Università degli Studi di Cagliari

FACOLTÀ DI SCIENZE
Corso di Laurea Magistrale in Informatica

Relazione di Matematica Computazionale

Metodi iterativi per la risoluzione dei sistemi lineari

Relatore:
Prof.Giuseppe Rodriguez

Candidati:
Giuseppe Bellisano
Matricola 65107

Alberto Pes
Matricola 65106

Anno Accademico 2018–2019

Indice

1 Introduzione

Lo scopo di questo progetto è analizzare i metodi iterativi per la risoluzione dei sistemi lineari e fornire un piccolo ma efficace strumento software per l'esecuzione dei test di confronto. Nello specifico sono stati implementati ed esaminati il metodo di Jacobi, il metodo di Gauss-Seidel e i metodi del gradiente classico e coniugato, entrambi con e senza precondizionamento. I test sono stati condotti confrontando queste implementazioni con gli algoritmi del gradiente predefiniti di Matlab e considerando due tipologie di matrici, una **diagonalmente dominante** e una **sparsa simmetrica definita positiva**, quest'ultima impostando due diversi valori di condizionamento. Sono stati impostati, per entrambe le tipologie di matrici, diversi valori di dimensionamento e mediante l'ausilio di grafici e tabelle è stato effettuato un raffronto delle prestazioni dei diversi metodi.

Al fine di effettuare un'analisi puntuale e ripetibile abbiamo progettato ed implementato uno strumento software semi-automatizzato mediante Matlab che consente, previa impostazione dei parametri, di visualizzare i risultati dell'esecuzione dei diversi metodi e procedere quindi ad un agevole raffronto.

La presente relazione è strutturata nel modo seguente:

- *Sistemi lineari e metodi risolutivi.* In questa prima sezione verranno introdotti brevemente i sistemi lineari e i metodi per la loro risoluzione. Verranno evidenziate le differenze tra i metodi diretti e i metodi iterativi per procedere poi ad un'indicazione dei rispettivi casi applicativi. Verranno inoltre indicate definizioni e teoremi che saranno la base teorica per l'analisi e le considerazioni sui risultati dei test condotti.

- *Metodi iterativi.* In questa sezione si tratteranno in modo più specifico i metodi iterativi considerati, procedendo ad una descrizione del loro funzionamento e proponendo un'implementazione ad alto livello in pseudo-codice. Si illustreranno inoltre le strategie di rilassamento e di precondizionamento finalizzate all'accelerazione della convergenza.

- *Implementazione, test e analisi dei risultati.* In questa sezione finale verranno indicate le scelte implementative intraprese nello sviluppo dello strumento di analisi e verrà fornita una breve descrizione del suo funzionamento. Si procederà dunque al confronto dei dati scaturiti dai test effettuati ed infine all'elaborazione delle conclusioni sulla base dei risultati ottenuti.

- *Codice Matlab degli algoritmi.* Riportiamo in questa sezione l'implementazione in codice Matlab dei metodi iterativi considerati.

2 Sistemi lineari e metodi risolutivi

Un sistema lineare di n equazioni lineari in n incognite può essere rappresentato come

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \cdots \\ \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}xn = b_n \end{cases}$$

oppure in forma matriciale, più compatta

$$Ax = b$$

dove $A = (a_{ij})_{i,j=1}^{n}$ è la matrice dei coefficienti, $b = (b_1, \ldots, b_n)^T$ è il vettore dei termini noti e $(x_1, \ldots, x_n)^T$ è il vettore di soluzione.
Un sistema lineare cosi definito può essere risolto sia con i metodi diretti che con i metodi iterativi, questi ultimi oggetto della nostra trattazione.

I metodi diretti, che valutano la soluzione di un sistema lineare in un numero finito di passi, si basano sulla fattorizzazione della matrice A, modificandola e spesso alterandone la struttura. I metodi iterativi invece, che per $k = 0, 1, \ldots,$ raggiungono la soluzione come limite di una successione di vettori $x^{(k)}$, necessitano solamente di accedere agli elementi della matrice A del sistema senza apportarvi alcuna modifica.

Altra differenza riscontrabile tra i due metodi è data dagli errori presenti nei risultati. Nei metodi diretti questi nascono esclusivamente dall'aritmetica finita e dagli eventuali errori presenti nei dati; nei metodi iterativi si hanno, inoltre, gli errori di troncamento, derivanti dal fatto che il limite cercato deve essere necessariamente approssimato troncando la successione per un indice sufficientemente grande.

I metodi iterativi risultano particolarmente convenienti per matrici di grandi dimensioni, specialmente se strutturate o sparse (es. matrice del Web) in quanto, a differenza dei metodi diretti, non determinano un aumento del numero degli elementi non nulli ed evitano quindi di generare matrici che perdono rapidamente la proprietà di sparsità (fenomeno del **fill-in**).

Inoltre questi metodi possono essere utilizzati nei seguenti casi:

- raffinamento di una soluzione approssimata ottenuta con altri algoritmi o con informazioni a priori sul problema
- riduzione dei tempi di elaborazione eseguendo un minor numero di iterazioni nei casi in cui non vi è necessità di un'elevata accuratezza

2.1 Definizioni e teoremi

Un metodo iterativo *del primo ordine* è un metodo in cui il calcolo di un termine della successione coinvolge solo il termine precedente. Esso assume la forma

$$x^{(k+1)} = \psi(x^{(k)})$$

Un metodo iterativo lineare, stazionario del primo ordine assume la forma:

$$\mathbf{x}^{(k+1)} = B\mathbf{x}^{(k)} + \mathbf{f}$$

	Metodi diretti	**Metodi iterativi**
Metodo risolutivo	Numero finito di passi	Limite di una successione di vettori
Modifica matrice A	Sì	No
Migliori prestazioni	Matrici di dimensioni contenute	Matrici sparse di grandi dimensioni

Tabella 1: Confronto tra i metodi

- Lineare, in quanto tale è la relazione che lo esprime.
- Stazionario, perché la matrice di iterazione B ed il vettore f non variano al variare dell'indice di iterazione k.
- Del prim'ordine, perché il calcolo del vettore $x^{(k+1)}$ dipende solo dal termine precedente $x^{(k)}$.

Definizione 1. *Un metodo si dice* ***globalmente convergente*** *se per ogni vettore iniziale $x^{(0)} \in \mathbb{R}$ n si ha che*

$$\lim_{x \to \infty} \|x^{(k)} - x\| = 0$$

dove x è la soluzione del sistema e $\|\cdot\|$ denota una qualsiasi norma vettoriale.

Definizione 2. *Un metodo iterativo si dice* ***consistente*** *se*

$$x^{(k)} = x \implies x^{k+1} = x$$

Teorema 1. *La consistenza è una condizione necessaria (ma non sufficiente) per la convergenza*

Teorema 2. *Un metodo iterativo lineare, stazionario del primo ordine è consistente se e solo se*

$$f = (I - B)A^{-1}b$$

Teorema 3. *Condizione sufficiente per la convergenza del metodo iterativo $x^{(k+1)} = Bx^{(k)} + f$ è che esista una noma consistente $\|\cdot\|$ tale che $\|B\| < 1$*

Teorema 4. *Un metodo iterativo lineare è convergente se e solo se il raggio spettrale $\rho(B)$ della matrice di iterazione B è minore di 1.*

Si noti inoltre che la convergenza è globale, quindi non dipende dal vettore iniziale $x^{(0)}$.
Con queste premesse è possibile applicare i metodi iterativi effettuando un troncamento all'iterazione desiderata.

I metodi iterativi richiedono quindi di affrontare una serie di questioni:

- Costruzione della matrice B
- Stabilire ipotesi che garantiscano la convergenza della successione $x^{(k)}$ alla soluzione

- Scelta della soluzione iniziale $x^{(0)}$
- Analizzare la rapidità di convergenza, complessità computazionale e stima dell'errore
- Scelta di un criterio di arresto

3 Metodi iterativi

Una strategia utilizzata per la costruzione dei metodi iterativi lineari è quella dello *splitting additivo* che consiste nello scrivere la matrice nella forma

$$A = P - N$$

dove la matrice di precondizionamento P è non singolare. Sostituendo questa relazione nel sistema lineare:

$$A\mathbf{x} = \mathbf{b}$$

si ottiene il sistema equivalente:

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

che porta alla definizione del metodo iterativo:

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

Un metodo così costruito è consistente. Esso, inoltre, rientra nella classe di metodi del prim'ordine. Infatti, essendo $det(P) \neq 0$, possiamo scrivere:

$$\mathbf{x}^{(k+1)} = P^{-1}N\mathbf{x}^{(k)} + P^{-1}\mathbf{b} = B\mathbf{x}^{(k)} + \mathbf{f}$$

con $B = P^{-1}N$ e $\mathbf{f} = P^{-1}\mathbf{b}$.
Per i Teoremi 3 e 4 abbiamo:

- condizione sufficiente per la convergenza data da $\|P^{-1}N\| < 1$, dove $\|\cdot\|$ è una qualsiasi norma matriciale consistente
- condizione necessaria e sufficiente data da $\rho(P^{-1}N) < 1$.

Naturalmente, perché il metodo risulti operativo è necessario che la matrice P sia più semplice da invertire di A.
Consideriamo ora il particolare splitting additivo:

$$A = D - E - F$$

In cui:

$$D_{ij} = \begin{cases} a_{ij}, & i = j \\ 0 & i \neq j \end{cases} \qquad E_{ij} = \begin{cases} -a_{ij}, & i > j \\ 0 & i \leq j \end{cases} \qquad F_{ij} = \begin{cases} -a_{ij}, & i < j \\ 0 & i \geq j \end{cases}$$

3.1 Il metodo di Jacobi

Questo metodo, che gode dell'importante caratteristica di essere parallelizzabile, si basa sul seguente splitting additivo

$$P = D, \quad N = E + F$$

Possono verificarsi due casi

- la non singolarità di P si traduce in questo caso nella richiesta che $a_{ij} \neq 0$, per $i = 1, ..., n$;
- se la precedente condizione non è verificata, se A è non singolare, allora è possibile effettuare una permutazione delle righe in modo da rendere la diagonale priva di elementi nulli.

