**LSSolver**

Metodi del Calcolo Scientifico

AA 2022-2023

Progetto 1.bis [Algebra lineare numerica]

Mini libreria per sistemi lineari

Lecchi Gabriele - 852134

Titta Lorenzo – 852107

**INTRODUZIONE**

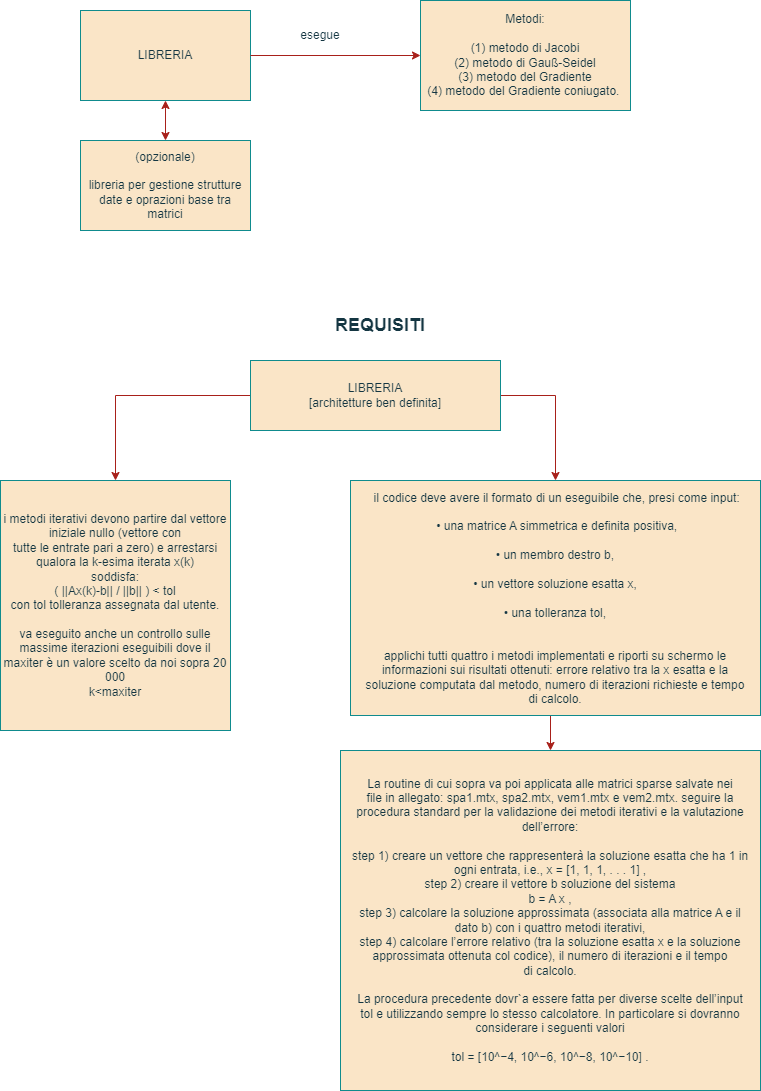
La richiesta di questo progetto prevedeva la creazione di una libreria con un linguaggio a nostra discrezione (nel nostro caso la scelta è ricaduta su Java) che riuscisse ad eseguire i 4 metodi iterativi qui menzionati limitandosi a matrici simmetriche e definite positive:

* Jacobi
* Gauß-Seidel
* Gradiente
* Gradiente coniugato

Per la gestione di: matrici, vettori e operazioni tra di essi ci siamo appoggiati a due librerie offerte da java, ovvero: Maths che è una delle librerie più utilizzate e la4j che per quanto riguarda le operazioni tra matrici e vettori risulta essere una delle più efficienti.

Tutti i metodi richiesti sono stati implementati totalmente da noi senza l’utilizzo di metodi già definiti da librerie di appoggio, di seguito verranno descritti e confrontati.

