Note brevi e diario per il corso di

# *Calcolo Numerico 2*

Elena Gaburro, elena.gaburro@univr.it

AA 2024-2025

Questi appunti non hanno alcuna pretesa di completezza. Sono solo alcune note che accompagnano il corso di Calcolo Numerico 2 dell'AA 2024-2025.

Queste note sono organizzate come segue:

- quando c'è letteratura disponibile sull'argomento della lezione, fanno solo riferimento alla letteratura, dando indicazioni precise sulle sezioni da studiare;
- quando la letteratura disponibile richiede troppi prerequisiti o il livello di dettaglio non è adeguato a questo corso, la letteratura viene comunque citata ma le note saranno più dettagliate;
- inoltre, sono descritti gli algoritmi da implementare durante le sessioni di laboratorio.

In ogni caso, prendere appunti e partecipare attivamente alle sessioni di laboratorio è considerato di fondamentale importanza.

Infine, questi appunti sono da considerarsi in perenne "under revision" e quindi possono contenere discrepanze, imprecisioni o errori.

"*Behind every result is a new challenge*"

# Indice

# Capitolo 1

# Introduzione

L'introduzione al corso è stata fatta durante la **Lezione 1. Venerdì 4 Ottobre 2024.**

La lezione è cominciata con una breve presentazione per introdurre gli argomenti del corso e le loro motivazioni; inoltre, sono state presentate le modalità e gli orari del corso e le modalità d'esame. Vedere `Calcolo2_Intro_Motivation.pdf`.

# Capitolo 2

# Metodi iterativi per la soluzione di sistemi lineari

## 2.1 Lezioni e referenze

Questo capitolo è stato affrontato durante

- **Lezione 1. Venerdì 4 Ottobre 2024.**
- **Lezione 2 (lab). Mercoledì 9 Ottobre 2024.**
- **Lezione 3. Venerdì 11 Ottobre 2024.**
- **Lezione 4. Lunedì 14 Ottobre 2024.**
- **Lezione 5 (lab). Mercoledì 16 Ottobre 2024.**

Si consiglia una rapida lettura dei Capitoli 3.1, 3.2, 3.4.1 del libro [1] per un ripasso della terminologia base dell'Algebra Lineare.

Ciò che segue è solo una sintesi del libro [1], in particolare

- Introduzione del Capitolo 4,
- Una parte della sezione 4.1.10,
- Capitolo 4.3 (le sezioni 4.3.1, 4.3.3, 4.3.4).

Si rimanda quindi al libro per completezza.
Infatti, tutte le sezioni e sottosezioni indicate sono state trattate a lezione in modo completo (anche se in questi appunti sono vuote o parziali).

Inoltre a laboratorio sono stati fatti interamente alcuni degli esercizi e sono state date le indicazioni per riuscire a svolgerli tutti.

## 2.2 Introduzione

Oggetto principale di questo capitolo è la risoluzione numerica dei sistemi lineari, ossia dei problemi che possono essere formulati nella seguente forma.

**Sistema lineare.** Dato un vettore $\mathbf{b} \in \mathbb{R}^m$ e una matrice $\mathbf{A} = [aij], i = 1, 2, \ldots, m; j = 1, 2, \ldots, n$ ad elementi reali, si cerca un vettore $\mathbf{x} \in \mathbb{R}^n$ tale che $A\mathbf{x} = \mathbf{b}$.

**Tipologie di metodi numerici per la soluzione di sistemi lineari.** Per risolvere un sistema lineare esistono diversi metodi tra cui scegliere a seconda della situazione e del tipo di matrice.

Esistono matrici con una struttura molto particolare: matrici diagonali, triangolari o ortogonali. Per queste esistono algoritmi di soluzione immediati, che ci evitano sia di calcolare l'inversa, sia di applicare procedure costose.

Quando invece si considera una matrice generica, esistono sostanzialmente due classi di metodi:

- I *metodi diretti*
- I *metodi indiretti*

**Matrice sparsa** Si dice *sparsa* una matrice di dimensione $n \times n$ in cui la quantità di elementi non nulli è dell'ordine di $O(n)$. In tal caso per esempio memorizzare la matrice ha un costo ridotto perché vengono memorizzati solo gli elementi non nulli. Però in generale data una matrice sparsa, le matrici ottenute fattorizzandola non sono sparse e il costo di memorizzazione aumenta: è questo uno dei motivi per cui non si sceglie di usare un metodo diretto per risolvere sistemi con matrici sparse.

**Inversa di una matrice e comando `inv` di Matlab** N.B. In generale non è consigliabile risolvere un sistema mediante il calcolo diretto dell'inversa della matrice del sistema. Infatti gli algoritmi per il calcolo diretto dell'inversa sono fortemente malcondizionati.