Nel metodo di Jacobi la (1)

$$Dx^{(k+1)} = b + Ex^{(k)} + Fx^{(k)}$$

o, in coordinate:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{n} a_{ij} x_j^{(k)} \right], \quad per \quad j \neq i \quad e \quad i = 1, ..., n.$$

Questa formula consente di ricavare le componenti di $x^{(k+1)}$ a partire da quelle di $x^{(k)}$ in qualsiasi ordine e indipendentemente l'una dall'altra ed evidenzia dunque la possibilità di esecuzione in parallelo.

La **matrice di iterazione** è

$$B_J = P^{-1}N = D^{-1}(E + F) = I - D^{-1}A$$

La convergenza sarà possibile se e solo se $\rho(B_j) < 1$.

Metodo Jacobi codice Matlab

Per il codice Matlab si veda il listato 5

3.2 Il metodo di Gauss-Seidel

In questo metodo le condizioni per l'Invertibilità di P sono identiche a quelle viste nel metodo di Jacobi. Il metodo di Gauss-Seidel corrisponde alla scelta

$$P = D - E, \quad N = F$$

Sostituendo tali matrici alla $Px^{(k+1)} = Nx^{(k)} + b$ si ottiene la relazione

$$(D - E)x^{(k+1)} = b + Fx^{(k)}$$

Da questa, risolvendo un sistema triangolare inferiore, è possibile ottenere $x^{(k+1)}$. Infatti, passando in coordinate possiamo esprimere esplicitamente le componenti del nuovo iterato, dalla (5.3) si ricava:

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

Matrice A	Jacobi	Gauss-Seidel
Predom. diagonale stretta	Converge	Converge
Irrid. diagonalmente dominante	Converge	Converge
Simmetrica definita positiva	Non converge	Converge

Tabella 2: confronto tra i metodi Jacobi e Gauss-Seidel

Questa formula è operativa solo se applicata per valori crescenti dell'indice i. Poiché la componente i-esima di $x^{(k+1)}$ dipende dalle componenti con indice compreso tra 1 e $i-1$, il metodo di Gauss-Seidel non è parallelizzabile. Esso però, in molte occasioni, è più rapidamente convergente del metodo di Jacobi, cioè richiede meno iterati per raggiungere una data accuratezza.

La **matrice di iterazione** è

$$B_{GS} = (D - E)^{-1}F$$

Metodo Gauss-Seidel codice Matlab

Per il codice Matlab si veda il listato 6

3.3 Criteri di arresto

Un metodo iterativo fornisce in generale una successione infinita di iterate $x^{(k)}$. Per questo motivo riveste particolare importanza il criterio di arresto che determina sia la qualità dell'approssimazione della soluzione, sia il tempo di elaborazione necessario alla sua determinazione. Prenderemo in considerazione tre criteri di arresto, di cui i primi due sono generali, quindi utilizzabili in qualsiasi metodo iterativo, mentre l'ultimo è rivolto alla sola risoluzione di sistemi lineari. In generale le condizioni utilizzate per l'arresto dei metodi iterativi non sono però costituite da un solo criterio ma spesso risulta vantaggioso utilizzare una loro combinazione ottenuta con l'ausilio di operatori logici *and* e/o *or*.

1. Questo metodo rileva la convergenza verificando che la successione delle iterate soddisfi il criterio di Cauchy. Questo avviene fissando una tolleranza $\tau > 0$, una norma vettoriale ed esaminando lo scarto tra due iterazioni successive. La condizione di stop è data dalla forma

   $$\|x^{(k} - x^{(k-1)}\| \leq \tau$$

   o, meglio, dalla condizione seguente che misura lo scarto relativo:

   $$\frac{\|x^{(k} - x^{(k-1)}\|}{\|x^{(k)}\|} \leq \tau$$

   Per motivi numerici, quest'ultimo criterio d'arresto viene implementato come segue:

   $$\|x^{(k} - x^{(k-1)}\| \leq \tau \, \|x^{(k)}\|$$

   L'errore al passo k quando la matrice A è simmetrica definitiva positiva verifica la maggiorazione

$$\|e^{(k)}\|_2 \leq \frac{1}{1-\rho(B)}\|x^{(k+1)} - x^{(k)}\|_2$$

La scelta di τ dipende dalla precisione desiderata che deve essere significativamente superiore alla precisione della macchina. Questo test di arresto è affidabile solo quando il raggio spettrale della matrice di iterazione B è sufficientemente inferiore a 1.

2. Bisogna sempre considerare la possibilità che il metodo non converga, introducendo un numero massimo di iterazioni N e quindi introducendo un criterio di arresto $k > N$.

3. Per l'esplicita risoluzione dei sistemi lineari è possibile ottenere, al passo k, una maggiorazione per l'errore $e^{(k)} = x^{(k)} - x$ in termini del vettore residuo $r^{(k)} = b - Ax^{(k)}$. Dalla relazione $Ae^{(k)} = -r^{(k)}$ si ricava

$$\|e^{(k)}\| = \|A^{-1}r^{(k)}\| \leq \|A^{-1}\|\|r^{(k)}\|$$

Ricordando che $\|b\| \leq \|A\|\|x\|$, si ottiene la maggiorazione per l'errore relativo al passo k

$$\frac{\|e^{(k)}\|}{\|x\|} \leq \|A\|\|A^{-1}\|\frac{\|r^{(k)}\|}{\|b\|} = k(A)\frac{\|r^{(k)}\|}{\|b\|}$$

Il criterio di stop è quindi

$$\frac{\|r^{(k)}\|}{\|b\|} \leq \tau$$

La stima dell'errore fornita dal residuo normalizzato è tanto meno attendibile quanto più la matrice A è mal condizionata.

3.4 Precondizionamento

Compito del precondizionatore è quello di accelerare la convergenza del metodo utilizzato. Si può scrivere

$$P^{-1}A\mathbf{x} = P^{-1}\mathbf{b}$$

con P matrice di *precondizionamento*.

La scelta di un precondizionatore ottimale è un compito difficile, sia perché ne esiste un numero infinito, sia perché è necessario tenere conto delle caratteristiche della matrice di riferimento (con matrici di dimensioni elevate si utilizza un P che preservi la struttura e la sparsità di A). Un buon precondizionatore deve far si che gli autovalori della matrice precondizionata $P^{-1}A$ si accumulino in un intorno di 1 (cluster a 1). In generale è opportuno che P approssimi la matrice A in modo che $P^{-1}A \simeq I$ mantenendo P facile da invertire.

3.5 Fattorizzazione LU incompleta

La fattorizzazione LU se applicata ad una matrice sparsa A comporta il verificarsi del fenomeno del *fill-in* con cui i fattori triangolari tendono ad avere un numero di elementi non nulli superiore a quelli della matrice A di partenza. Questo fatto risulta sconveniente per due motivi:

– aumenta lo spazio necessario per la memorizzazione degli elementi della matrice

– aumenta il numero di calcoli necessari per effettuare la fattorizzazione LU

Per i motivi di qui sopra, si preferisce lavorare con una fattorizzazione leggermente modificata, definita appunto **LU incompleta**, con cui si calcolano solo quegli elementi dei fattori, il cui valore assoluto è superiore ad una certa soglia σ. Questa fattorizzazione costituisce una importante classe di precondizionatori. In questo modo si impone una struttura di sparsità ai fattori, che determina però la perdita dell'uguaglianza $A = LU$, è però possibile tarando opportunamente il valore σ fare in modo che il prodotto LU approssimi la matrice A, venendo così a colmare quei lati negati evidenziati precedentemente nella fattorizzazione LU, ovvero:

– la complessità computazionale si abbassa poiché si calcolano solo gli elementi maggiori di σ

– diminuiscono il numero di operazioni utili a risolvere i sistemi triangolari per via del fatto che vengono mantenuti gli zeri nei fattori L e U

Il precondizionatore sarà dunque $P = LU$

Se A è una matrice simmetrica definita positiva è possibile calcolare un altro tipo di fattorizzazione denominata *Cholesky incompleta*. Con questa fattorizzazione si determina la matrice triangolare superiore R tale che $A \simeq R^T R$ e si ottiene il precondizionatore $P = R^T R$

3.6 Metodi di rilassamento

I metodi di rilassamento hanno l'importante obiettivo di rendere minimo il raggio spettrale della matrice, determinando quindi un'accellerazione della convergenza. Questo è possibile mediante l'introduzione di un parametro ω, detto *parametro di rilassamento*. L'iterazione assume la forma

$$x^{(k+1)} = (1 - \omega)x^{(k)} + \omega\, \tilde{x}^{(k+1)}$$

dove $\tilde{x}^{(k+1)}$ è il vettore generato da Jacobi (metodo JOR *Jacobi over relaxation*) oppure dal metodo Gauss-Seidel (SOR *successive over relaxation*).

3.6.1 Convergenza di JOR e SOR

Prendendo in esame le matrici simmetriche definite positive riportiamo due teoremi per cui valgono:

- il metodo iterativo JOR converge se

$$0 < \omega < \frac{2}{\rho(D^{-1}A)}$$

- il metodo iterativo SOR converge se e solo se

$$0 < \omega < 2$$

3.6.2 Metodi di Richardson

È possibile introdurre un parametro di rilassamento anche nella formulazione generale di un metodo lineare precondizionato:

$$x^{(k+1)} = x^{(k)} + P^{-}1\mathbf{r}^{(k)}$$

La classe di metodi così ottenuta è quella dei metodi di Richardson stazionari:

$$x^{(k+1)} = x^{(k)} + \alpha P^{-}1\mathbf{r}^{(k)}$$

Se si consente al parametro di rilassamento di variare al cambio dell'iterazione, si ottengono i metodi di Richardson non stazionari:

$$x^{(k+1)} = x^{(k)} + \alpha_k P^{-}1\mathbf{r}^{(k)}$$

I metodi JOR e SOR sono metodi di Richardson stazionari con $\alpha = \omega$ e rispettivamente, $P = D$ e $P = D - E$.

Generalmente si riesce a stimare un parametro di rilassamento ottimale quando si conoscono informazioni sullo spettro di $P^{-1}A$. Riportiamo a riguardo il seguente teorema.

