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

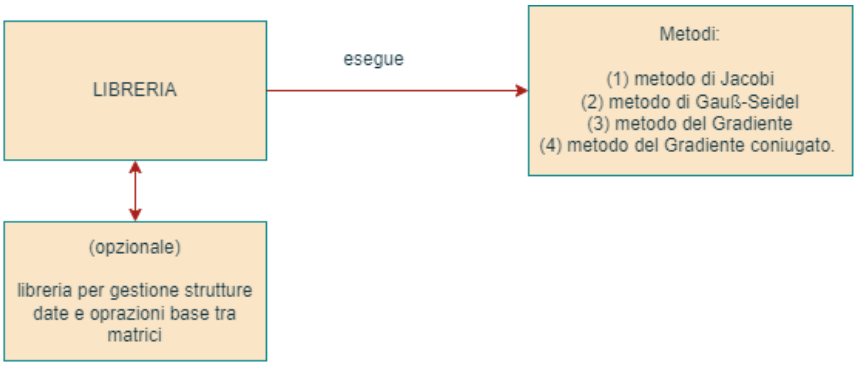
La richiesta di questo progetto prevedeva la creazione di una libreria con un linguaggio a nostra discrezione (nel nostro caso la scelta è ricaduta su Java) che riuscisse ad eseguire i 4 metodi iterativi qui menzionati limitandosi a matrici simmetriche e definite positive:

- Jacobi
- Gauß-Seidel
- Gradiente
- Gradiente coniugato

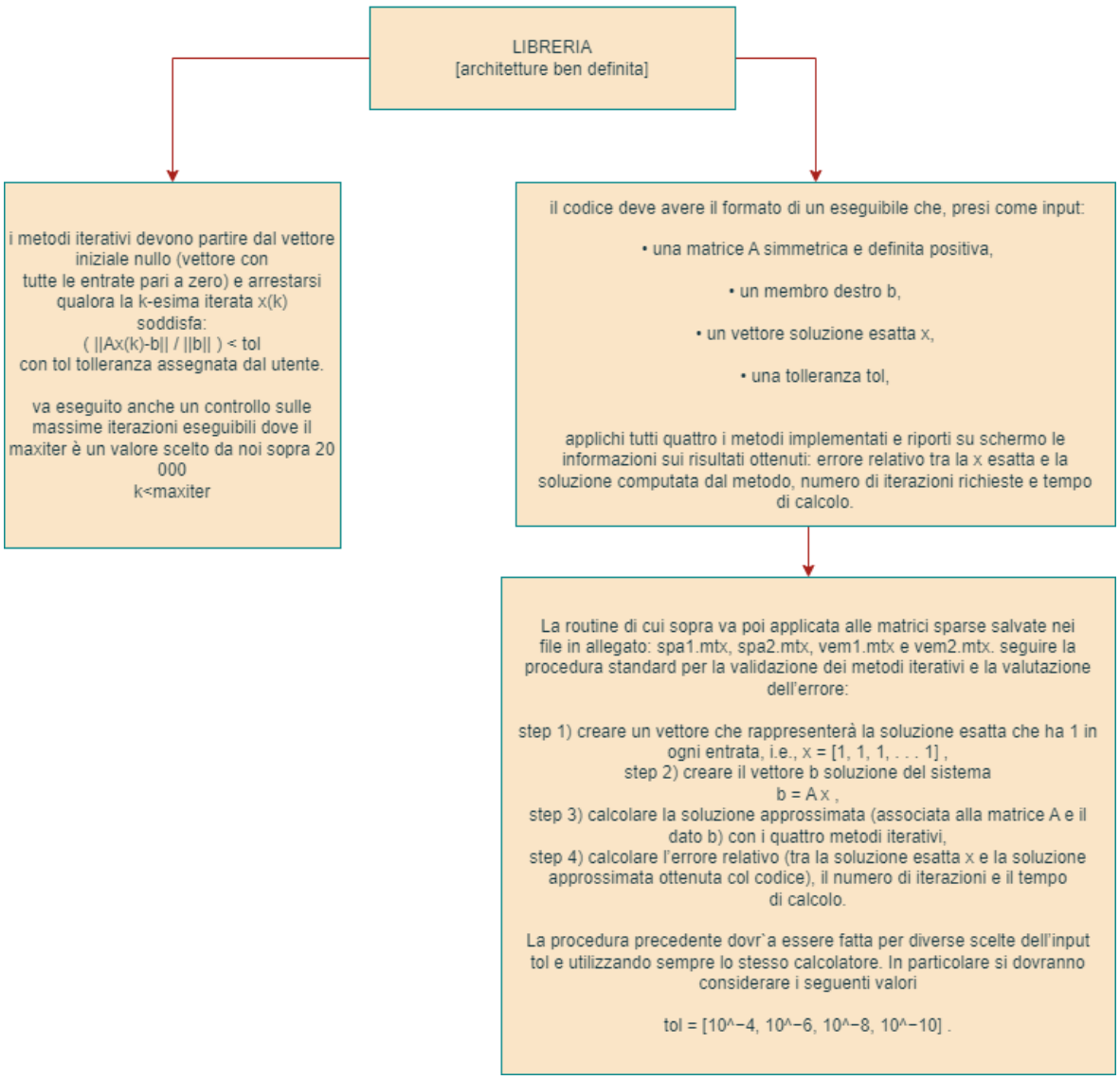
Per la gestione di: matrici, vettori e operazioni tra di essi ci siamo appoggiati a due librerie offerte da java, ovvero: Maths che è una delle librerie più utilizzate e la4j che per quanto riguarda le operazioni tra matrici e vettori risulta essere una delle più efficienti.