Al contrario, un metodo numerico per la risoluzione di un sistema lineare può essere utilizzato anche per il calcolo dell'inversa di una matrice. Infatti, le colonne $X_j, j = 1, 2, \ldots, n$ della matrice inversa $\mathbf{A}^{-1}$ possono essere calcolate mediante la risoluzione dei seguenti $n$ sistemi lineari, tutti con la stessa matrice dei coefficienti $\mathbf{A}$

$$\mathbf{A}X_j = \mathbf{e}_j \rightarrow \mathbf{A}^{-1} = [X_1, X_2, \ldots, X_n],$$

con $\mathbf{e}_j$ vettori elementari.

Nei laboratori, negli esercizi e durante l'esame non è permesso utilizzare il comando `inv` di Matlab, se non a scopo di confronto. (Il comando `inv` è stato ottimizzato nel corso degli anni, quindi spesso è affidabile, ma raramente calcola l'inversa in modo diretto. Analizza la matrice e sceglie tra emtodi diretti e indiretti la cosa che ritiene migliore. Non potendo controllare ciò che fa in realtà non è opportuno il suo uso nel contesto di un corso di Matematica).

## 2.3 Matrici sparse

In termini qualitativi, una matrice è considerata sparsa quando il suo ordine è sufficientemente elevato, ma i suoi elementi sono in "predominanza" nulli. Più precisamente, si dice *sparsa* una matrice di dimensione $n \times n$ in cui la quantità di elementi non nulli è dell'ordine di $O(n)$ invece di $O(n^2)$.

Le matrici a banda (triangolari, diagonali, ...) sono casi particolari di matrici sparse. Ma, più in generale, nelle applicazioni una matrice sparsa può avere gli elementi diversi dallo zero non necessariamente addensati intorno alla diagonale.

Per tali matrici si pone da una parte il problema di una loro conveniente rappresentazione in memoria, e dall'altra l'individuazione di opportune strategie per la loro risoluzione. Si ricorda che con i metodi diretti si rischia il fenomeno del *fill-in*. Per *fill-in* si intende il fatto che un elemento della matrice originariamente nullo diventa diverso dallo zero per effetto di una sostituzione, tipica dei metodi diretti.

### 2.3.1 Alcuni comandi per matrici sparse

É utile ricordare che in matlab esistono i seguenti comandi per rappresentare in modo economico dal punto di vista della memoria le matrici sparse.

- Il comando `speye(n)` genera la matrice identità di ordine $n$.
- Il comando `spdiags(v,0,n,n)`, ove `v` è un vettore colonna, genera la matrice diagonale di ordine $n$ avente `v` in diagonale. Se la dimensione di `v` è minore di $n$, la diagonale viene riempita con zeri posti dopo il vettore `v`. Se invece la dimensione di `v` è maggiore di $n$, vengono usate solo le prime $n$ componenti di `v`. Inoltre sia `V` la matrice
$$\mathbf{V} = \begin{pmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \\ \vdots & \vdots & \vdots \\ v_{n1} & v_{n2} & v_{n3} \end{pmatrix},$$
il comando `spdiags(V,-1:1,n,n)` genera la matrice
$$\begin{pmatrix} v_{12} & v_{23} & 0 & 0 & \dots & 0 \\ v_{11} & v_{22} & v_{33} & 0 & \dots & 0 \\ 0 & v_{21} & v_{32} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & v_{n-2,1} & v_{n-1,2} & v_{n,3} \\ 0 & \dots & \dots & 0 & v_{n-1,1} & v_{n,2} \end{pmatrix}.$$
- il comando `sparse` memorizza la matrice in formato sparso per *colonne*, detto *Compressed Column Storage (CCS)*.
- il comando `full` permette di visualizzare in modo completo una matrice salvata in modo sparso.

## 2.4 L'idea generale alla base dei metodi iterativi

Svolto interamente a lezione.

### 2.4.1 Studio della convergenza

Svolto interamente a lezione.

### 2.4.2 Test di arresto per un metodo iterativo

I metodi presentati sono iterativi: ciò significa che a partire da una soluzione approssimata (un qualsiasi *guess* iniziale) ne cercano, con successive iterazione, una sempre più vicina a quella reale (se convergono). Bisogna quindi stabilire un criterio detto *criterio d'arresto* che quando risulta verificato fa interrompere questa ricerca. Vi sono due test principali che possono essere utilizzati:

- Test sullo *scarto* tra due iterate: si potrà fermare un metodo iterativo all'iterazione $k+1$ quando $||x^{(k+1)} - x^{(k)}||_\infty < tol_1 + tol_2||x^{(k)}||$ ove $tol_1$ è una tolleranza prefissata sull'errore assoluto e $tol_2$ è una tolleranza sull'errore relativo.
- Test sul *residuo* (se $\mathbf{b} \neq 0$ ): si potrà fermare un metodo all'iterazione $k$ quando $\frac{||r^{(k)}||_2}{||b||_2} < tol$ dove $r^{(k)} = b - Ax^{(k)}$.

## 2.5 Metodo di Jacobi

Vediamo una prima idea per ricavare un'iterazione di punto fisso da cui ottenere ogni componente $x_i$ del vettore $\mathbf{x}$ tale che $\mathbf{Ax} = \mathbf{b}$.