Teorema 5. *Se $P^{-1}A$ ha autovalori reali positivi $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n > 0$. allora il metodo di Richardson stazionario converge se e solo se $0 < \alpha < 2/\lambda_1$. Inoltre il raggio spettrale della matrice di iterazione è minimo per*

$$\alpha = \frac{2}{\lambda_1 + \lambda_2}$$

3.7 Metodo del Gradiente

Consideriamo una matrice A che sia simmetrica definita positiva.
Valgono dunque le seguenti proprietà:

$$A = A^T$$

$$x^T Ax > 0 \quad \forall x \in \mathbb{R}^n$$

Consideriamo la seguente forma quadratica (graficamente un paraboloide ellittico):

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

Tale funzione è minima nel punto in cui si annulla il gradiente, tenendo conto della simmetria di A scriviamo dunque:

$$\nabla\phi(y) = \frac{1}{2}(A + A^T)y - b = \frac{1}{2} \cdot 2Ay - b = Ay - b = 0 \tag{3}$$

Dunque, il problema della minimizzazione della (2) è equivalente alla soluzione del sistema lineare $Ax = b$. Intendiamo calcolare questa soluzione minimizzando $\phi(y)$ con un metodo iterativo non stazionario del tipo:

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

a partire da un vettore $x^{(0)}$, lungo direzioni di decrescita $d^{(k)}$, con passi di lunghezza α_k (si vedano le figure seguenti).
Occupiamoci dapprima della lunghezza del passo. Qualunque sia la direzione di discesa $d^{(k)}$, possiamo determinare il minimo di $\phi(x^{(k+1)})$ rispetto alla variazione di $\alpha = \alpha_k$. Infatti considerando la forma quadratica abbiamo:

$$\phi(x^{(k+1)}) = \frac{1}{2}(x^{(k)} + \alpha d^{(k)})^T A(x^{(k)} + \alpha d^{(k)}) - (x^{(k)} + \alpha d^{(k)})^T b =$$
$$= \frac{1}{2}[{x^{(k)}}^T A x^{(k)} + \alpha {x^{(k)}}^T A d^{(k)} + \alpha {d^{(k)}}^T A x^{(k)} + \alpha^2 {d^{(k)}}^T A d^{(k)}] - {x^{(k)}}^T b - \alpha {d^{(k)}}^T b =$$

con

$$\phi(x^{(k)}) = \frac{1}{2}{x^{(k)}}^T A x^{(k)} - {x^{(k)}}^T b$$

possiamo scrivere

$$= \phi(x^{(k)}) + \frac{1}{2}\alpha^2 {d^{(k)}}^T A d^{(k)} - \alpha {d^{(k)}}^T b + \frac{1}{2} \cdot 2\alpha {d^{(k)}}^T A x^{(k)} =$$
$$= \frac{1}{2}\alpha^2 {d^{(k)}}^T A d^{(k)} - \alpha {d^{(k)}}^T (b - A x^{(k)}) + \phi(x^{(k)}) =$$

con $r^{(k)} = b - Ax^{(k)}$ scriviamo:

$$= \phi(x^{(k)}) + \frac{1}{2}{d^{(k)}}^T A d^{(k)} \cdot \alpha^2 - {d^{(k)}}^T r^{(k)} \cdot \alpha$$

segue quindi:

$$\frac{d}{d\alpha}\phi(x^{(k+1)}) = {d^{(k)}}^T A d^{(k)} \cdot \alpha - {d^{(k)}}^T r^{(k)} \tag{4}$$

Il minimo si ottiene dunque nel modo seguente:

$$\alpha_k = \frac{{d^{(k)}}^T r^{(k)}}{{d^{(k)}}^T A d^{(k)}} \tag{5}$$

Nel metodo del gradiente si sceglie come direzione $d^{(k)}$ quella di massima discesa (steepest descent), ossia quella opposta alla direzione del gradiente della funzione ϕ nel punto $x^{(k)}$. Questa direzione coincide col residuo al passo k, infatti, come già visto in 3 abbiamo:

$$\nabla\phi(x^{(k)}) = Ax^{(k)} - b = -r^{(k)}$$

L'iterazione assume dunque la forma:

$$x^{(k+1)} = x^{(k)} + \alpha_k r^{(k)}$$

dove il valore α_k è ottenuto sostituendo $d^{(k)} = r^{(k)}$ nella 5.
Osserviamo infine che, considerando la precedente relazione, la complessità computazionale può essere ridotta aggiornando il residuo nel modo seguente (invece di calcolarlo ad ogni iterazione):

$$r^{(k+1)} = b - Ax^{(k+1)} = b - A(x^{(k)} - \alpha_k r^{(k)}) = b - Ax^{(k)} - \alpha_k Ar^{(k)} = r^{(k)} - \alpha_k Ar^{(k)}$$

Riportiamo di seguito i passi principali dell'algoritmo in pseudo-codice:

1. Scegli $\mathbf{x}^{(0)}$
2. $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$
3. $k = 0$
4. repeat
   (a) $\mathbf{s} = A\mathbf{r}^{(k)}$
   (b) $\alpha_k = \frac{{\mathbf{r}^{(k)}}^T \mathbf{r}^{(k)}}{{\mathbf{r}^{(k)}}^T \mathbf{s}}$
   (c) $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{r}^{(k)}$
   (d) $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha_k \mathbf{s}$
   (e) $k = k + 1$
5. until <condizione di convergenza>

Metodo del Gradiente codice Matlab

Per il codice Matlab si veda il listato 1

3.8 Metodo del Gradiente Precondizionato

Il metodo del gradiente è spesso caratterizzato da una convergenza piuttosto lenta da risultare poco utilizzabile senza l'accelerazione fornita da un opportuno precondizionatore. Questo presuppone avere a disposizione una matrice P che approssimi A e che al tempo stesso sia invertibile con basso costo computazionale. Il metodo assume in questo caso la forma generale di un metodo di Richardson non stazionario:

$$x^{(k+1)} = x^{(k)} + \alpha_k P^{-1} r^{(k)} = x^{(k)} + \alpha_k z^{(k)}$$

Ad ogni passo è necessario risolvere il sistema $Pz^{(k)} = r^{(k)}$. Bisogna inoltre determinare $\alpha^{(k)}$ tenendo conto del cambiamento della direzione di discesa.
Si ottiene così il **metodo del gradiente precondizionato** che riportiamo di seguito:

1. Scegli $\mathbf{x}^{(0)}$
2. $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$

3. risolvi $Pz^{(0)} = r^{(0)}$
4. $k = 0$
5. repeat
   (a) $\mathbf{s} = A\mathbf{z}^{(k)}$
   (b) $\alpha_k = \frac{\mathbf{z}^{(k)^T}\mathbf{r}^{(k)}}{\mathbf{z}^{(k)^T}\mathbf{s}}$
   (c) $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k\mathbf{z}^{(k)}$
   (d) $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha_k\mathbf{s}$
   (e) risolvi $P\mathbf{z}^{(k+1)} = \mathbf{r}^{(k+1)}$
   (f) $k = k + 1$
6. until <condizione di convergenza>

Metodo del Gradiente precondizionato codice Matlab

Per il codice Matlab si veda il listato 2

3.9 Metodo del Gradiente Coniugato

Il metodo del gradiente coniugato è una variante del metodo del gradiente ottenuta operando una scelta più accurata delle direzioni di discesa. Si può dimostrare che, grazie a questa differente scelta, in aritmetica infinita il metodo converge alla soluzione esatta in n iterazioni ed è quindi da considerare un metodo diretto. Viene però di fatto utilizzato come un metodo iterativo perché se opportunamente precondizionato, fornisce un'approssimazione sufficientemente accurata della soluzione in un numero di iterazioni molto inferiore alla dimensione del sistema, fatto di particolare importanza quando si opera su sistemi lineari di dimensione estremamente elevata.

Definizione 3. *Un vettore $x^{(k)}$ è ottimale rispetto ad una direzione p se*

$$\phi(x^{(k)}) \leq \phi(x^{(k)} + \lambda p), \quad \forall \lambda \in \mathbb{R}$$

Se $x^{(k)}$ è ottimale rispetto ad ogni direzione p di un sottospazio V, diremo che è ottimale rispetto a V.

Teorema 6. *Il vettore $x^{(k)}$ è ottimale rispetto a p se e solo se la direzione p è ortogonale al residuo $r^{(k)}$, cioè:*

$$p^T r^{(k)} = 0$$

Dimostrazione: Procedendo come nella 4 partiamo dalla forma quadratica:

$$\phi(x) = \frac{1}{2}x^T Ax - x^T b$$

Abbiamo dunque:

$$\phi(x^{(k)} + \lambda p) = \frac{1}{2}(x^{(k)} + \lambda p)^T A(x^{(k)} + \lambda p) - (x^{(k)} + \lambda p)^T b =$$
$$= \frac{1}{2}[x^{(k)^T} Ax^{(k)} + \lambda x^{(k)^T} Ap + \lambda p^T Ax^{(k)} + \lambda^2 p^T Ap] - x^{(k)^T} b - \lambda p^T b =$$

con

$$\phi(x^{(k)}) = \frac{1}{2}{x^{(k)}}^T Ax^{(k)} - {x^{(k)}}^T b$$

possiamo scrivere:

$$= \phi(x^{(k)}) + \frac{1}{2} \cdot 2 \cdot \lambda p^T Ax^{(k)} + \frac{1}{2}\lambda^2 p^T Ap - \lambda p^T b =$$
$$= \phi(x^{(k)}) + \frac{1}{2}\lambda^2 p^T Ap - \lambda p^T (b - Ax^{(k)})$$

con $r^{(k)} = b - Ax^{(k)}$ scriviamo:

$$= \phi(x^{(k)}) + \frac{1}{2} p^T Ap \cdot \lambda^2 - p^T r^{(k)} \cdot \lambda$$

Segue quindi:

$$\frac{d}{d\lambda}\phi(x^{(k)} + \lambda p) = p^T Ap \cdot \lambda - p^T r^{(k)} = 0$$

Se $x^{(k)}$ è ottimale rispetto a p, $\phi(x^{(k)})$ deve avere un minimo per $\lambda = 0$, il che implica la tesi. Viceversa, se $p^T r^{(k)} = 0$ si ha un minimo per $\lambda = 0$.