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.


LIBRERIA
esegue
Metodi:
(1) metodo di Jacobi
(2) metodo di Gauß-Seidel
(3) metodo del Gradiente
(4) metodo del Gradiente coniugato.
(opzionale)
libreria per gestione strutture date e oprazioni base tra matrici

REQUISITI


LIBRERIA
[architetture ben definita]
i metodi iterativi devono partire dal vettore iniziale nullo (vettore con tutte le entrate pari a zero) e arrestarsi qualora la k-esima iterata x(k) soddisfa:
(||Ax(k)-b|| / ||b||) < tol
con tol tolleranza assegnata dal utente.
va eseguito anche un controllo sulle massime iterazioni eseguibili dove il maxiter è un valore scelto da noi sopra 20 000
k<maxiter
il codice deve avere il formato di un eseguibile che, presi come input:
• una matrice A simmetrica e definita positiva,
• un membro destro b,
• un vettore soluzione esatta x,
• una tolleranza tol,
applichi tutti quattro i metodi implementati e riporti su schermo le informazioni sui risultati ottenuti: errore relativo tra la x esatta e la soluzione computata dal metodo, numero di iterazioni richieste e tempo di calcolo.
La routine di cui sopra va poi applicata alle matrici sparse salvate nei file in allegato: spa1.mtx, spa2.mtx, vem1.mtx e vem2.mtx. seguire la procedura standard per la validazione dei metodi iterativi e la valutazione dell'errore:
step 1) creare un vettore che rappresenterà la soluzione esatta che ha 1 in ogni entrata, i.e., x = [1, 1, 1, . . . 1] ,
step 2) creare il vettore b soluzione del sistema
b = A x ,
step 3) calcolare la soluzione approssimata (associata alla matrice A e il dato b) con i quattro metodi iterativi,
step 4) calcolare l'errore relativo (tra la soluzione esatta x e la soluzione approssimata ottenuta col codice), il numero di iterazioni e il tempo di calcolo.
La procedura precedente dovr`a essere fatta per diverse scelte dell'input tol e utilizzando sempre lo stesso calcolatore. In particolare si dovranno considerare i seguenti valori
tol = [10^−4, 10^−6, 10^−8, 10^−10] .

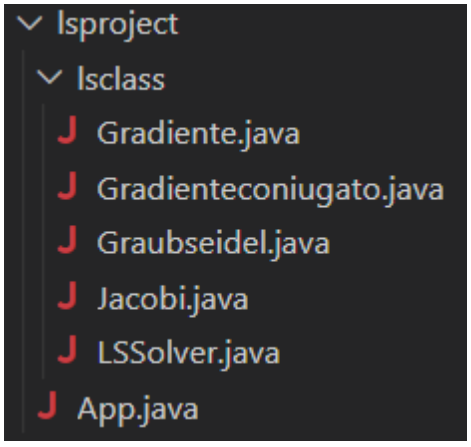
ARCHITETTURA

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.



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):

$$\frac{\|Ax^{(k)} - b\|}{\|b\|} < tolleranza$$

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).