Il primo passaggio è stato quello di rappresentare i requisiti della richiesta in un grafico per mettere in chiaro i passaggi principali da svolgere, le task principali da implementare e come andare a progettare l’architettura, nella pagina successiva si può vedere il seguente grafico.



**ARCHITETTURA**

Per l’architettura abbiamo deciso di mantenere una struttura il più possibile semplice e pulita senza creare un solo file di pieno di codice.  
Per fare ciò abbiamo diviso la libreria in 5 differenti file (una classe padre e quattro classi figlie):

La classe LSSolver è la classe padre dove vengono istanziate le varie matrici dai differenti file forniti da e-learning. Il principale compito di questa classe è fornire tutto ciò che serve alle classi figlie per l’esecuzione del metodo iterativo necessario per la risoluzione del sistema lineare.

Le restanti 4 classi rappresentano invece i 4 metodi richiesti sotto la forma di classi figlie di LSSolver. In queste classi viene inizialmente istanziato un oggetto della loro classe padre tramite i loro costruttore, e successivamente da un file eseguibile esterno creato appositamente per testare la libreria viene chiamata la funzione che eseguirà effettivamente il metodo generato nella classe riportando tutto a console.

Di seguito viene mostrata un’immagine che rappresenta la disposizione dei file della libreria all’interno del package.

Immagine che contiene testo

Descrizione generata automaticamente

**LSSOLVER**

Come anticipato in precedenza questa classe fa da “padre” a tutte le altre quattro classi che rappresentano i metodi iterativi, abbiamo scelto di utilizzare una classe padre per far si che tutte le iterazioni comuni tra le varie classi dei metodi venissero scritte solamente una volta senza andare a riscrivere ogni volta il codice all’interno di ogni classe di metodo, evitando quindi codice ripetuto.

All’ interno di questa classe è possibile trovare diversi metodi:

* Un metodo **LSSolver**, ovvero il costruttore, che serve per istanziare nuovi oggetti.
* Il metodo **setMatrix** che inizializza le matrici utilizzate per i vari calcoli all’interno del nostro codice (secondo le richieste date).
* All’interno del metodo **executeMethods** vengono inserite le iterazioni uguali per tutti e quattro gli algoritmi (per esempio il controllo sulla tolleranza e il numero di iterazioni) e in cui viene effettivamente risolto il sistema lineare facendo appoggio sul metodo “risoluzione”.
* Un metodo **risoluzione** che viene creato come astratto di conseguenza non presenterà alcun corpo all’interno di questa classe che verrà poi ridefinito in ognuna delle classi figlie dando la possibilità di definire il calcolo della soluzione al passo successivo per ogni metodo di risoluzione.
* I metodi **norma2**, **inversa** e **prodotto scalare** che servono per calcolare ciò che effettivamente viene definito nel loro titolo.
* Infine un metodo **importMtxFile** che permette la lettura dei file .mtx inserendo i valori nelle matrici istanziate precedentemente.

Come accennato precedentemente all’interno di questa classe andiamo a implementare l’esecuzione della risoluzione del sistema lineare con i due criteri di arresto e le tempistiche impiegate da ogni algoritmo per ogni file .mtx fornito con tutte le 4 tolleranze indicate.

Come primo metodo di arresto abbiamo calcolato il cambiamento della soluzione tra un passo e il successivo e l’abbiamo confrontata con la tolleranza secondo questa disequazione che utilizza la norma 2 (da noi descritta in questa classe):

Se questa condizione venisse verificata allora l’iterazione terminerebbe e verrebbe calcolato l’errore come norma della differenza tra la soluzione calcolata e quella esatta.

Mentre per il secondo metodo di arresto il ciclo iterativo verrebbe arrestato nel caso in cui si superasse il numero massimo di iterazioni (20000).