Importante, vogliamo fare in modo che l'ottimalità del vettore k-esimo rispetto ad una certa direzione venga ereditata da tutti i successivi elementi della successione. Ipotizzando di avere tre punti (soluzioni) $x^{(k-1)}$, $x^{(k)}$ e $x^{(k+1)}$ in cui la direzione p conduce da $x^{(k-1)}$ a $x^{(k)}$ e la direzione q conduce da $x^{(k)}$ a $x^{(k+1)}$ possiamo scrivere:

1. Se $x^{(k)}$ è ottimale rispetto a p allora vale: $p^T r^{(k)} = 0$
2. Se $x^{(k+1)}$ è ottimale rispetto a q allora vale: $q^T r^{(k+1)} = 0$ ma perdo l'ottimalità rispetto a p

Supponiamo che $x^{(k)}$ sia ottimale rispetto a p e, quindi, che $p^T r^{(k)} = 0$, e poniamo:

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

Come possiamo garantire che $x^{(k+1)}$ conservi l'ottimalità rispetto a p?
Ci chiediamo a quali condizioni valga la seguente:

$$p^T r^{(k+1)} = 0$$

Scriviamo dunque:

$$p^T r^{(k+1)} = p^T (b - Ax^{(k+1)}) = p^T (b - A(x^{(k)} + q)) =$$

con $r^{(k)} = b - Ax^{(k)}$ scriviamo:

$$p^T (r^{(k)} - Aq) = p^T r^{(k)} - p^T Aq = 0$$

con $p^T r^{(k)} = 0$.

Cioè le direzioni p e q devono essere A-ortogonali o A-coniugate.
Come troviamo dunque le direzioni A coniugate?
Partiamo con $p^{(0)} = r^{(0)}$ e supponiamo di aver già trovato k direzioni A coniugate $p^{(0)}, ..., p^{(k)}$ e consideriamo dunque direzioni del tipo:

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

In cui β_k è uno scalare che sottraggo per il vettore precedente.
La richiesta che $p^{(k+1)} e p^{(k)}$ siano due direzioni A-coniugate, cioè che:

$$p^{(k)^T} A p^{(k+1)} = 0$$

Si traduce nella seguente:

$$\begin{aligned} p^{(k)^T} A p^{(k+1)} &= p^{(k)^T} A(r^{(k+1)} - \beta_k p^{(k)}) = \\ &= p^{(k)^T} A r^{(k+1)} - \beta_k p^{(k)^T} A p^{(k)} = 0 \end{aligned}$$

da cui otteniamo:

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

È possibile dimostrare che con questa scelta del parametro β_k si ha:

$$p^{(i)^T} A p^{(k+1)} = 0, \quad i = 0, \ldots, k$$

cioè che la direzione $p^{(k+1)}$ è A-coniugata con tutte le direzioni generate precedentemente. Il fatto che il vettore:

$$x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)}$$

con α_k scelto in base alla 5, sia ottimale rispetto alle direzioni $p^{(i)}, i = 0, \ldots, k$, significa che lungo tali direzioni non è possibile far diminuire ulteriormente il valore della funzione obiettivo $\phi(y)$ e giustifica il fatto che l'algoritmo converga alla soluzione esatta in un numero finito di iterazioni.

Riportiamo quindi l'**algoritmo del gradiente coniugato**:

1. scegli $\mathbf{x}^{(0)}$
2. $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$
3. $\mathbf{p}^{(0)} = \mathbf{r}^{(0)}$
4. $k = 0$
5. repeat
   (a) $\mathbf{s} = A\mathbf{p}^{(k)}$
   (b) $\delta = (\mathbf{p}^{(k)})^T \mathbf{s}$

(c) $\alpha_k = (\mathbf{p}^{(k)})^T \mathbf{r}^{(k)} / \delta$
(d) $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{p}^{(k)}$
(e) $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha_k \mathbf{s}$
(f) $\beta_k = \mathbf{s}^T \mathbf{r}^{(k+1)} / \delta$
(g) $\mathbf{p}^{(k+1)} = \mathbf{r}^{(k+1)} - \beta_k \mathbf{p}^{(k)}$
(h) $k = k + 1$

6. until <condizione di convergenza>

Metodo del Gradiente coniugato codice Matlab

Per il codice Matlab si veda il listato 3

3.10 Metodo del Gradiente Coniugato Precondizionato

Nell'ipotesi che la matrice A sia simmetrica definita positiva è possibile dimostrare che la relazione:

$$\|\mathbf{y}\|_A = \sqrt{\mathbf{y}^T A \mathbf{y}}$$

Definisce una norma per ogni $\mathbf{y} \in \mathbb{R}^n$. Riguardo alla velocità di convergenza, vale allora il risultato seguente:

$$\|\mathbf{e}^{(k)}\|_A \leq 2 \left(\frac{\sqrt{k_2(A)} - 1}{\sqrt{k_2(A)} + 1} \right)^k \|\mathbf{e}^{(0)}\|_A$$

Il metodo del gradiente coniugato precondizionato che riportiamo di seguito, spesso abbreviato con la sigla PCG, si ottiene precondizionando il residuo k-esimo con una matrice P che sia semplice da invertire e tale che $P^{(-1)}A \simeq I$.
Riportiamo di seguito l'algoritmo del metodo del gradiente coniugato precondizionato:

1. scegli $\mathbf{x}^{(0)}$
2. $\mathbf{r}(0) = \mathbf{b} - A\mathbf{x}^{(0)}$
3. risolvi $P\mathbf{z}^{(0)} = \mathbf{r}^{(0)}$
4. $\mathbf{p}^{(0)} = \mathbf{z}^{(0)}$
5. $k = 0$
6. repeat
   (a) $\mathbf{s} = A\mathbf{p}^{(k)}$
   (b) $\delta = (\mathbf{p}^{(k)})^T \mathbf{s}$
   (c) $\alpha_k = (\mathbf{p}^{(k)})^T \mathbf{r}^{(k)} / \delta$
   (d) $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{p}^{(k)}$
   (e) $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha_k \mathbf{s}$
   (f) risolvi $P\mathbf{z}^{(k+1)} = \mathbf{r}^{(k+1)}$

(g) $\beta_k = \mathbf{s}^T \mathbf{z}^{(k+1)} / \delta$

(h) $\mathbf{p}^{(k+1)} = \mathbf{z}^{(k+1)} - \beta_k \mathbf{p}^{(k)}$

(i) $k = k + 1$

7. until <condizione di convergenza>

Metodo del Gradiente coniugato precondizionato codice Matlab

Per il codice Matlab si veda il listato 4

Risulta importante affermare che, se la matrice A non è simmetrica definita positiva, né il metodo del gradiente né il metodo del gradiente coniugato sono applicabili. Tuttavia è possibile applicare entrambi i metodi al sistema normale seguente:

$$A^T A\mathbf{x} = A^T \mathbf{b} \tag{6}$$

il quale, se la matrice A è quadrata non singolare, ha la stessa soluzione del sistema di partenza $A\mathbf{x} = \mathbf{b}$. In questo caso, tuttavia, per ridurre la complessità computazionale connessa al calcolo della matrice $A^T A$, migliorando anche la stabilità, gli algoritmi visti in questa sezione (gradiente e sue varianti) vengono opportunamente modificati in modo da evitare il calcolo esplicito di $A^T A$.

Resta il fatto che il sistema 6 ha un numero di condizionamento pari al quadrato di quello corrispondente al sistema di partenza e questo fatto, ovviamente, provoca un maggior accumulo di errori e un rallentamento della velocità di convergenza del metodo iterativo adottato. Naturalmente, se la matrice di partenza non fosse quadrata non singolare, ma ad esempio, rettangolare con più righe che colonne, l'applicazione di un metodo iterativo al sistema normale 6 fornirebbe un'approssimazione della soluzione $A\mathbf{x} = \mathbf{b}$ nel senso dei minimi quadrati.

4 Implementazione, test e analisi dei risultati

Lo strumento di analisi è stato progettato allo scopo di essere di semplice e rapido utilizzo ed inoltre particolarmente flessibile e personalizzabile. All'avvio del programma l'utente potrà:

1. selezionare la tipologia di matrice che desidera utilizzare per i test
2. definire il numero di test (step) di diversa dimensione che si vuole considerare (es. 10, 100, 1000)
3. modificare i parametri di generazione della matrice predefiniti (es. condizionamento e sparsità)
4. configurare i parametri per l'esecuzione del test (es. tolleranza e numero massimo di iterazioni)
5. visualizzare in formato tabellare i risultati relativi a tempi, iterazioni ed errori relativi

6. visualizzare infine le seguenti tipologie di grafici:
   (a) grafici in scala logaritmica per i residui
   (b) grafici a barre per i tempi e le iterazioni

Le funzioni implementate sono dunque le seguenti:

- **IterMethodsTool:** funzione semi-automatizzata per:
  - acquisire l'input utente per la configurazione del problema
  - effettuare le chiamate alle singole funzioni dei metodi iterativi
  - memorizzare e visualizzare i risultati
- **MatrixCreator:** funzione per la creazione delle matrici dalla differente struttura
- **Varie:** funzioni specifiche per i singoli metodi iterativi

Si allega il collegamento al repository Github con il codice opportunamente commentato.

4.1 Tipologie di test

I test condotti mirano al raffronto tra le nostre implementazioni e gli algoritmi di Matlab che rappresentano un punto di riferimento per la comunità scientifica. Questo verrà effettuato considerando il numero di iterazioni, il tempo impiegato e l'errore relativo al variare della tipologia, della dimensione e del condizionamento delle matrici. La soluzione del sistema utilizzato è stata impostata come un vettore di componenti unitarie.

Vengono effettuati tre test considerando due tipologie di matrici:

1. diagonalmente dominante
   (a) Fattore di dominanza diagonale: 2
   (b) Formato di memorizzazione: sparsa
2. sparsa simmetrica definita positiva
   (a) Sparsità della matrice: 5%
   (b) Numero di condizionamento: 100
3. sparsa simmetrica definita positiva meglio condizionata
   (a) Sparsità della matrice: 5%
   (b) Numero di condizionamento: 20

I parametri impostati per l'esecuzione degli algoritmi sono i seguenti:

- Tolleranza: 1e-12
- Numero massimo di iterazioni: 200

I parametri impostati per il precondizionamento mediante Cholesky incompleta sono i seguenti:

- Tipologia: ict (Cholesky incompleta con sogliatura)
- Valore di soglia: 1e-3
- Formato: upper (cioè $R^T R$)

Elenchiamo di seguito il significato delle sigle utilizzate nel seguito per denotare gli algoritmi considerati:

- Metodi Matlab:
  - **MCG**: Metodo del gradiente coniugato Matlab
  - **MPCG**: Metodo del gradiente coniugato precondizionato Matlab
- Nostre implementazioni:
  - **SG**: Metodo del gradiente classico o di massima discesa
  - **SPG**: Metodo del gradiente classico precondizionato
  - **SCG**: Metodo del gradiente coniugato
  - **SPCG**: Metodo del gradiente coniugato precondizionato
  - **Jacobi**: Metodo di Jacobi
  - **G. Seidel**: Metodo di Gauss-Seidel

4.2 Test 1 - Matrice diagonalmente dominante

In questo primo test il dimensionamento della matrice è stato impostato con n pari a 100, 1000 e 2500. Come possiamo evincere dai grafici in Figura 1 e dalle tabelle 3 e 5 delle iterazioni e degli errori relativi, tutti i metodi analizzati convergono. Possiamo però notare rapidamente come Jacobi sia in generale il metodo meno performante. Approfondendo l'analisi e concentrandoci sulla variazione di dimensione della matrice notiamo come per la dimensione pari a 100 i metodi di Matlab non siano quelli migliori. Per quanto riguarda la rapidità di esecuzione Gauss-Seidel e Jacobi offrono le migliori prestazioni anche se osservando la tabella 4 e la tabella 5 notiamo come a fronte di un leggera perdita di rapidità SPG e SPCG sono più precisi.

