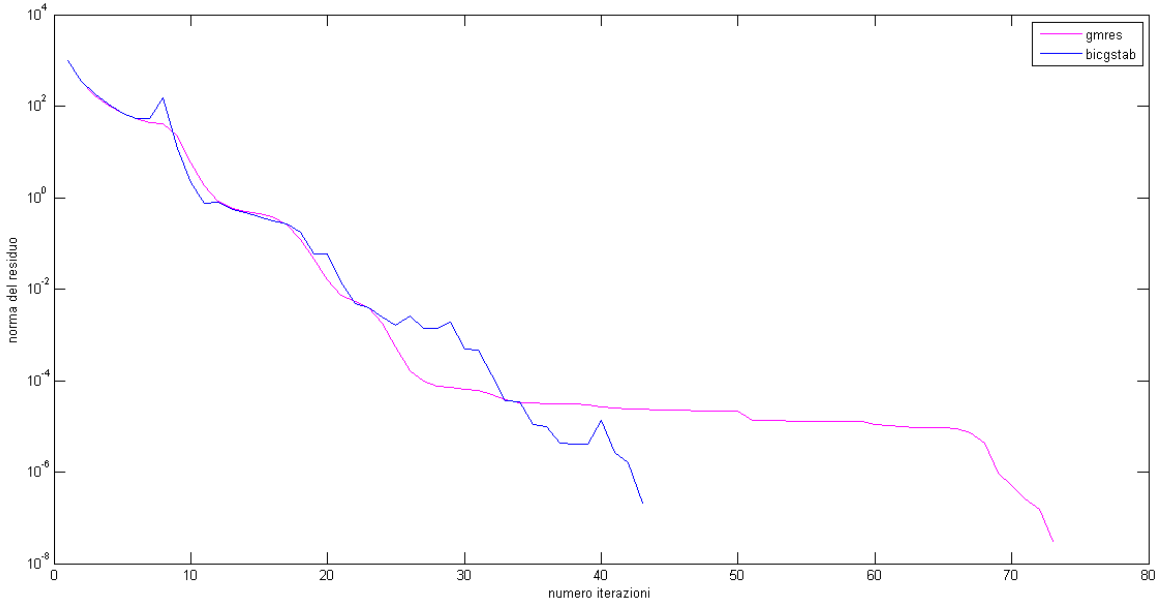


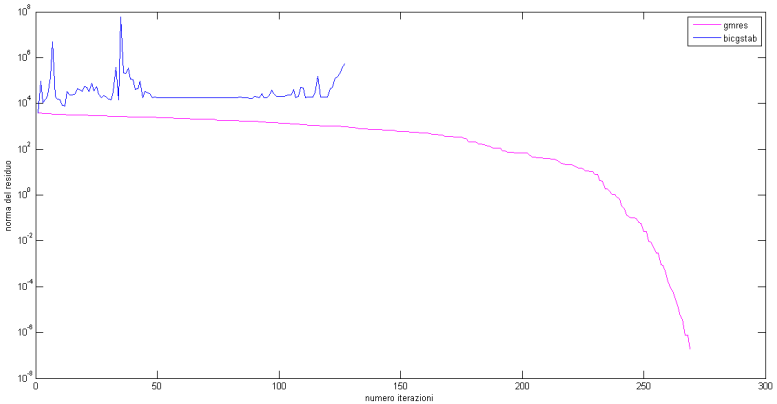
Figura : Bicgstab è più veloce e fa meno iterazioni nonostante la matrice sia per metà piena

Confronto: GMRES e BICGSTAB

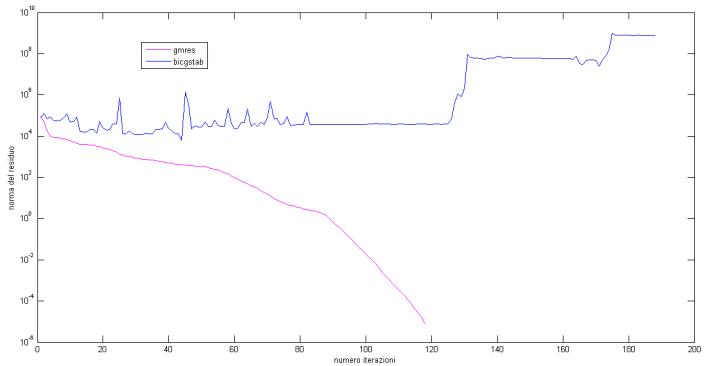
Qualche esempio in cui GMRES è più efficiente:

matrice gallery('randhess',350)

- 52.9% elementi diversi da zero
- GMRES: Converge in 0.8602s
  BICGSTAB: Non converge !



- matrice sprand(400,400,0.1)
- precondizionatore `ilu` con `droptol = 0.1`
- 9.53% elementi diversi da zero
- GMRES: Converge
  BICGSTAB: Non converge



Altri test che potrebbero essere realizzati

Per migliorare i risultati di confronto tra i due metodi si potrebbe

- Testare l'ipotesi fatta su più matrici
- Confrontare il BICGSTAB con il GMRES con diverse opzioni per il restarting
- Testare i metodi usando anche matrici che necessitano di un precondizionatore
- ...

Grazie per l'attenzione!

Riferimenti bibliografici


Yousef Saad *Iterative Method for Sparse Linear Systems*.


Alfio Quarteroni, *Modellistica Numerica per Problemi Differenziali*.