CODICE

LSSolver:

```
public LSSolver(Matrix aIn, Vector bIn, Vector xIn) {
    a = aIn;
    b = bIn;
    sol = xIn;
    diag = new BasicVector(new double[b.length()]);
    for (int i = 0; i < a.columns(); i++) {
        diag.set(i, value: a.get(i, j: i));
    }
}

public LSSolver(String str) throws IOException {
    a = ImportMtxFile(fileName: str);
    setMatrix();
}
```

```
public void executeMethods(double tol) {
    startTime = System.currentTimeMillis();
    Vector xVecchio = new BasicVector(new double[sol.length()]);
    for (int i = 0; i < xVecchio.length(); i++) {
        xVecchio.set(i, value: 0);
    }
    double delta = 0;
    int k = 0;
    do {
        // calcolo soluzione
        solutionX = risoluzione(xVecchio);
        // verifica convergenza
        delta = norma2(v: a.multiply(that: solutionX).subtract(that: b)) / norma2(v: b);
        xVecchio = solutionX;
        if (k > maxIter) {
            System.out.println(x: "Errore: Raggiunto il numero massimo di iterazioni");
            break;
        }
        k++;
    } while (delta > tol && norma2(v: sol.subtract(that: solutionX)) > tol);
    endTime = System.currentTimeMillis();
    System.out.println("esecuzione metodo con tolleranza a: " + tol + " in " + k
            + " iterazioni con un tempo di: " + (endTime - startTime) + " millisecondi ("
            + ((double) endTime - (double) startTime) / 1000 + " sec)");
    System.out.println(x: "Errore relativo:");
    System.out.println(x: String.format(format: "%.20f", (norma2(v: sol.subtract(that: solutionX)) /
            norma2(v: sol))));
    // System.out.println("soluzione: " + solutionX);
}
```

Jacobi:

```
public class Jacobi extends LSSolver {

    public Jacobi(String str) throws IOException {
        super(str);
    }

    public Jacobi(Matrix aIn, Vector bIn, Vector xIn) {
        super(aIn, bIn, xIn);
    }

    public void reset() {

    }

    public Vector risoluzione(Vector xVecchio) {
        Vector solutionX = new BasicVector(new double[sol.length()]);
        for (int i = 0; i < sol.length(); i++) {
            double sum = b.get(i);
            for (int j = 0; j < a.columns(); j++) {
                if (j != i) {
                    sum -= a.get(i, j) * xVecchio.get(i: j);
                }
            }
            solutionX.set(i, sum / diag.get(i));
        }
        return solutionX;
    }
}
```

Gauß-Seidel:

```
public class Graubseidel extends LSSolver {

    public Graubseidel(String str) throws IOException {
        super(str);
    }

    public Graubseidel(Matrix aIn, Vector bIn, Vector xIn) {
        super(aIn, bIn, xIn);
    }

    public void reset() {

    }

    public Vector risoluzione(Vector xVecchio) {
        Vector solutionX = new BasicVector(new double[sol.length()]);
        for (int i = 0; i < sol.length(); i++) {
            double sum = b.get(i);
            for (int j = 0; j < a.columns(); j++) {
                if (j > i) {
                    sum -= a.get(i, j) * xVecchio.get(i: j);
                }
                if (j < i) {
                    sum -= a.get(i, j) * solutionX.get(i: j);
                }
            }
            solutionX.set(i, sum / diag.get(i));
        }
        return solutionX;
    }
}
```

Gradiente:

```
public class Gradiente extends LSSolver {

    public Gradiente(String str) throws IOException {
        super(str);
    }

    public Gradiente(Matrix aIn, Vector bIn, Vector xIn) {
        super(aIn, bIn, xIn);
    }

    public void reset() {

    }

    public Vector risoluzione(Vector xVecchio) {
        Vector residuo = b.subtract(that: a.multiply(that: xVecchio));
        Vector y = a.multiply(that: residuo);
        Double va = prodottoScalare(v1: residuo, v2: residuo);
        Double vb = prodottoScalare(v1: residuo, v2: y);
        Double alpha = va / vb;
        solutionX = xVecchio.add(that: residuo.multiply(value: alpha));
        return solutionX;
    }
}
```

Gradiente coniugato:

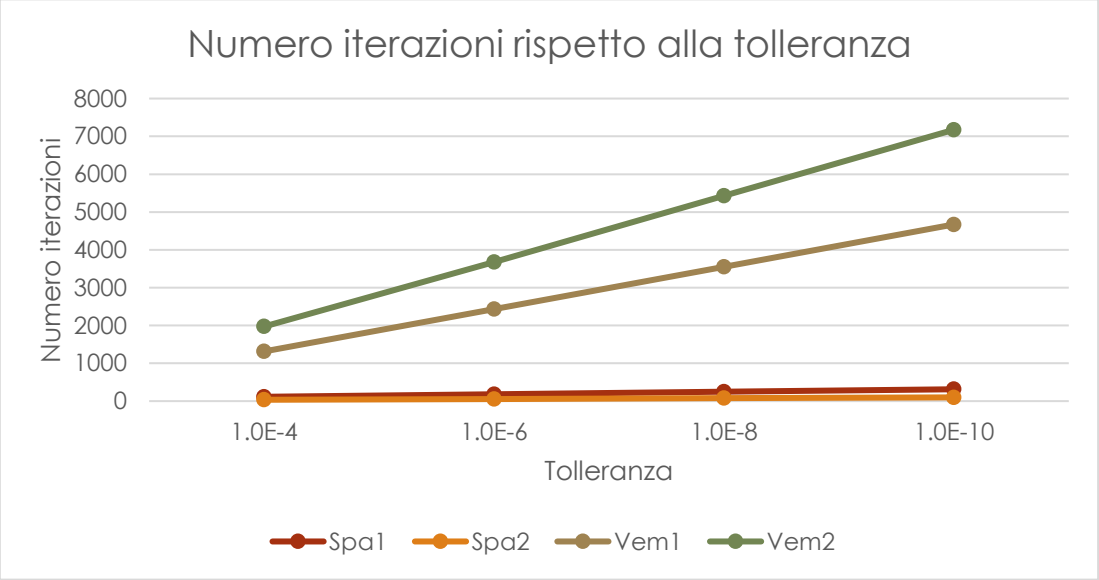
```
public class Gradienteconiugato extends LSSolver {
    Vector residuo;
    Vector p;

    public Gradienteconiugato(String str) throws IOException {
        super(str);
        residuo = b.subtract(that: a.multiply(new BasicVector(new double[sol.length()])));
        p = residuo;
    }

    public Gradienteconiugato(Matrix aIn, Vector bIn, Vector xIn) {
        super(aIn, bIn, xIn);
        residuo = b.subtract(that: a.multiply(new BasicVector(new double[b.length()])));
        p = residuo;
    }

    public void reset() {
        residuo = b.subtract(that: a.multiply(new BasicVector(new double[sol.length()])));
        p = residuo;
    }

    public Vector risoluzione(Vector xVecchio) {
        Double alpha = prodottoScalare(v1: p, v2: residuo) / prodottoScalare(v1: p, v2: a.multiply(that: p));
        solutionX = xVecchio.add(that: p.multiply(value: alpha));
        residuo = b.subtract(that: a.multiply(that: solutionX));
        Double betha = prodottoScalare(v1: p, v2: a.multiply(that: residuo)) / prodottoScalare(v1: p, v2: a.multiply(that: p));
        p = residuo.subtract(that: p.multiply(value: betha));
        return solutionX;
    }
}
```