Aumentando la dimensione, nello specifico per n pari a 1000, notiamo nel secondo grafico di Figura 1 come le linee che identificano i metodi precondizionati iniziano ad avvicinarsi separandosi dai metodi non precondizionati e dal metodo di Gauss-Seidel. Dal punto di vista dei tempi gli algoritmi SPG e SPCG risultano essere i migliori, SCG invece è il migliore per precisione. Le prestazioni di Matlab migliorano rispetto al caso precedente.

L'ultimo step, dimensionamento pari a 2500, segna il sorpasso da parte di Matlab e l'affermazione dell'algoritmo MPCG. In questo caso è ancor più evidente la velocità di convergenza dei metodi precondizionati, con delle linee quasi verticali nell'ultimo grafico di Figura 1. Ancor più ampia è inoltre la forbice che li separa dai metodi non precondizionati. Gauss-Seidel si allontana in modo più netto dai metodi del gradiente, Jacobi rimane stabile nel numero delle iterazioni, ma cresce il suo tempo di esecuzione.

In conclusione quindi, per questo primo test, i metodi del gradiente precondizionato risultano i migliori per matrici di grandi dimensioni e l'implementazione Matlab risulta quella più performante. Per dimensioni minori di 1000, si affermano invece le nostre implementazioni, questo può essere motivato, probabilmente, dai sofisticati controlli adottati da Matlab nei suoi algoritmi che per matrici piccole determinano un calo delle prestazioni.

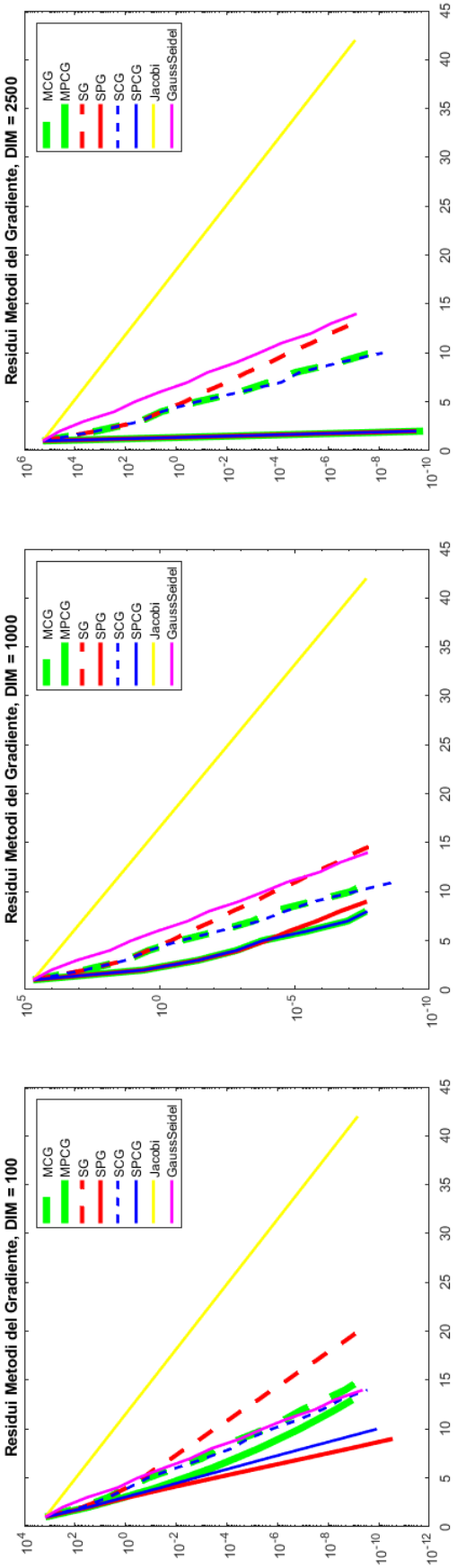

Figura 1: Test 1. Grafici dei residui in funzione delle iterazioni. Da sinistra a destra la dimensione della matrice è: 100, 1000, 2500

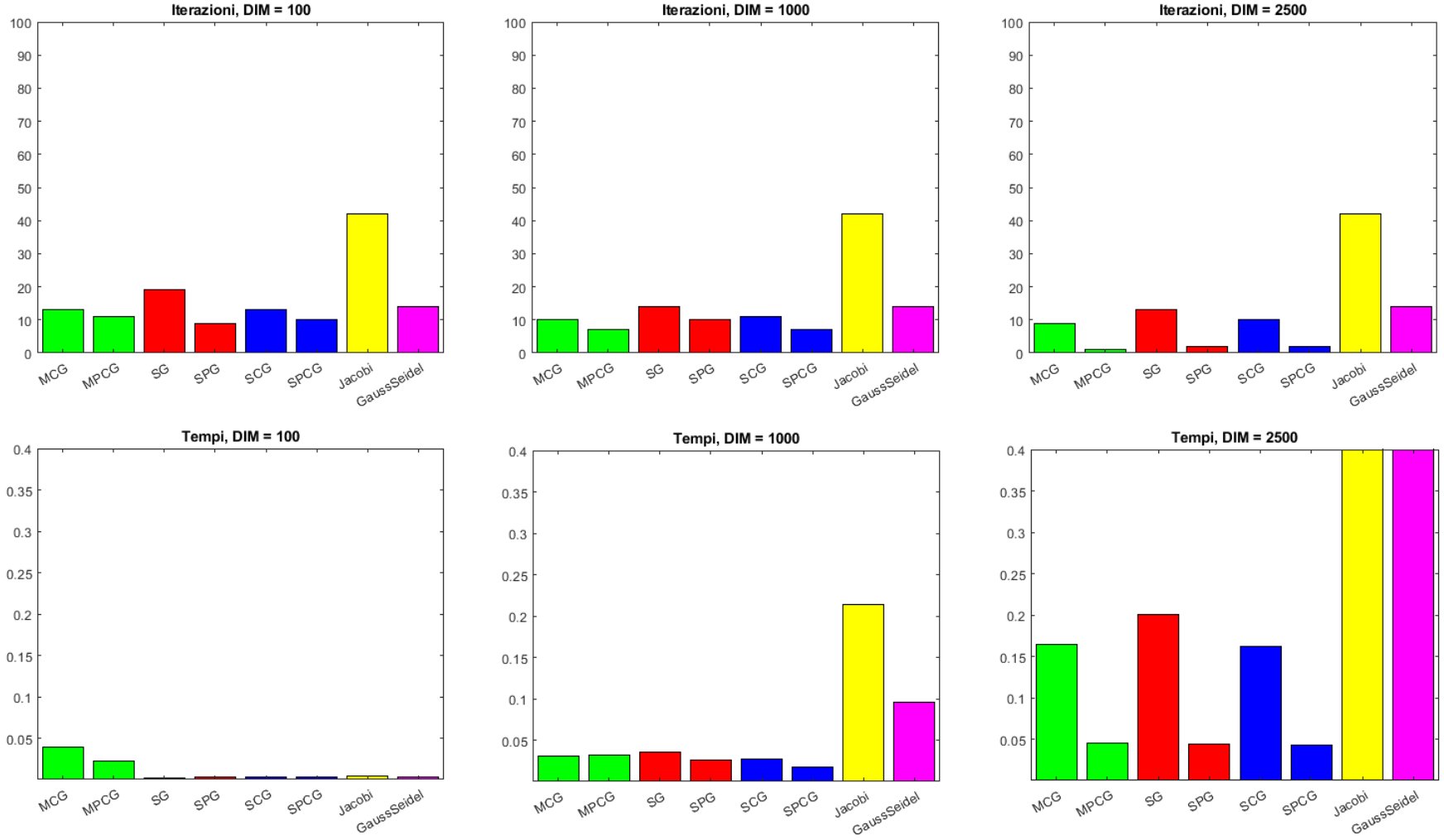

Figura 2: Test 1. Grafici delle iterazioni e dei tempi. Da sinistra a destra la dimensione della matrice è: 100, 1000, 2500

n	MCG (t)	MPCG (t)	SG (t)	SPG (t)	SCG (t)	SPCG (t)	Jacobi (t)	G. Seidel (t)
100	14	12	20	9	14	10	42	14
1000	10	7	15	9	11	8	42	14
2500	9	1	13	2	10	2	42	14

Tabella 3: Test 1. Iterazioni

n	MCG (t)	MPCG (t)	SG (t)	SPG (t)	SCG (t)	SPCG (t)	Jacobi (t)	G. Seidel (t)
100	0.033569	0.010479	0.0080638	0.0078992	0.0077705	0.0078705	0.006949	0.0051472
1000	0.028508	0.024294	0.036201	0.022908	0.026445	0.020157	0.21114	0.095977
2500	0.17646	0.047579	0.21243	0.050552	0.16558	0.054361	1.5075	0.62184

Tabella 4: Test 1. Tempi

n	MCG (t)	MPCG (t)	SG (t)	SPG (t)	SCG (t)	SPCG (t)	Jacobi (t)	G. Seidel (t)
100	4.784e-13	1.0822e-12	1.6397e-13	9.7725e-16	3.0378e-14	4.0286e-15	2.2742e-13	4.0022e-14
1000	4.2749e-13	5.7663e-13	3.9119e-14	6.4682e-14	8.5527e-15	1.7608e-14	2.2741e-13	4.2286e-14
2500	1.1966e-13	1.3933e-15	1.0622e-13	3.2134e-15	2.9927e-15	3.2143e-15	2.2741e-13	4.1747e-14

Tabella 5: Test 1. Errore relativo

4.3 Test 2 - Matrice sparsa simmetrica definita positiva

In questo secondo test sono stati considerati i medesimi step di dimensionamento del precedente, 100, 1000 e 2500. Osservando i grafici in Figura 3 appare evidente l'assenza di convergenza del metodo di Jacobi, per questa tipologia di matrice infatti, come indicato nel paragrafo 3 a pagina 6, questo metodo non converge. Riscontriamo anche la lentissima convergenza di SG e Gauss-Seidel che determina il breakdown di entrambi, Figura 3. Possiamo inoltre notare come, a differenza del test precedente, anche per dimensioni piccole della matrice i metodi del gradiente precondizionato siano più performanti rispetto ai restanti, risultando graficamente ben separati dagli altri algoritmi. Dal punto di vista dei tempi per questo primo step non si evidenziano particolari differenze tra i metodi, fatta eccezione per gli algoritmi di Matlab che risultano più lenti. Dal punto di vista della precisione invece SPG e SPCG risultano quelli migliori.