**JACOBI**

Per l’implementazione del metodo di Jacobi abbiamo seguito l’equazione per il calcolo della soluzione al passo successivo:

Con questo metodo applicato alle 4 matrici (spa1, spa2, vem1, vem2) sono stati ottenuti i seguenti risultati:

Seguono i tempi di esecuzione del metodo di Jacobi espressi in secondi:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1.0E-4 | 1.0E-6 | 1.0E-8 | 1.0E-10 |
| Spa1 | 5.144 | 7.211 | 9.229 | 11.32 |
| Spa2 | 21.328 | 29.555 | 37.738 | 45.657 |
| Vem1 | 9.117 | 21.145 | 33.038 | 44.987 |
| Vem2 | 29.792 | 74.149 | 119.318 | 162.679 |

Come si può notare dai grafici questo metodo performa meglio per quanto riguarda l’errore sulle matrici più piccole quali Spa1 e Spa2, inoltre per queste matrici ha un incremento minore del numero di iterazioni al diminuire della tolleranza che risulta in una crescita più graduale dei tempi di esecuzione al diminuire della tolleranza.

**GAUß-SEIDEL**

Per l’implementazione del metodo di Gauß-Seidel (variante del metodo di jacobi) abbiamo seguito l’equazione per il calcolo della soluzione che sfrutta le entrate del vettore x già calcolate durante l’iterazione attuale, come viene mostrato nella seguente formula:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1.0E-4 | 1.0E-6 | 1.0E-8 | 1.0E-10 |
| Spa1 | 0.63 | 0.83 | 1.101 | 1.318 |
| Spa2 | 3.801 | 4.937 | 6.582 | 8.045 |
| Vem1 | 6.662 | 13.927 | 21.262 | 28.32 |
| Vem2 | 22.117 | 49.105 | 76.627 | 103.372 |

Possiamo osservare che anche in questo caso il metodo performa meglio per la risoluzione delle matrici Spa1 e Spa2, che vengono risolte in un tempo più contenuto e con un errore migliore rispetto alle matrici più grandi; infatti, come si vede dal grafo delle iterazioni per le matrici più piccole al diminuire della tolleranza la crescita del numero delle iterazioni è meno ripida rispetto a Vem1 e Vem2.

**GRADIENTE**

Per l’implementazione del metodo del Gradiente abbiamo suddiviso le operazioni per il calcolo della soluzione, inizialmente viene calcolata la direzione di discesa che coincide con il residuo al passo k:

E per andare ad aggiornare la variabile x(k), otterremo la seguente equazione:

Dove:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1.0E-4 | 1.0E-6 | 1.0E-8 | 1.0E-10 |
| Spa1 | 0.225 | 4.716 | 9.331 | 13.598 |
| Spa2 | 1.967 | 22.197 | 54.187 | 85.977 |
| Vem1 | 0.063 | 0.125 | 0.188 | 0.257 |
| Vem2 | 0.131 | 0.301 | 0.455 | 0.612 |

Il gradiente è piuttosto consistente per quanto riguarda l’errore, infatti rispetto ai metodi visti in precedenza questo non cambia molto tra una matrice e l’altra. Nonostante ciò, ha tempi di esecuzione piuttosto dipendenti dalla matrice su cui viene eseguito, che lo porta ad avere un numero di iterazioni che cresce in maniera più accentuata per le matrici Spa1 e Spa2.

**GRADIENTE CONIUGATO**

Possiamo considerarlo come un miglioramento del metodo del Gradiente in quanto si rimedia all’effetto di “convergenza a zig-zag” quando λmin ≪ λmax.