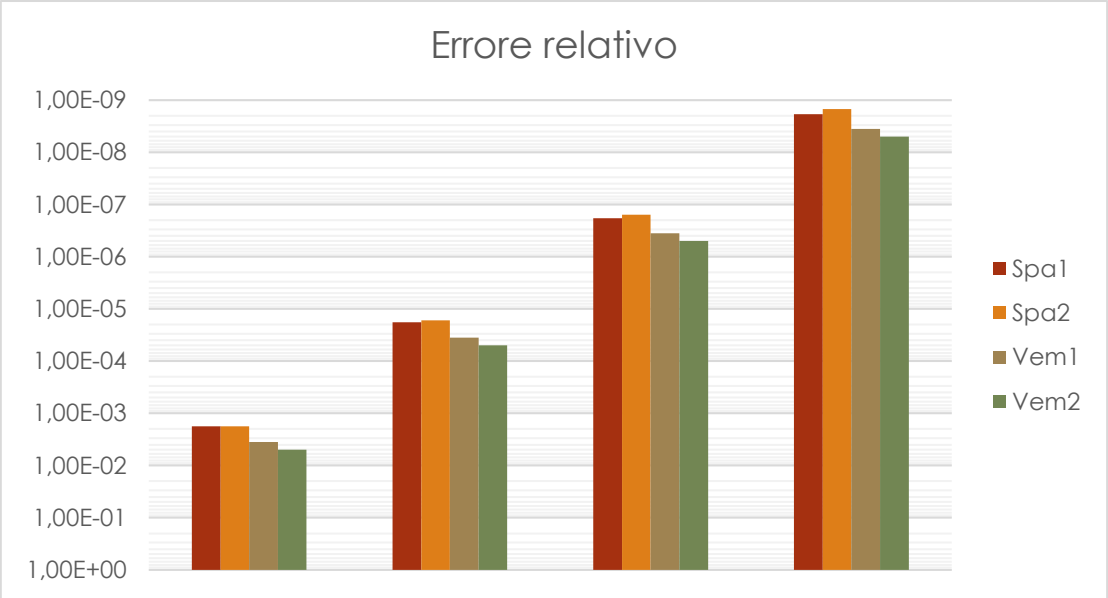
JACOBI

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

$$x_i^{(k+1)} = \frac{b_i - \sum_{j \neq i} a_{ij} x_j^{(k)}}{a_{ii}}$$

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:

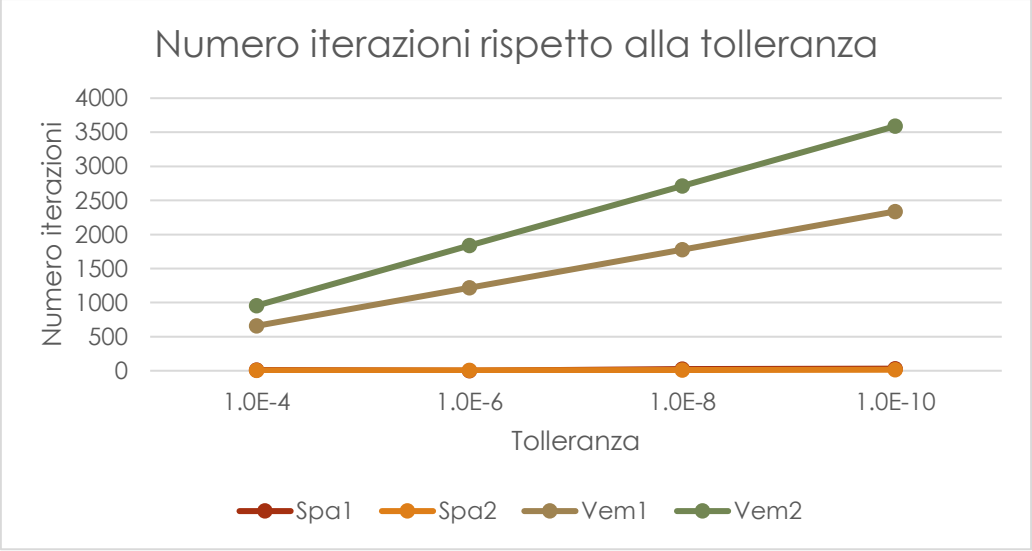
	1.0E-4	*1.0E-6*	*1.0E-8*	*1.0E-10*
Spa1	3.753	7.809	8.017	10.373
Spa2	15.041	24.492	33.363	42.214
Vem1	15.074	28.229	40.461	51.115
Vem2	49.792	96.237	141.098	185.05