Aumentando la dimensione ed arrivando allo step intermedio di 1000, notiamo un aumento generale del numero di iterazioni richieste e del tempo di esecuzione. Inizia ad essere più evidente la perdita di prestazioni del metodo SG, mentre gli algoritmi di Matlab risultano più competitivi rispetto allo step precedente. SPG e SPCG risultano gli algoritmi migliori per tempi e precisione come notiamo dalle tabelle 7 e 8.

Aumentando ulteriormente la dimensione, step di ordine 2500, aumentano il numero di iterazioni e i tempi di esecuzione. Si fa un po' più stretta la forbice tra i metodi del gradiente precondizionati e quelli non precondizionati, lasciandoci ipotizzare che per n molto grande questi metodi tendano via via ad assomigliarsi nelle prestazioni. Inoltre le linee che denotano MCG, SCG e Gauss-Seidel si avvicinano a quella relativa a SG. Anche per questo dimensionamento risulta più performante SPCG rispetto a MPCG, sia dal punto di vista dei tempi, seppur per pochi millesimi di secondo, sia dal punto di vista della precisione.

In conclusione dunque per questa tipologia di matrice e per gli step di dimensionamento analizzati l'algoritmo SPCG risulta il migliore. Possiamo ipotizzare con buona certezza, considerando il trand di miglioramento, che per dimensioni più elevate di quelle analizzate l'algoritmo MPCG risulti in definitiva quello più performante.

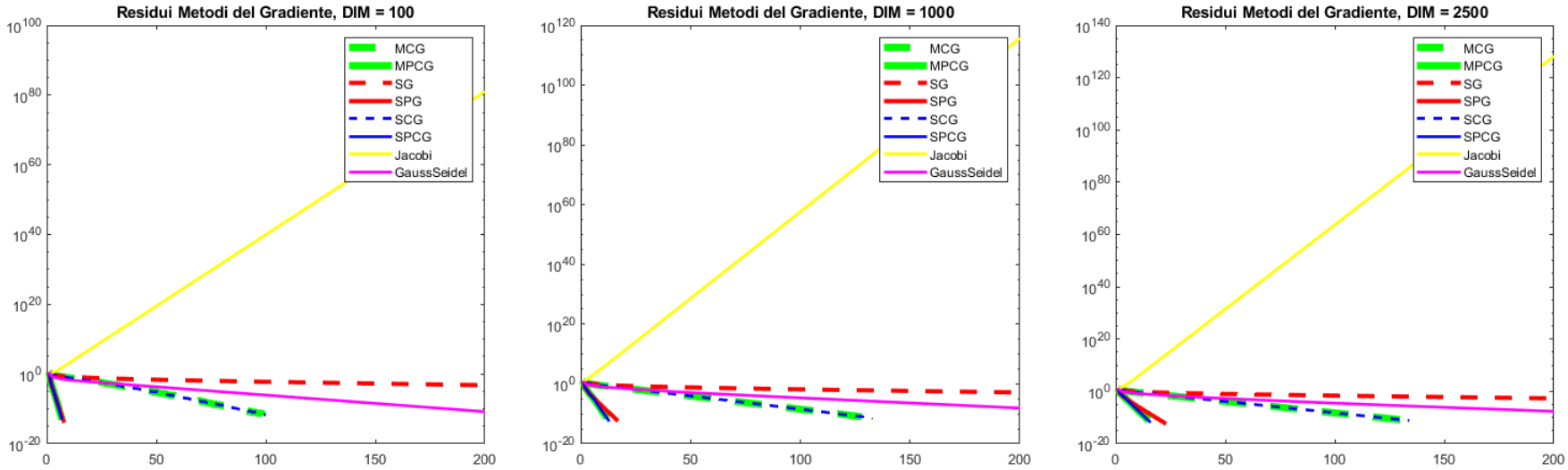

Figura 3: Test 2. Grafici dei residui in funzione delle iterazioni. Da sinistra a destra la dimensione della matrice è: 100, 1000, 2500

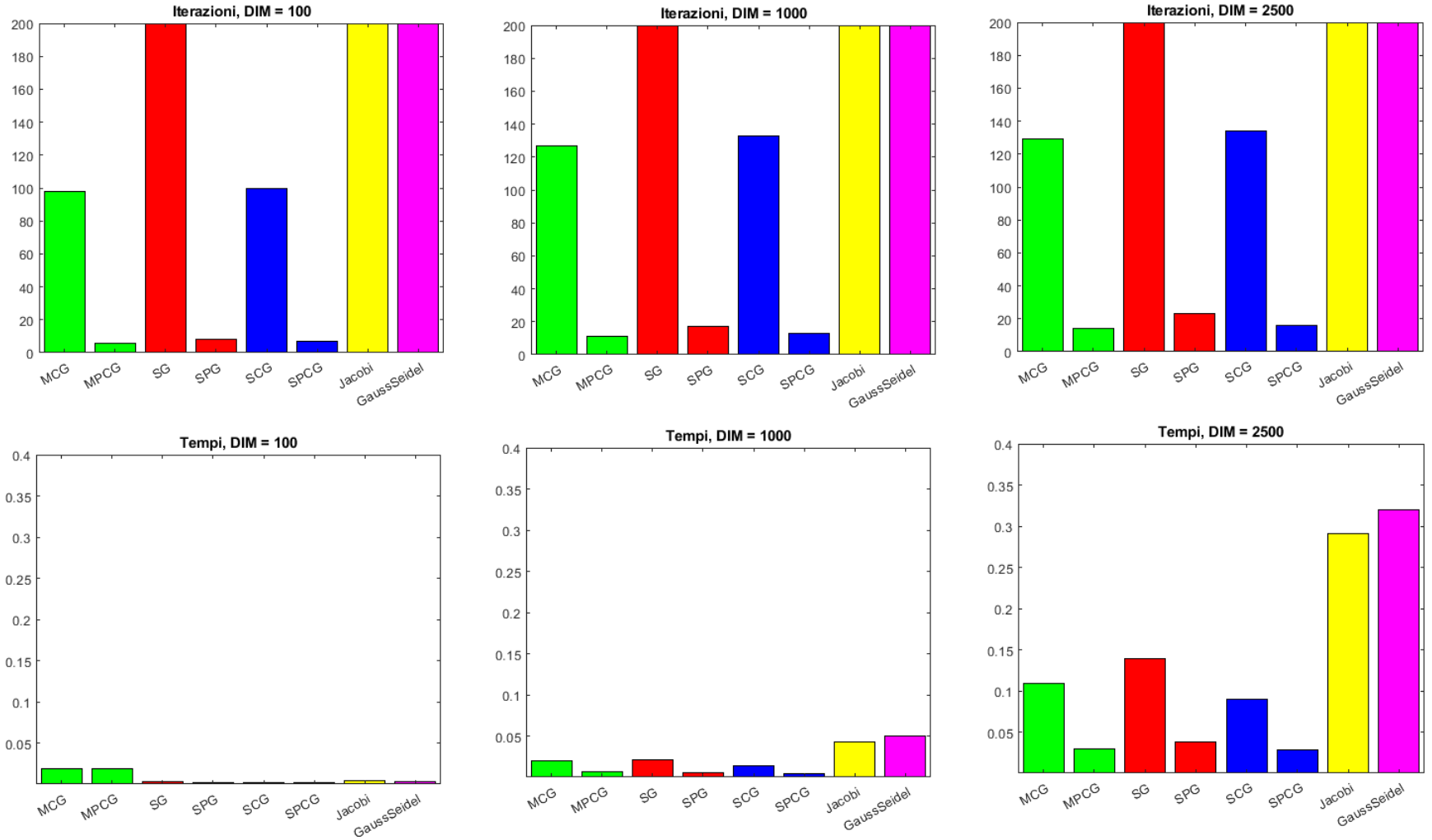

Figura 4: Test 2. Grafici delle iterazioni e dei tempi. Da sinistra a destra la dimensione della matrice è: 100, 1000, 2500

n	MCG (t)	MPCG (t)	SG (t)	SPG (t)	SCG (t)	SPCG (t)	Jacobi (t)	G. Seidel (t)
100	98	6	200	8	100	7	200	200
1000	127	11	200	17	133	13	200	200
2500	129	14	200	23	134	16	200	200

Tabella 6: Test 2 - Iterazioni

n	MCG (t)	MPCG (t)	SG (t)	SPG (t)	SCG (t)	SPCG (t)	Jacobi (t)	G. Seidel (t)
100	0.018885	0.018248	0.002781	0.0020018	0.0024748	0.0019381	0.0038562	0.0033344
1000	0.0203	0.0062865	0.020704	0.0059102	0.014084	0.0048979	0.042455	0.049854
2500	0.10927	0.029769	0.13904	0.038567	0.089936	0.028686	0.29115	0.32016

Tabella 7: Test 2 - Tempi

n	MCG (t)	MPCG (t)	SG (t)	SPG (t)	SCG (t)	SPCG (t)	Jacobi (t)	G. Seidel (t)
100	7.5704e-12	1.181e-13	0.003891	8.3474e-16	3.7684e-12	1.1434e-15	3.4342e+80	5.8585e-11
1000	7.9646e-12	3.2932e-12	0.002908	1.1652e-13	2.3431e-12	2.4346e-14	6.9061e+114	1.7838e-08
2500	7.221e-12	4.5475e-12	0.0027443	1.0312e-13	2.5775e-12	1.1232e-13	2.5169e+127	2.0259e-08

Tabella 8: Test 2 - Errore relativo

4.4 Test 3 - Matrice sparsa simmetrica definita positiva meglio condizionata

In quest'ultimo test abbiamo considerato la stessa tipologia di matrice del test precedente con dimensione 2500, modificando però il numero di condizionamento. Il grafico a sinistra in Figura 9 mostra l'esecuzione con condizionamento pari a 100, mentre nel grafico a destra della medesima figura abbiamo riportato il caso con condizionamento pari a 20. Possiamo notare che con un condizionamento minore, come è facile presumere, la convergenza avviene più rapidamente. Ad esclusione di Jacobi ed di SG tutti gli altri metodi migliorano notevolmente. Il miglioramento più significativo è da attribuire al metodo di Gauss-Seidel che passa da una situazione di breakdown ad una situazione di convergenza avvicinandosi alle prestazioni dei metodi del gradiente coniugato non precondizionati. In entrambi i casi comunque il metodo migliori risultano SPCG e MPCG con quest'ultimo leggermente meno preciso ma più rapido nel caso peggio condizionato, si vedano le tabelle 11, 12 e 13.