Usiamo la riga $i$ del sistema per isolare $x_i$

$$\begin{aligned} &\sum_{j=1}^{n} a_{ij}x_j = b_i \\ &a_{ii}x_i + \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}x_j = b_i \\ &a_{ii}x_i = b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}x_j \quad \longrightarrow \quad x_i = \frac{1}{a_{ii}}\left\{ b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}x_j \right\}. \end{aligned} \tag{2.1}$$

Da cui si deduce l'iterazione di punto fisso scritta per componenti

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left\{ b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}x_j^{(k)} \right\}, \quad a_{ii} \neq 0, \quad i = 1, 2, \cdots, n, \tag{2.2}$$

che può anche essere riscritta anche nella forma

$$\underbrace{a_{ii}x_i^{(k+1)}}_{j=i} = b_i - \underbrace{\sum_{j=1}^{i-1} a_{ij}x_j^{(k)}}_{j<i} - \underbrace{\sum_{j=i+1}^{n} a_{ij}x_j^{(k)}}_{j>i}, \tag{2.3}$$

i.e.

$$\underbrace{a_{ii}x_i^{(k+1)}}_{\text{diagonale}} = b_i - \underbrace{\sum_{j=1}^{i-1} a_{ij}x_j^{(k)}}_{\text{sotto diagonale}} - \underbrace{\sum_{j=i+1}^{n} a_{ij}x_j^{(k)}}_{\text{sovra diagonale}}. \tag{2.4}$$

Quest'ultima formulazione è ora semplice da riscrivere in forma matriciale e manipolare come segue

$$\begin{aligned}\mathbf{D}\mathbf{x}^{(k+1)} &= \mathbf{b} - (\mathbf{L}+\mathbf{U})\mathbf{x}^{(k)} \\ \mathbf{x}^{(k+1)} &= \mathbf{D}^{-1}\mathbf{b} - \mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})\mathbf{x}^{(k)}.\end{aligned} \tag{2.5}$$

Chiamiamo

$$\mathbf{J} = -\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U}) \tag{2.6}$$

la matrice associata all'iterazione di Jacobi. Infine, notiamo che l'iterazione di Jacobi è facilmente *parallelizzabile* poiché ogni componente $x_i^{(k+1)}$ può essere calcolata indipendentemente dalle altre $x_j^{(k+1)}$.

### 2.5.1 Convergenza del metodo di Jacobi

Svolto interamente nella lezione di laboratorio.

## 2.6 Metodo di Gauss-Seidel

Ricordiamo l'iterazione di Jacobi

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left\{ b_i - \underbrace{\sum_{j=1}^{i-1} a_{ij}x_j^{(k)}}_{\text{i precedenti}} - \underbrace{\sum_{j=i+1}^{n} a_{ij}x_j^{(k)}}_{\text{i successivi}} \right\}, i = 1, 2, \cdots, n$$

e ricordiamo che permette di calcolare ogni $x_i^{(k+1)}$ indipendentemente dagli altri $x_j^{(k+1)}$. Allora, quando stiamo per calcolare $x_2{}^{(k+1)}$, la nuova stima, $x_1{}^{(k+1)}$ è già disponibile! E quando stiamo per calcolare $x_3{}^{(k+1)}$, le nuove stime, $x_1{}^{(k+1)}$ e $x_2{}^{(k+1)}$ sono già disponibili!

L'idea di Gauss-Seidel è dunque usare le nuove stime subito quando sono disponibili! Possiamo quindi scrivere l'algoritmo di Gauss-Seidel per componenti

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left\{ b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)} \right\}, i = 1, 2, \cdots, n \tag{2.7}$$

con il vincolo che $a_{ii} \neq 0$, $1 \leq i \leq n$.

Inoltre, in forma matriciale si può scrivere come

$$\mathbf{x}^{(k+1)} = (\mathbf{D}+\mathbf{L})^{-1}\mathbf{b} - (\mathbf{D}+\mathbf{L})^{-1}\,\mathbf{U}\mathbf{x}^{(k)}, \tag{2.8}$$

dove la *matrice di iterazione* è definita come segue

$$\mathbf{G} = -(\mathbf{D}+\mathbf{L})^{-1}\,\mathbf{U}. \tag{2.9}$$

### 2.6.1 Convergenza del metodo di Gauss-Seidel

Le condizioni sufficienti per la convergenza di Gauss-Seidel sono state enunciate a lezione di laboratorio.

## 2.7 Metodo di SOR

Un modo semplice per accelerare la convergenza dei metodi di Jacobi e di Gauss-Seidel consiste nell'introduzione nella matrice di iterazione di un opportuno parametro, noto come *parametro di rilassamento*. Da questa idea nasce il metodo SOR (Successive Over-Relaxation).

Infatti l'iterazione di Gauss-Seidel si può modificare come segue inserendo il parametro reale di accelerazione $\omega$ nel seguente modo:

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right), \quad i = 1, \ldots, n. \quad (2.10)$$

Per $\omega < 1$ si ha il metodo di sottorilassamento, per $\omega = 1$ il metodo coincide con l'iterazione di Gauss-Seidel mentre per $\omega > 1$ abbiamo il metodo di sovrarilassamento.