Per il seguente metodo alla k-esima iterazione ci sarà il seguente aggiornamento della soluzione:

dove è:

Nel passo successivo calcoliamo:

Dove:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1.0E-4 | 1.0E-6 | 1.0E-8 | 1.0E-10 |
| Spa1 | 0.225 | 0.309 | 0.434 | 0.498 |
| Spa2 | 1.958 | 3.729 | 4.827 | 5.901 |
| Vem1 | 0.008 | 0.009 | 0.011 | 0.012 |
| Vem2 | 0.013 | 0.017 | 0.02 | 0.023 |

Questo metodo è molto consistente riguardo sia i tempi di esecuzione che l’errore, avendo infatti cambiamenti minimi nei tempi di esecuzione per le quattro matrici con solo un leggero aumento nei tempi di Spa2. Anche in questo caso però si può notare una pendenza leggermente maggiore per le due matrici Spa1 e Spa2 rispetto alle altre due che però non impatta pesantemente i tempi. Gli errori rimangono simili per le tolleranze più basse, con un aumento della differenza tra le due coppie di matrici quando la tolleranza aumenta.

**OSSERVAZIONI E ANALISI FINALE**

**Numero di iterazioni**

Come prima analisi confrontiamo il numero di iterazioni impiegate per la risoluzione del sistema tramite i vari metodi. Analizzando i grafici delle iterazioni possiamo notare come i due metodi iterativi stazionari risultino essere più prestanti sulle matrici più dense (spa1 e spa2) mentre per i metodi iterativi non stazionari le migliori prestazioni, a livello di iterazioni, le si notano sulle due matrici meno dense (vem1 e vem2).

Confrontando i due metodi stazionari notiamo che per le matrici più grandi Gauß-Seidel risulti eseguire molte meno iterazioni che Jacobi, questo perché durante l’esecuzione del codice di Gauß-Seidel vengono utilizzati subito i risultati appena calcolati senza dover poi aspettare la successiva iterazione come per Jacobi.

I due metodi iterativi non stazionari invece, differiscono in maniera sostanziale per il numero di iterazioni, infatti il metodo del gradiente coniugato è un miglioramento del metodo del gradiente, il miglioramento consiste nell’eliminazione del comportamento a “zig-zag” scegliendo le direzioni di discesa A-ortogonali consentendo di calcolare la soluzione in un numero di passi <= n.

**Errore assoluto e relativo**

In generale i metodi stazionari ottengono un errore che risulta essere minore sulle matrici spa1 e spa2 (quelle più dense), mente per i due metodi iterativi non stazionari l’errore risulta calare per le matrici vem1 e vem2 (meno dense).

Tra Gauß-Seidel e Jacobi gli errori sono molto simili (possono quasi essere considerati uguali) anche se, osservando attentamente i due grafici possiamo notare dei leggeri vantaggi sull’errore da parte di entrambi i metodi (l’uno con l’altro) dati dal tipo di matrice utilizzata e dalla tolleranza impostata.

Tra Gradiente e Gradiente coniugato invece la differenza di errore è visibilmente riconoscibile dai grafici; infatti, il secondo metodo ottiene degli errori nettamente inferiori rispetto al metodo del gradiente.

**Tempo impiegato**

I due metodi non stazionari risultano avere tempi nettamente minori di quelli stazionari per quanto riguarda le matrici meno dense. Inoltre, tra i metodi stazionari Gauß-Seidel è tendenzialmente più veloce del metodo di Jacobi, dato che utilizza immediatamente risultati già calcolati, senza dover aspettare l’iterazione successiva per poterli usare.   
Nei metodi non stazionari il metodo del Gradiente coniugato è più veloce del metodo del Gradiente, con tempi inferiori al mezzo secondo in tutti i casi, tranne per Spa2, che ha creato rallentamenti anche nel metodo del Gradiente.

**Conclusioni**

Possiamo concludere che è opportuno utilizzare metodi diversi per bisogni diversi, scegliendo il metodo e la relativa tolleranza a seconda del sistema lineare da risolvere e il tempo a disposizione.

Nonostante ciò, abbiamo osservato che il metodo del Gradiente coniugato ha tempi di esecuzione molto bassi e errori abbastanza simili tra le varie tipologie di matrici; di conseguenza è possibile utilizzare questo metodo quando si è incerti su quale utilizzare e impostare la tolleranza ad un numero molto basso, dato che i tempi sono comunque molto contenuti.