**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

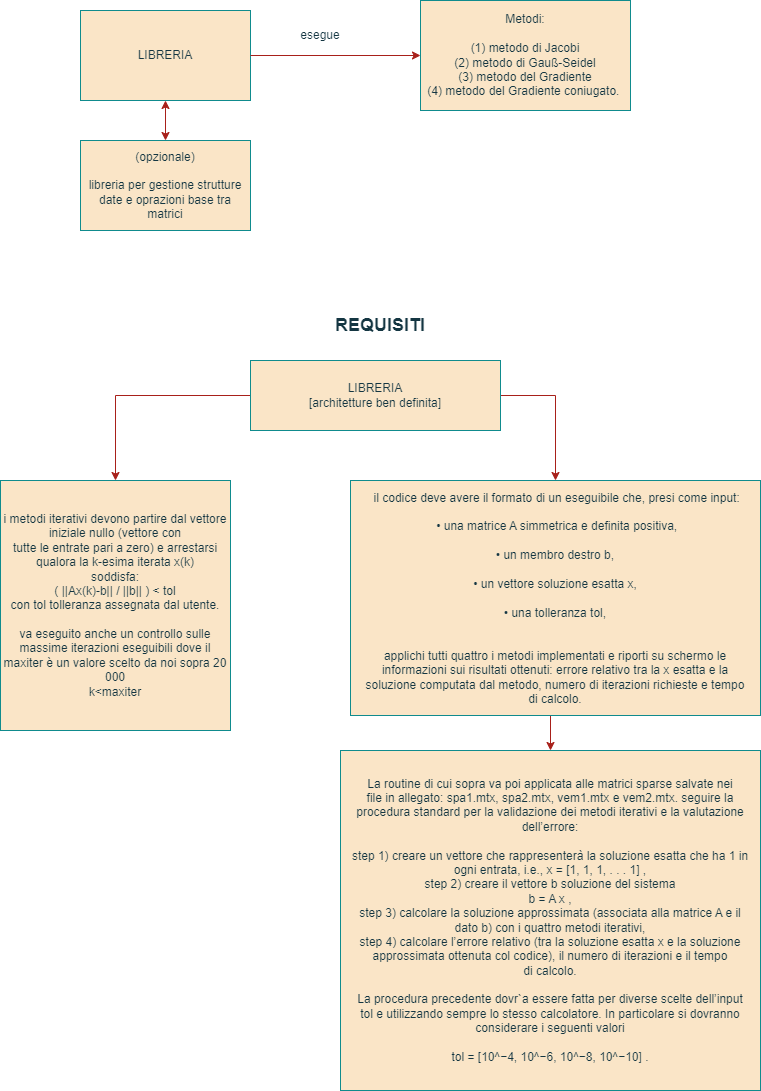
La richiesta di questo progetto prevedeva la creazione di una libreria con un linguaggio a nostra discrezione (nel nostro caso la scelta è ricaduta su Java) che riuscisse ad eseguire i 4 metodi iterativi qui menzionati limitandosi a matrici simmetriche e definite positive:

* 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 |

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 | 0.117 | 0.14 | 0.239 | 0.262 |
| 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 |

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 |