La formula precedente scritta in forma matriciale diventa

$$\mathbf{x}^{(k+1)} = (\omega L + D)^{-1}[(1-\omega)D - \omega U]\mathbf{x}^{(k)} + \omega(\omega L + D)^{-1}b,$$

in cui la matrice di iterazione risulta (dipendente dal parametro $\omega$)

$$B_\omega = (\omega L + D)^{-1}[(1-\omega)D - \omega U] \quad (2.11)$$

Nell'applicazione del metodo SOR è naturalmente importante la scelta del parametro $\omega$. Tale scelta deve essere tale da rendere più elevata possibile la velocità di convergenza del metodo. Tale obiettivo è raggiunto dal valore di $\omega$ che minimizza il raggio spettrale della matrice di rilassamento $B_\omega$.

## 2.8 Metodo del gradiente coniugato

**(o metodo di *Hestenes-Stiefel*, 1952)**

Spiegato con tutti i dettagli in aula. Si richiede una conoscenza approfondita di questo metodo.

### 2.8.1 Altri metodi basati sulla stessa idea di minimizzazione

**Metodo del gradiente.** É una versione meno sofisticata del metodo appena visto che si limita a minimizzare $f(x)$ seguendo sempre la direzione del gradiente, i.e. il residuo. Converge più lentamente del metodo del gradiente coniugato perchè non sceglie direzioni ortogonali nelle successive iterazioni del metodo.

Ecco qui le formule base per implementare l'algoritmo (notare che in realtà basta semplificare il precedente). Si pone

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} p^{(k)} \tag{2.12}$$

con

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

e la costante $\alpha^{(k)}$

$$\alpha^{(k)} = \frac{{p^{(k)}}^T r^{(k)}}{{p^{(k)}}^T A p^{(k)}} = \frac{{r^{(k)}}^T r^{(k)}}{{r^{(k)}}^T A r^{(k)}}.$$

**Metodo di Gauss-Seidel.** Anche il metodo di Gauss-Seidel potrebbe essere visto come un metodo di minimizzazione della funzione $f(x) = \frac{1}{2} x^T A x - b^T x$ eseguita seguendo le direzioni degli assi coordinati.

## 2.9 Calcolo dell'inversa di una matrice

A lezione abbiamo visto come evitare il calcolo diretto dell'inversa di una matrice anche quando lo scopo principale non è risolvere un sistema lineare. Negli esercizi numerici potrà quindi essere richiesto di evitare il calcolo dell'inversa e di adottare invece altre strategie.

## 2.10 Esercizi numerici

**Esercizio .1** (Convergenza metodi iterativi, autonomamente in aula per Gauss-Seidel)**.** Esporre dei criteri per verificare che i metodi di Jacobi e Gauss Seidel convergono per le seguenti matrici ed eventualmente verificarli numericamente.

$$A1 = \begin{pmatrix} 3 & 0 & 4 \\ 7 & 4 & 2 \\ -1 & 1 & 2 \end{pmatrix} \quad A2 = \begin{pmatrix} -3 & 3 & -6 \\ -4 & 7 & -8 \\ 5 & 7 & -9 \end{pmatrix} \quad A3 = \begin{pmatrix} 4 & 1 & 1 \\ 2 & -9 & 0 \\ 0 & -8 & 6 \end{pmatrix} \quad A4 = \begin{pmatrix} 7 & 6 & 9 \\ 4 & 5 & -4 \\ -7 & -3 & 8 \end{pmatrix}$$

**Suggerimenti operativi**:

- Verificare se sussistono condizioni *sufficienti* di *facile* verifica.
- In caso contrario, enunciare la condizione *necessaria e sufficiente* per la convergenza.
- Calcolare lo spettro della matrice di iterazione di Jacobi $J$, definita in (2.6).
- Calcolare lo spettro della matrice di iterazione di Gauss-Seidel $G$, definita in (2.9)
- Dire, analizzando gli spettri di tali matrici, se tali metodi potranno convergere per qualunque valore di $\mathbf{x}_0$ (facendo riferimento al Teorema **??**.

**Suggerimenti pratici**:

- in questo esercizio è consentito usare il comando inv di matlab
- per estrarre la diagonale, la parte triangolare inferiore o superiore di una matrice considerare l'uso dei comandi `diag, tril, triu`. Considerare anche l'uso di `spdiags`.
- Per spettro si intende l'insieme degli autovalori di una matrice; possiamo rappresentarli sul piano complesso (con i comandi `real` e `imag` si estraggono le parti reali e immaginarie di un numero complesso). Ad esempio per la matrice $A1$ si dovrebbe ottenere un risultato simile a quello riportato in Figura 2.1.

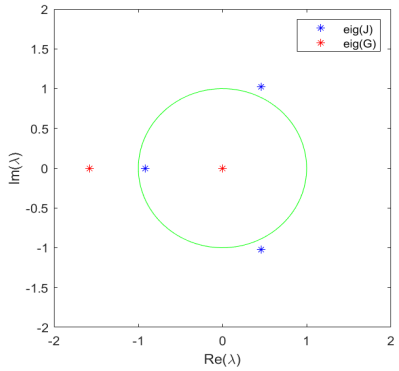


Figura 2.1: Rappresentazione dello spettro d A1.