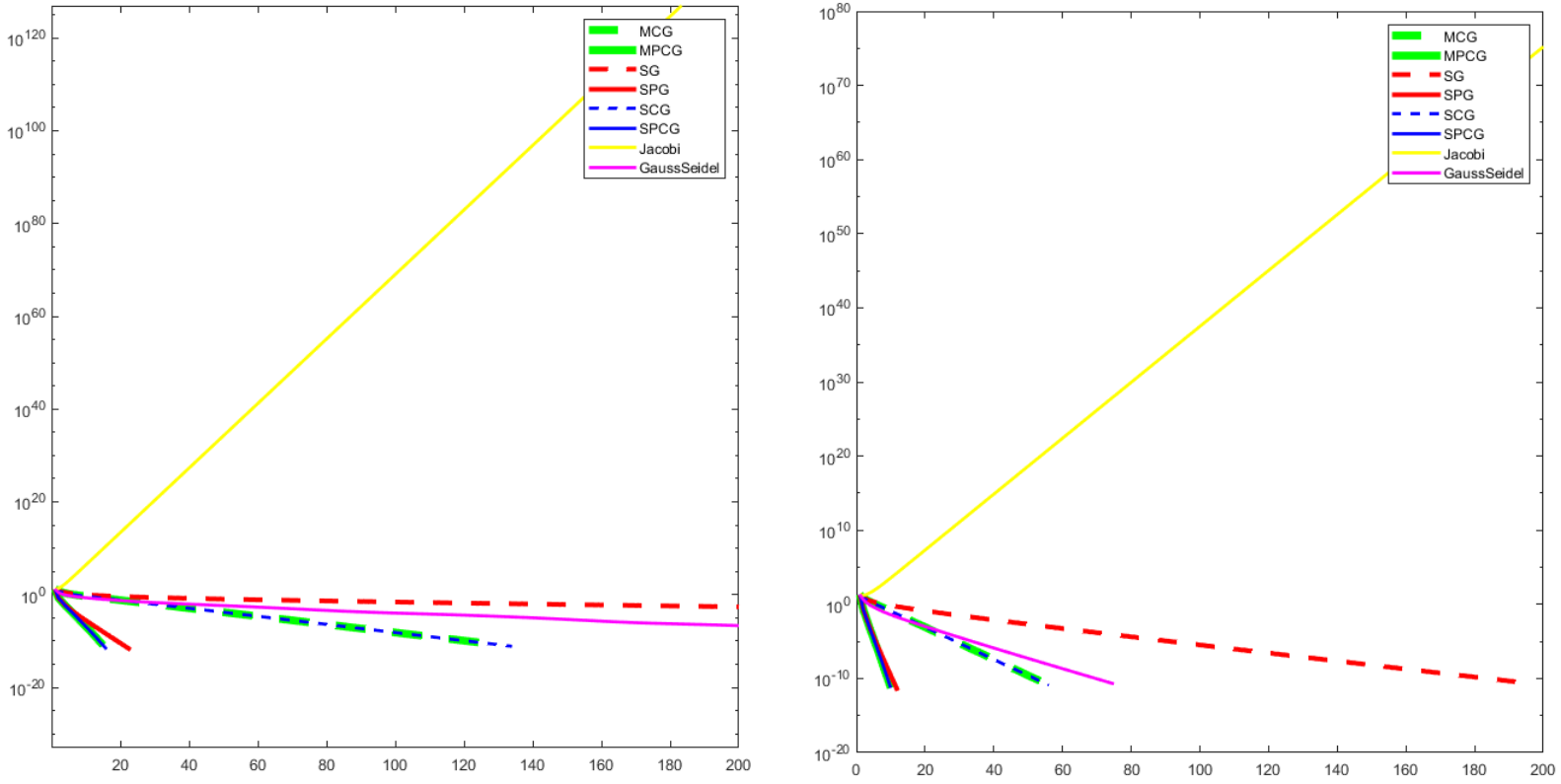

Tabella 9: Test 3. Grafici dei residui in funzione delle iterazioni. A sinistra il condizionamento è pari a 100, a destra è pari a 20.

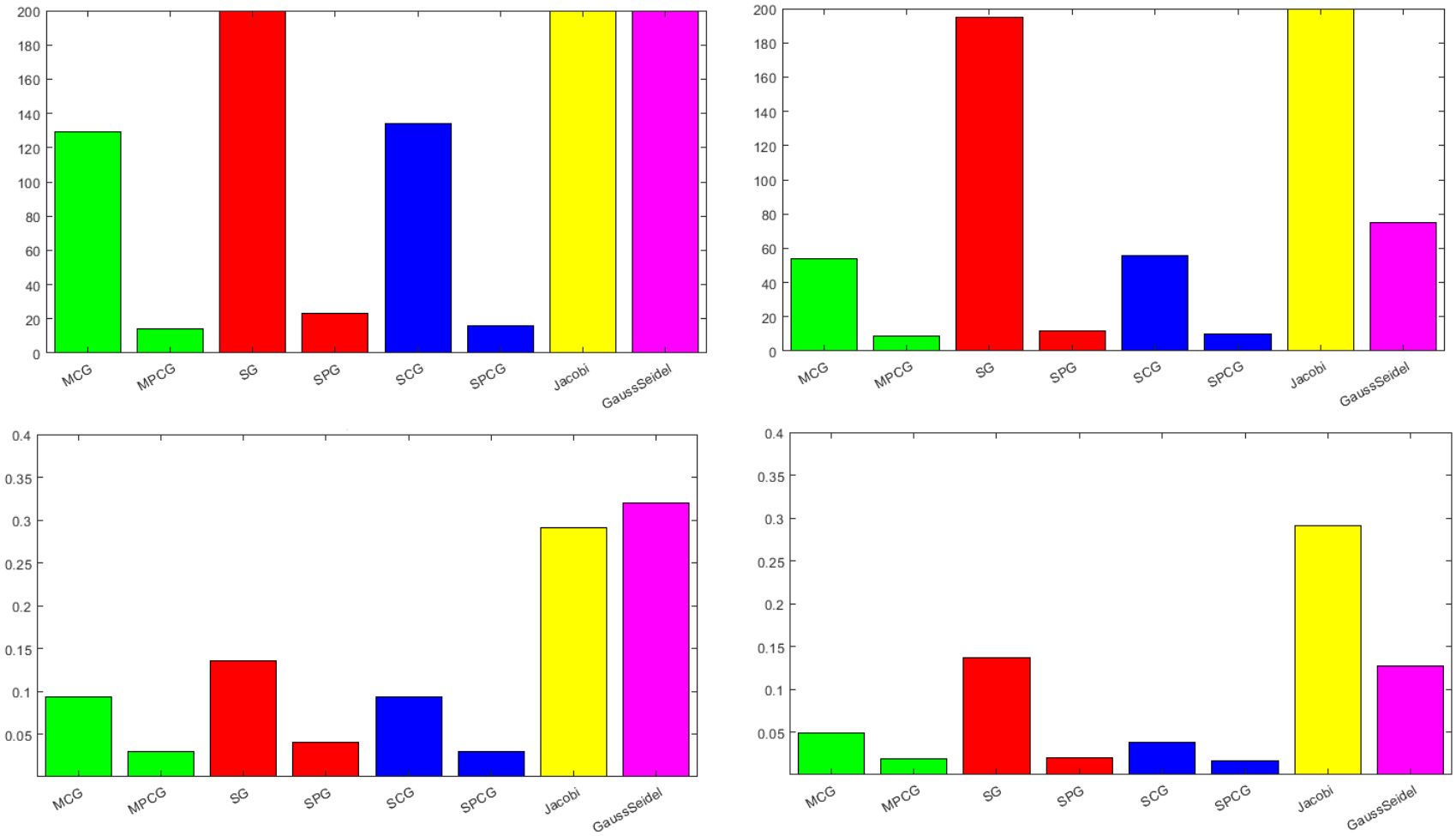

Tabella 10: Test 3.In alto grafici delle iterazioni, in basso dei tempi. A sinistra il condizionamento è pari a 100, a destra è pari a 20.

4.5 Conclusioni finali

In conclusione possiamo affermare che l'algoritmo SPCG, ossia la nostra implementazione del metodo del gradiente coniugato precondizionato, risulta più efficiente nel caso di matrici diagonalmente dominanti di ordine inferiore o vicino a 1000 con le caratteristiche da noi considerate. Sempre per questa tipologia di matrici, per un ordine prossimo a 100 risultano validi anche Jacobi e Gauss-Seidel. Il metodo SPCG risulta efficiente anche nel caso di matrici sparse simmetriche definite positive, per dimensioni persino più elevate, ossia per un ordine inferiore o vicino a 2500. L'algoritmo MPCG di Matlab, risulta il migliore nel primo test per dimensioni prossime a 2500, mentre nel secondo test ipotizziamo che risulti il migliore per dimensioni leggermente più elevate avendo quasi raggiunto SPCG nell'ultimo dimensionamento considerato, ossia 2500. Nel terzo ed ultimo test abbiamo notato come una sistema ben condizionato abbia un effetto positivo su tutti i metodi del gradiente, in particolare su quelli coniugati e precondizionati. Significativo in questo test è stato inoltre il miglioramento riscontrato sul metodo di Gauss-Seidel. Lo strumento software realizzato consente una rapida impostazione di diverse configurazioni di test, possiede buon grado di modularità qualora si voglia estendere lo spettro dei metodi considerati e si dimostra quindi un valido alleato per l'esecuzione di raffronti e analisi dei metodi iterativi.