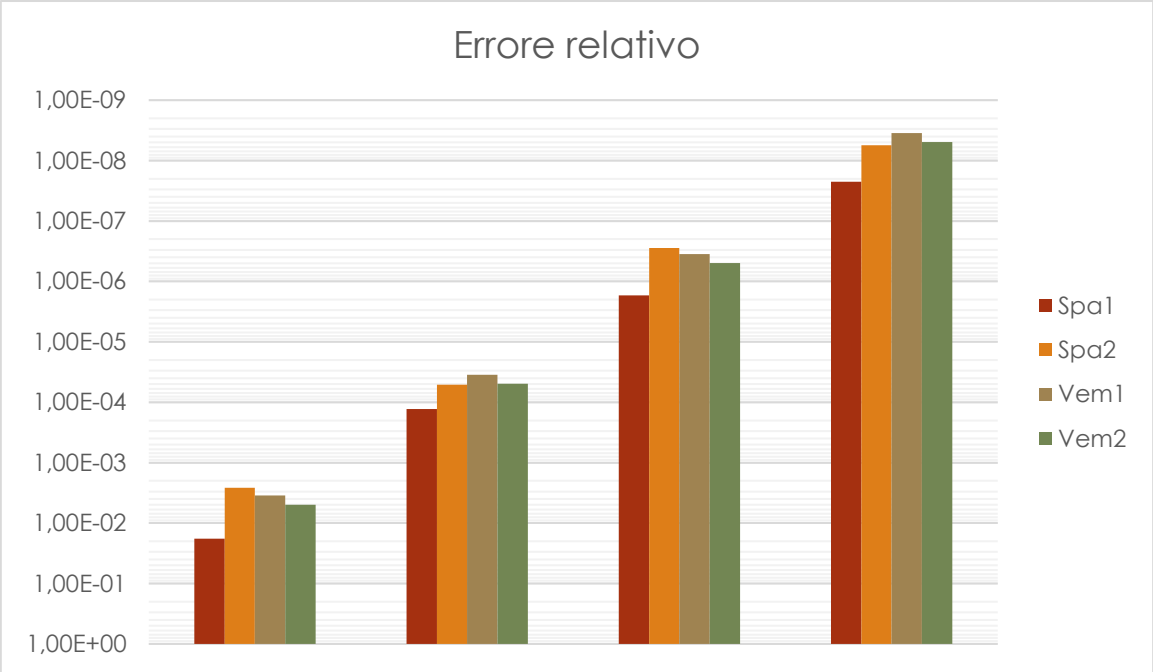
Come si può notare dai grafici questo metodo performa in modo simile per quanto riguarda l'errore con tutte le matrici, ma ha un numero di iterazioni e dei tempi di esecuzione che crescono più velocemente nelle matrici meno dense quali Vem1 e Vem2.

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:

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





	1.0E-4	*1.0E-6*	*1.0E-8*	*1.0E-10*
Spa1	0.317	0.557	0.76	0.973
Spa2	2.101	3.182	4.886	5.928
Vem1	7.528	13.952	20.245	27.225
Vem2	26.91	49.974	72.663	95.915

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 anche se l'errore rimane simile su tutte le matrici; infatti, come si vede dal grafo delle iterazioni per le matrici più dense 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:

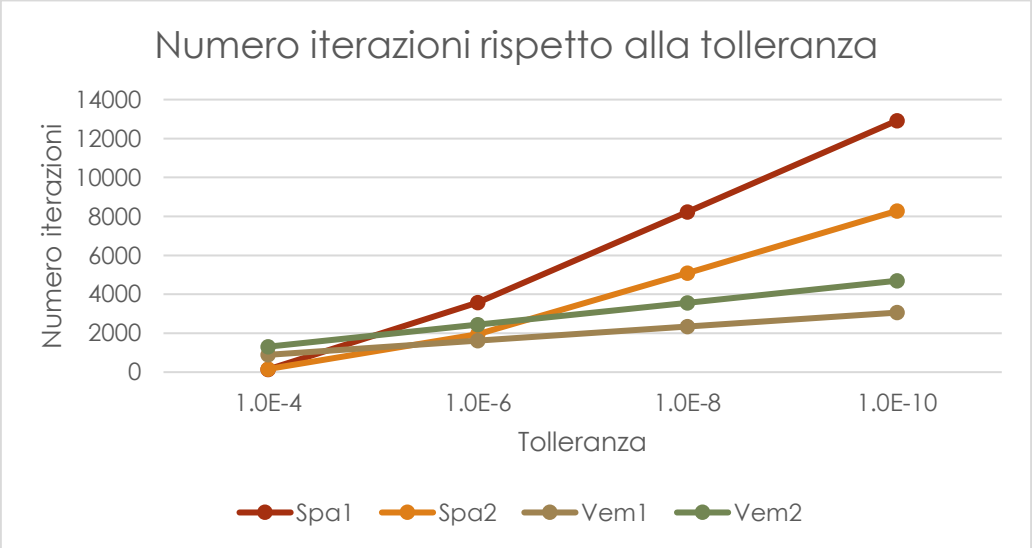
$$r^{(k)} = b - Ax^{(k)}$$

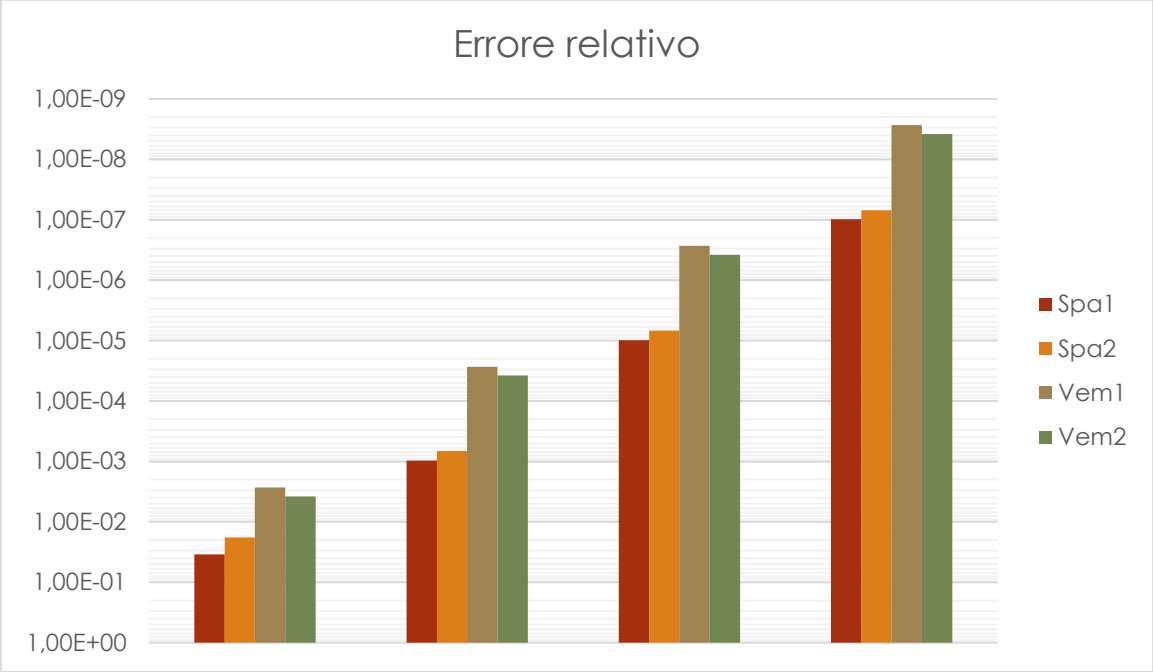
E per andare ad aggiornare la variabile x(k), otterremo la seguente equazione:

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

Dove:

$$\alpha^{(k)} = \frac{(r^{(k)})^t r^{(k)}}{(r^{(k)})^t A r^{(k)}}$$





	1.0E-4	*1.0E-6*	*1.0E-8*	*1.0E-10*
Spa1	0.298	6.586	13.435	20.306
Spa2	2.407	29.046	76.322	125.418
Vem1	0.128	0.217	0.314	0.413
Vem2	0.292	0.526	0.769	1.029

Il gradiente a differenza dei metodi precedentemente visti è presente una differenza significativa dell'errore relativo delle matrici più dense e quelle meno dense, con quest'ultime avente un errore minore. Inoltre, i tempi di esecuzione e la crescita delle iterazioni sono inferiori per le matrici Vem1 e Vem2 diversamente da jacobi e gauss-seidell, che agivano in maniera opposta.

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:

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

dove $\alpha^{(k)}$ è:

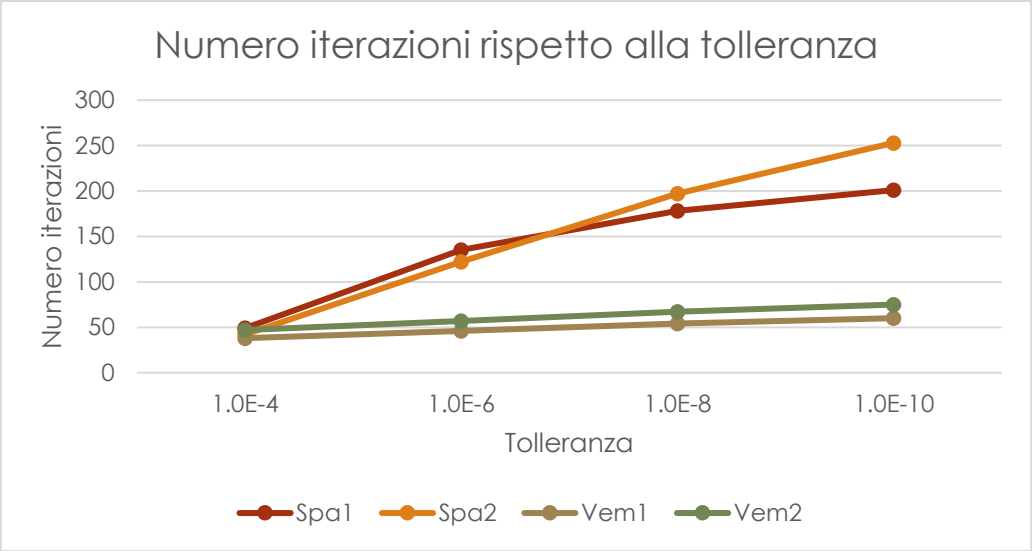
$$\alpha^{(k)} = \frac{(P^{(k)})^T r^{(k)}}{(P^{(k)})^T A P^{(k)}}$$