**Esercizio .2** (Jacobi)**.** Scrivere una `routine` $MATLAB$ che implementi l'iterazione di Jacobi in (2.4). Provare tale routine sulle matrici dell'esercizio precedente scegliendo il termine noto $b$ in modo che la soluzione di riferimento sia fatta da tutti 1.

Suggerimenti per l'implementazione della `routine` per Jacobi:

- La function deve prendere come argomenti in **input** la matrice del sistema $A$, il termine noto $b$, un'approssimazione iniziale della soluzione $x_0$, la tolleranza con cui si vuole cercare la soluzione `tol`, e il numero massimo di iterazioni che concediamo al metodo `maxiter`.
- La function deve restituire in **output**: la soluzione del sistema (a meno della tolleranza concessa) $x$, una variabile booleana che indichi se il metodo è arrivato a convergenza oppure no, e una matrice `xhist` sulle cui colonne siano presenti tutte le soluzioni calcolate durante le varie iterazioni (in particolare sulla prima colonna deve esserci $x_0$ e sull'ultima $x$).
- Prima fase: inizializzare `xhist`, la variabile booleana, il contatore delle iterazioni e una variabile `scarto` per il test di arresto (vedi sezione 2.4.2).
- Seconda fase: costruzione di un ciclo `while` che terminerà o una volta trovata una soluzione abbastanza precisa (che supera il test dello scarto), o al raggiungimento del numero massimo di iterazioni concesse.
- All'interno del ciclo `while` bisogna implementare, attraverso dei cicli `for` la formula in (2.4): la quale permette di trovare ciascuna componente della nuova soluzione a partire dalle componenti della vecchia.
- Aggiornare le variabili `scarto`, $x$, e `xhist`.
- Dopo il ciclo `while` occuparsi della variabile booleana che indica la convergenza o meno del metodo e se necessario di `xhist`
- Testare per prima cosa il metodo su una matrice, su cui si è certi che il metodo convergerà.

**Esercizio .3** (Gauss-Seidel)**.** Scrivere una `routine` $MATLAB$ che implementi l'iterazione di Gauss-Seidel in (2.7). Provare tale routine sulle matrici dell'esercizio 2 scegliendo il termine noto $b$ in modo che la soluzione di riferimento sia fatta da tutti 1.

Suggerimenti per l'implementazione della `routine` per Gauss-Seidel: la `routine` è sostanzialmente uguale a quella di Jacobi, l'unica differenza sta nel fatto che per calcolare la componente $x(i)$ della nuova soluzione invece di usare tutte le vecchie componenti si usano anche quelle nuove, se si sono già trovate: cioè si usano le $x(1 : i-1)$ nuove e le $x(i+1 : n)$ vecchie.

**Esercizio .4** (Confronto Jacobi vs Gauss-Seidel, matrice differenze finite, autonomamente)**.** La seguente matrice è usata nel contesto del metodo delle *differenze finite* per risolvere un'equazione differenziale, in particolare si usa per discretizzare le derivate seconde

```
B = (1/h^2)*toeplitz(sparse(1,[1,2],[-2,1],1,n),sparse(1,[1,2],[-2,1],1,n));
```

Scegliere per esempio $n = 10$ e $h = 0.1$.

a) Studiare la convergenza del metodo di Jacobi e di Gauss-Seidel su questa matrice analizzando lo spettro delle matrici d'iterazione (il comando `eigs` calcola i 6 autovalori in modulo maggiori di una matrice sparsa).

b) Scegliere un termine noto $b$ in modo che la soluzione di riferimento del sistema sia fatta da tutti 1. Risolvere il sistema sia con Jacobi che con Gauss-Seidel.

c) Servendosi della matrice `xhist` restituita dai due metodi, tracciare un grafico in scala semi-logaritmica dell'andamento dell'errore relativo.

I risultati ottenuti dovrebbero essere simili a quelli riportati in Figura 2.2. (Il numero di iterazioni dipende molto dalla tolleranza e dall'approssimazione iniziale scelte. Inoltre usando come soluzione di riferimento quella costruita apposta nel secondo grafico della Figura vengono proprio 2 rette; se invece si usa come soluzione di riferimento l'ultima trovata dal vostro metodo allora si ottiene il grafico curvo come mostrato nella Figura.).

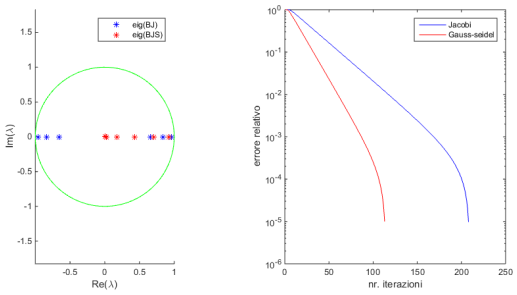


Figura 2.2: Spettro delle matrici di iterazione dei metodi di Jacobi e Gauss-Seidel e analisi della convergenza.