5 Codice Matlab degli algoritmi

Riportiamo in questa sezione il codice Matlab degli algoritmi implementati.

Algoritmo del Gradiente Classico

```
% Metodo del gradiente classico
% x - soluzione calcolata
% k - numero di iterazioni eseguite
% resvec - vettore che contiene il residuo ad ogni iterazione
function [x,k,resvec] = SelfGradient(A,b,tau,maxn,x)
% Determino la dimensione della matrice
n=size(A,1);
% Inizializzo il vettore x0
x0=100*ones(n,1);
% Inizializzazione residuo
r = b - A*x;
% Inizializzazione contatore iterazioni
k = 0;
% Avvio ciclo dell'algoritmo con condizione di controllo, posso
    scegliere
% tra:
% - controllo del residuo: norm(r)>tau*norm(b)
% - condizione di Cauchy: norm(x-x0)>tau*norm(x)
% Imposto comunque un numero di iterazioni massimo
% Preallocazione risorse per il vettore residuo
resvec=zeros(maxn,1);

while(norm(x-x0)>tau*norm(x)) && (k<maxn)
    x0=x;
    k=k+1;
    % Memorizzo la norma del residuo nel vettore
    resvec(k)=norm(r);
    % Ottimizzo calcolando solo una volta il prodotto matrice
    vettore:
    s = A*r;
    % Calcolo del passo:
    alpha = (r'*r)/(r'*s);
    % Calcolo nuovo soluzione
    x=x0+alpha*r;
    % Aggiornamento residuo
    r=r-alpha*s;

end
```

Listato 1: Metodo Self Gradient

n	rc	MCG (i)	MPCG (i)	SG (i)	SPG (i)	SCG (i)	SPCG (i)	Jacobi (i)	G. Seidel (i)
2500	0.01	129	14	200	23	134	16	200	200
2500	0.05	54	9	195	12	56	10	200	75

Tabella 11: Test 3 - Iterazioni

n	rc	MCG (t)	MPCG (t)	SG (t)	SPG (t)	SCG (t)	SPCG (t)	Jacobi (t)	G. Seidel (t)
2500	0.01	0.093306	0.029104	0.1356	0.039866	0.093054	0.029489	0.29111	0.32013
2500	0.05	0.048343	0.01903	0.13707	0.01958	0.038124	0.016674	0.29134	0.12768

Tabella 12: Test 3 - Tempi

n	rc	MCG (e)	MPCG (e)	SG (e)	SPG (e)	SCG (e)	SPCG (e)	Jacobi (e)	G. Seidel (e)
2500	0.01	7.2819e-12	5.9652e-12	0.0027733	3.2244e-13	2.6375e-12	1.3798e-13	5.6382e+137	2.1441e-07
2500	0.05	2.4956e-12	6.3562e-13	5.0338e-12	3.4338e-14	9.0568e-13	2.8007e-14	8.9017e+73	2.5987e-12

Tabella 13: Test 3 - Errore relativo

Algoritmo del Gradiente Classico Precondizionato

```
% Metodo del gradiente classico precondizionato
% x - soluzione calcolata
% k - numero di iterazioni eseguite
% resvec - vettore che contiene il residuo ad ogni iterazione
function [x,k,resvec] = SelfPreGradient(A,b,tau,maxn,Rt,R,x)
% Determino la dimensione della matrice
n=size(A,1);
% Inizializzo il vettore x0
x0=100*ones(n,1);
% Inizializzazione residuo
r = b - A*x;
% Risoluzine del sistema lineare
y=Rt\r;
z=R\y;
% Inizializzazione contatore iterazioni
k = 0;
% Avvio ciclo dell'algoritmo con condizione di controllo, posso
    scegliere
% tra:
% - controllo del residuo: norm(r)>tau*norm(b)
% - condizione di Cauchy: norm(x-x0)>tau*norm(x)
% Imposto comunque un numero di iterazioni massimo
% Preallocazione risorse per il vettore residuo
resvec=zeros(maxn,1);
while(norm(x-x0)>tau*norm(x)) && (k<maxn)
    x0=x;
    k=k+1;
    % Memorizzo la norma del residuo nel vettore
    resvec(k)=norm(r);
    % Ottimizzo calcolando solo una volta il prodotto matrice
    vettore:
    s = A*z;
    % Calcolo del passo:
    alpha = (z'*r)/(z'*s);
    % Calcolo nuovo soluzione
    x=x0+alpha*z;
    % Aggiornamento residuo
    r=r-alpha*s;
    % Risolvo il sistema Pz = r per k+1
    y=R'\r;
    z=R\y;

end
```

Listato 2: Metodo Self Preconditioned Gradient

Algoritmo del Gradiente Coniugato

```
% Metodo del gradiente coniugato
% x - soluzione calcolata
% k - numero di iterazioni eseguite
% resvec - vettore che contiene il residuo ad ogni iterazione
function [x,k,resvec] = SelfConiugGradient(A,b,tau,maxn,x)
% Determino la dimensione della matrice
n=size(A,1);
% Inizializzo il vettore x0
x0=100*ones(n,1);
% Inizializzazione residuo
r = b - A*x;
% Inizializzazione direzione iniziale
p = r;
% Inizializzazione contatore iterazioni
k = 0;
% Avvio ciclo dell'algoritmo con condizione di controllo, posso
    scegliere
% tra:
% - controllo del residuo: norm(r)>tau*norm(b)
% - condizione di Cauchy: norm(x-x0)>tau*norm(x)
% Imposto comunque un numero di iterazioni massimo
% Preallocazione risorse per il vettore residuo
resvec=zeros(maxn,1);
while(norm(x-x0)>tau*norm(x)) && (k<maxn)
    x0=x;
    k=k+1;
    % Memorizzo la norma del residuo nel vettore
    resvec(k)=norm(r);
    % Ottimizzo calcolando solo una volta il prodotto matrice
    vettore:
    s = A*p;
    delta = p'*s;
    % Calcolo del passo:
    alpha = (p'*r)/delta;
    % Calcolo nuovo soluzione
    x=x0+alpha*p;
    % Aggiornamento residuo
    r=r-alpha*s;
    beta = s'*r/delta;
    p = r-beta*p;
end
```

Listato 3: Metodo Self Conjugate Gradient

Algoritmo del Gradiente Coniugato Precondizionato

```
% Metodo del gradiente coniugato precondizionato
% x - soluzione calcolata
% k - numero di iterazioni eseguite
% resvec - vettore che contiene il residuo ad ogni iterazione
function [x,k,resvec] = SelfPreConiugGradient(A,b,tau,maxn,Rt,R,x)
% Determino la dimensione della matrice
n=size(A,1);
% Inizializzo il vettore x0
x0=100*ones(n,1);
% Inizializzazione residuo
r = b - A*x;
% Risoluzine del sistema lineare
y=Rt\r;
z=R\y;
% Inizializzazione direzione iniziale
p = z;
% Inizializzazione contatore iterazioni
k = 0;
% Avvio ciclo dell'algoritmo con condizione di controllo, posso
    scegliere
% tra:
% - controllo del residuo: norm(r)>tau*norm(b)
% - condizione di Cauchy: norm(x-x0)>tau*norm(x)
% Imposto comunque un numero di iterazioni massimo
% Preallocazione risorse per il vettore residuo
resvec=zeros(maxn,1);
while(norm(x-x0)>tau*norm(x)) && (k<maxn)
    x0=x;
    k=k+1;
    % Memorizzo la norma del residuo nel vettore
    resvec(k)=norm(r);
    % Ottimizzo calcolando solo una volta il prodotto matrice
    vettore:
    s = A*p;
    delta = p'*s;
    % Calcolo del passo:
    alpha = (p'*r)/delta;
    % Calcolo nuovo soluzione
    x=x0+alpha*p;
    % Aggiornamento residuo
    r=r-alpha*s;
    y=R'\r;
    z=R\y;
    beta = s'*z/delta;
    p = z-beta*p;
end
```

Listato 4: Metodo Self Preconditioned Conjugate Gradient

Algoritmo Jacobi

```
% Metodo di Jacobi
% x - soluzione calcolata
% k - numero di iterazioni eseguite
% resvec - vettore che contiene il residuo ad ogni iterazione
function [x,k,resvec] = Jacobi(A,b,tau,maxn,x)
% Individuo dimensione del sistema
n=size(A,1);
% Preallocazione risorse
x0=100*ones(n,1);
resvec=zeros(maxn,1);
% Impostazione dello splitting additivo
d=diag(A);
EF = A - diag(d);
k=0;
% Ciclo di iterazione con criterio di Cauchy
while(norm(x-x0)>tau*norm(x)) && (k<maxn)
    x0 = x;
    k = k+1;
    % Calcolo del residuo
    r = b-A*x;
    % Memorizzazione del residuo
    resvec(k) = norm(r);

    step = b - EF*x0;
    x = (step)./d;
end
```

Listato 5: Metodo di Jacobi

Algoritmo Gauss-Seidel

```
% Metodo del Gauss-Seidel
% x - soluzione calcolata
% k - numero di iterazioni eseguite
% resvec - vettore che contiene il residuo ad ogni iterazione
function [x,k,resvec] = GaussSeidel(A,b,tau,maxn,x)
% Individuo dimensione del sistema
n=size(A,1);
% Preallocazione risorse
x0=100*ones(n,1);
resvec=zeros(maxn,1);
% Impostazione dello splitting additivo
P = tril(A);
N = P - A;
k=0;
% Ciclo di iterazione con criterio di Cauchy
while(norm(x-x0)>tau*norm(x)) && (k<maxn)
    x0 = x;
    k = k+1;
    % Calcolo del residuo
    r = b-A*x;
    % Memorizzazione del residuo
    resvec(k) = norm(r);

    step = N*x0+b;
    x = P\(step);

end
```

Listato 6: Metodo di Gauss-Seidel