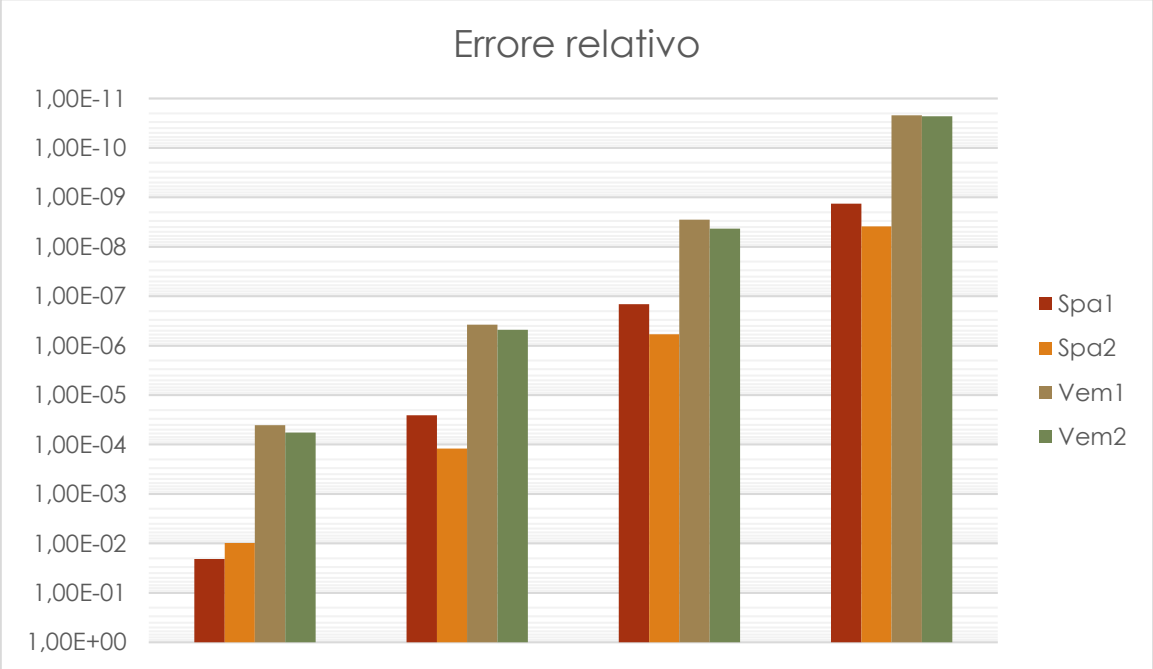
Nel passo successivo calcoliamo:

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

Dove:

$$\beta^{(k)} = \frac{(P^{(k)})^T A r^{(k+1)}}{(P^{(k)})^T A P^{(k)}}$$





	1.0E-4	*1.0E-6*	*1.0E-8*	*1.0E-10*
Spa1	0.169	0.459	0.551	0.608
Spa2	1.16	3.351	5.424	6.995
Vem1	0.012	0.013	0.016	0.015
Vem2	0.019	0.021	0.029	0.03

Questo metodo si comporta in modo simile a quello del gradiente, in quanto riporta una differenza a favore delle matrici meno dense.

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 Gauß-Seidel risulti eseguire molte meno iterazioni rispetto a Jacobi, questo perché durante l'esecuzione del codice di Gauß-Seidel vengono utilizzati subito i risultati appena calcolati senza dover 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; questo porta ad una diminuzione sostanziale del numero di iterazioni.

Errore relativo

In generale i metodi non stazionari ottengono un errore che risulta essere minore sulle matrici Vem1 e Vem2 (quelle meno dense), mente per i due metodi iterativi stazionari l'errore risulta essere simile per tutte le matrici.

Tra Gauß-Seidel e Jacobi gli errori sono molto simili (anche se c'è un leggero miglioramento si Spa1 e Spa2 per jacobi)

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

Tempo impiegato

I due metodi non stazionari risultano avere tempi nettamente inferiori a 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 invece 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 ed errori contenuti; 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, per avere un risultato comunque accettabile.