**Esercizio .5** (Condizioni sufficienti per la convergenza, autonomamente, in classe)**.** Si consideri la matrice seguente:

```
C = toeplitz(sparse(1,[1,3],[-3,1],1,n),sparse(1,[1,3],[-3,1],1,n)).
```

Dire se i metodi di Jacobi e Gauss-Seidel convergeranno per qualunque `n`, qualunque **b** e qualunque guess iniziale $\mathbf{x}^{(0)}$ quando usati per risolvere un sistema del tipo $C\mathbf{x} = \mathbf{b}$, senza ricorrere al calcolo di autovalori.

**Esercizio .6** (Test di arresto, autonomamente, in classe)**.** Modificare le routine di Jacobi e/o Gauss Seidel affinchè il test di arresto

a) sia basato sullo *scarto*, su una tolleranza assoluta e anche su una tolleranza *relativa*;

b) sia basato sul *residuo*.

**Suggerimento**: Rileggere la sezione 2.4.2.

**Esercizio .7** (Matrici sparse, autonomamente)**.** Testare i comandi per matrici sparse elencati nella sezione 2.3.1.

**Esercizio .8** (SOR, autonomamente in classe)**.**

a) Scrivere una `routine` $MATLAB$ che implementi l'iterazione di SOR in (2.10).

b) Provare tale routine sulla matrice che si ottiene attraverso il comando `full(gallery('poisson',3))` scegliendo il termine noto $b$ in modo che la soluzione di riferimento sia fatta da tutti 1, con $\omega = 1.1$.

c) Riportare l'andamento sia dell'errore relativo (lo scarto diviso la miglior approssimazione disponibile della soluzione) sia del residuo ($||b - Ax^{(k)}||_2$) e confrontarlo con gli andamenti dell'errore relativo e del residuo ottenuti con il metodo di Jacobi e di Gauss-Seidel. (Scala semi-logaritmica)

**Suggerimenti**:

- A partire dalla `routine` per Gauss-Seidel modificarla appropriatamente seguendo la formula in (2.10), in particolare il metodo SOR deve prendere in ingresso anche il parametro $\omega$.
- I grafici ottenuti devono essere simili a quelli riportati in Figura 2.3.

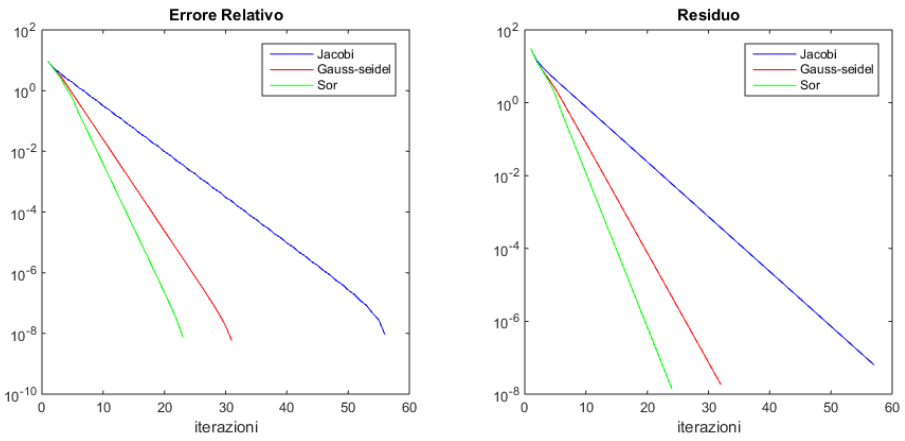


Figura 2.3: Errore relativo e Residuo: confronto.

**Esercizio .9** (SOR, autonomamente)**.** Si consideri la matrice ottenuta mediante il comando
`full(gallery('poisson',3))`.

a) Si determini per quale valore di $\omega \in [0, 2]$ il raggio spettrale della matrice d'iterazione $B_w$ in (2.11) è minimo. Si indichi tale valore con $\omega_{\mathbf{ott}}$ (vietato usare `inv`). Si riporti in un grafico il valore del raggio spettrale in corrispondenza di $\omega$.

b) Si studi la convergenza (tramite un grafico in scala semi-logaritmica) del metodo SOR con $\omega = (\omega_{\mathbf{ott}} - 0.1, \omega_{\mathbf{ott}}, \omega_{\mathbf{ott}} + 0.1)$.

**Suggerimenti**: I grafici ottenuti dovrebbero essere come quelli riportati in Figura 2.4.

**Esercizio .10.** Nell'esercizio .1 avevamo calcolato le matrici di iterazione di Jacobi e Gauss Seidel attraverso l'uso del comando `inv`. Ripetere l'esercizio sostituendo il calcolo delle matrici inverse con la soluzione di un opportuno sistema lineare.

**Nota.** Evitare il calcolo della matrice inversa (è permesso l'uso del comando \ di Matlab, ma potete provare anche ad evitarlo ed usare routine apposite per la soluzione di sistemi triangolari e diagonali viste a Calcolo Numerico 1).

**Esercizio .11.** Implementare una variante dell'algoritmo di Jacobi e/o dell'algoritmo di Gauss Seidel utilizzando la versione matriciale dell'algoritmo e testare il funzionamento delle routine su alcuni esempi. Variare il tipo di test d'arresto utilizzato.

**Suggerimento**: Fare riferimento agli esercizi .2 e .3: in particolare, con la versione matriciale sarà necessario solo il ciclo `while` e nessun ciclo `for` interno.

Evitare il calcolo della matrice inversa (è permesso l'uso del comando \ di Matlab, ma potete provare anche ad evitarlo ed usare routine apposite per la soluzione di sistemi triangolari e diagonali viste a Calcolo Numerico 1).

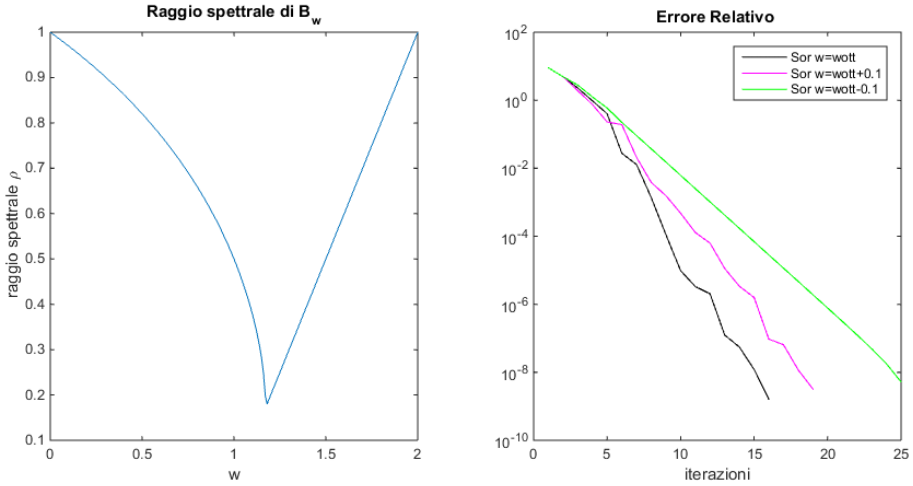


Figura 2.4: Convergenza del metodo SOR al variare di $\omega$.

**Esercizio .12** (Metodo del gradiente)**.** Scrivere una `routine` $MATLAB$ che implementi il metodo del gradiente. Provare tale routine scegliendo il termine noto $b$ in modo che la soluzione di riferimento sia fatta da tutti 1 e usando come matrici quelle che si ottengono con i seguenti comandi

```
A1= full(gallery('poisson',3));
A2= full(gallery('tridiag',10));
```

Riportare infine l'andamento del residuo in funzione del numero di iterazioni in un grafico in scala semi-logaritmica.

**Suggerimento:** Per la seconda matrice, si dovrebbe ottenere un grafico come quello riportato in Figura (2.5).

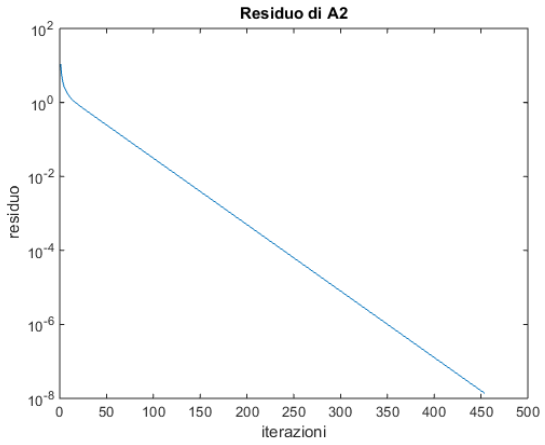


Figura 2.5: Andamento del residuo per la matrice A2 con il metodo del gradiente.

**Esercizio .13** (Metodo del gradiente coniugato)**.** Scrivere una `routine` $MATLAB$ che implementi il metodo del gradiente coniugato. Provare tale routine scegliendo il termine noto $b$ in modo che la soluzione di riferimento sia fatta da tutti 2 e usando come matrici

```
A1= full(gallery('poisson',3));
A2= full(gallery('tridiag',10)).
```

Riportare infine l'andamento dell'errore in funzione del numero di iterazioni e confrontarlo con i risultati ottenuti col metodo del gradiente.
Commentare i risultati ottenuti.

Suggerimenti per l'implementazione della `routine` per il metodo del gradiente coniugato:

- Salvare ad ogni iterazione il risultato del prodotto tra $A$ e $p$ in modo da effettuare il prodotto matrice vettore una sola volta per ciascuna iterazione invece che due.
- I risultati ottenuti per la seconda matrice dovrebbero essere come quelli riportati in Figura 2.6.

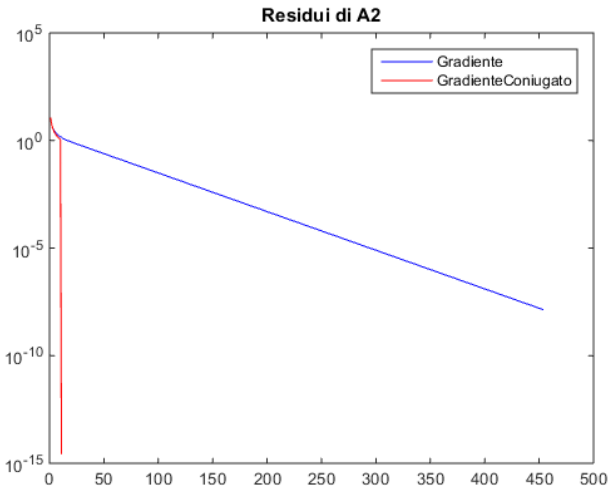


Figura 2.6: Confronto tra metodo del gradiente e del gradiente coniugato.

# Capitolo 3

# Fattorizzazioni di matrici rettangolari e loro applicazioni

## 3.1 Lezioni e referenze

## 3.2 Introduzione

## 3.3 Fattorizzazione QR

Questa sezione sarà trattata seguendo 3.4.3 del libro [2] (solo pag. 84, 85 e 86).

A lezione abbiamo visto l'idea di base per la fattorizzazione **QR** e l'algoritmo base ottenuto dall'algoritmo di *Gram-Schmidt*.

L'algoritmo base è malcondizionato. Lo studio delle tecniche di stabilizzazione dell'algoritmo è facoltativo. Negli esercizi si chiederà o di implementare la versione base dell'algoritmo o di utilizzare il comando `qr` di MATLAB.

### 3.3.1 Uso della fattorizzazione QR per la soluzione delle equazioni normali

## 3.4 Fattorizzazione SVD

## 3.5 Esercizi numerici

**Esercizio .14** (Sistemi triangolari e diagonali)**.** Rivedendo il corso di Calcolo Numerico 1, scrivere delle routine apposite per la soluzione di sistemi diagonali, triangolari inferiori e triangolari superiori che non facciano uso nè di fattorizzazioni, nè del calcolo diretto dell'inversa.

**Esercizio .15** (QR per risolvere un sistema)**.** Risolvere il sistema $Ax = b$ con

$$A = \begin{bmatrix} 4 & 1 & 0 & 1 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ 1 & 2 & 1 & 4 \end{bmatrix}$$

mediante l'uso della fattorizzazione **QR**, cioè seguendo la procedura in (**??**) che porta a risolvere un sistema ortogonale e uno triangolare superiore. Scegliere **b** in modo che la soluzione di riferimento sia un vettore di tutti 1. Commentare le fasi della soluzione.

Suggerimento: per calcolare la fattorizzazione **QR** della matrice usare il comando `qr` di $MATLAB$; per risolvere il sistema triangolare superiore si utilizzi una routine apposita (vedere corso di Calcolo Numerico 1).

**Esercizio .16** (Equazioni normali, esercizio preliminare)**.** Si trovi **z** tale che

$$\mathbf{A}^T\mathbf{A}\mathbf{z} - \mathbf{A}^T\mathbf{b} = 0$$

con

$$\mathbf{A} = \begin{pmatrix} 1 & -5 \\ 1 & -3 \\ 1 & 1 \\ 1 & 3 \\ 1 & 4 \\ 1 & 6 \\ 1 & 8 \end{pmatrix}, \quad \text{e} \quad \mathbf{b} = \begin{pmatrix} 18 \\ 7 \\ 0 \\ 7 \\ 16 \\ 50 \\ 67 \end{pmatrix}, \tag{3.1}$$

i) risolvendo direttamente il sistema e ii) utilizzando la fattorizzazione **QR** di **A**.
Inoltre, calcolare (usando il comando matlab) il numero di condizionamento di **A** e $\mathbf{A}^T\mathbf{A}$.

# Capitolo 4

# Soluzione di sistemi non lineari

# Capitolo 5

# Analisi numerica parallela

# Capitolo 6

# Calcolo di autovalori

# Capitolo 7

# Approssimazione

# Capitolo 8

# Quadratura numerica

# Capitolo 9

# Informazioni riguardanti l'esame

L'esame consiste di:

**A**. Una prova in laboratorio in cui verrà chiesto di implementare alcuni programmi in MATLAB, motivarli e commentare i risultati ottenuti;

**B**. Un esame orale di conoscenze teoriche e competenze.

La prova A si intende superata quando si ottiene un voto maggiore o uguale a 18.
Si è ammessi alla prova B solo quando la prova A risulterà superata.
Il voto complessivo è dato da una media pesata dei voti ottenuti durante le prove A e B.

La **presenza** ad almeno il 75% delle lezioni permetterà di ottenere un +1 sul voto finale.

# Bibliografia

[1] Valeriano Comincioli. *Analisi numerica: metodi, modelli, applicazioni.* Apogeonline (Feltrinelli), 2005.

[2] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Matematica numerica.* Springer Science & Business Media, 2010.