AA 2022-2023 - Metodi del Calcolo Scientifico - Progetto 1

Algebra lineare numerica
Sistemi lineari con matrici sparse simmetriche e definite positive

Introduzione. Lo scopo di questo progetto è di studiare l'implementazione in ambienti di programmazione open source del metodo di Choleski per la risoluzione sistemi lineari per matrici sparse, simmetriche e definite positive, e di confrontarli con l'implementazione di MATLAB. Comprende la implementazione di un codice e la scrittura di una relazione da consegnare ai docenti.

Immaginate che la vostra azienda abbia la necessità di munirsi di un ambiente di programmazione per **risolvere con il metodo di Choleski sistemi lineari con matrici sparse e definite positive di grandi dimensioni**. L'alternativa è tra **software proprietario (MATLAB)** oppure **open source** e anche tra **Windows** oppure **Linux.** Vi si richiede quindi di confrontare in ambiente Linux e Windows, e sulla stessa macchina, MATLAB e una libreria open source (non Octave) a vostra scelta. Il confronto deve avvenire in termini di tempo, accuratezza, impiego della memoria e anche facilità d'uso e documentazione. In altre parole, è meglio affidarsi alla sicurezza di MATLAB pagando oppure vale la pena di avventurarsi nel mondo open source? Ed è meglio lavorare in ambiente Linux oppure in ambiente Windows?

Matrici sparse. Le matrici sparse sono matrici con un grande numero di elementi uguali a zero. Spesso il numero di elementi diversi di zero su ogni riga è un numero piccolo (per esempio dell'ordine di 10^1) indipendente dalla dimensione della matrice, che può essere anche dell'ordine di 10^8. Molti problemi del calcolo scientifico si riconducono alla soluzione di uno o più sistemi lineari con matrice dei coefficienti sparsa. Per esempio, i motori di ricerca del web portano naturalmente a matrici grandissime estremamente sparse.

Le matrici sparse si possono memorizzare in modo compatto, tenendo solo conto degli elementi diversi da zero; per esempio, è sufficiente per ogni elemento diverso da zero memorizzare solo la sua posizione (i, j) e il suo valore a_{ij}, e semplicemente ignorare gli elementi uguali a zero.

Risoluzione di sistemi lineari con matrici sparse. Le considerazioni seguenti fatte per il metodo di eliminazione di Gauss valgono anche per il metodo di Choleski, applicabile ovviamente solo a matrici simmetriche e definite positive.

Il semplice metodo di eliminazione di Gauss (con pivot o senza pivot) applicato alle matrici sparse tende a generare nuovi elementi diversi da zero, causando il riempimento (*fill-in*) della matrice triangolare finale (oppure, equivalentemente, delle matrici L ed U).

Il *fill-in* rende il metodo di eliminazione di Gauss inutilizzabile, perché saremmo costretti ad allocare lo spazio sufficiente per rappresentare l'*intera* matrice.

Ci sono tuttavia matrici sparse particolari per le quali l'algoritmo di Gauss non genera fill-in: per esempio le matrici *tridiagonali*, nelle quali gli elementi diversi da zero sono solo sulla diagonale principale e sulle due sottodiagonali.

L'idea per trattare matrici sparse generali è di fare una *permutazione preliminare di righe e colonne* in modo che l'algoritmo di Gauss generi il minor numero possibile di elementi diversi da zero.

Ci sono diverse tecniche per effettuare questa permutazione preliminare; una eccellente introduzione all'argomento è il libro Direct Methods for Sparse Linear Systems (SIAM 2006), di Timothy A. Davis (se vi interessa posso procurarvene una copia).

Per tutti i dettagli si veda la sua pagina web:
http://faculty.cse.tamu.edu/davis/background.html.
Le matrici simmetriche e definite positive che dovrete considerare per la relazione fanno parte della **SuiteSparse Matrix Collection** che colleziona matrici sparse derivanti da applicazioni di problemi reali (ingegneria strutturale, fluidodinamica, elettromagnetismo, termodinamica, computer graphics/vision, network e grafi). Trovate tutte le matrici collegandovi alla pagina https://sparse.tamu.edu/.
Consultate anche la guida https://sparse.tamu.edu/about.

In particolare le matrici simmetriche e definite positive che analizzerete sono le seguenti:

- Flan_1565
- StocF-1465
- cfd2
- cfd1
- G3_circuit
- parabolic_fem
- apache2
- shallow_water1
- ex15

Osservate che il trattamento per matrici simmetriche e definite positive e matrici generiche può essere molto diverso. Si veda per esempio la seguente documentazione MATLAB
https://it.mathworks.com/help/matlab/ref/mldivide.html.

Progetto. Lo scopo del progetto è confrontare il solutore per matrici sparse simmetriche e definite positive di MATLAB con quello di (almeno una) libreria open-source a vostra scelta. Il confronto va fatto **sulla stessa macchina e sulle due architetture Windows e Linux.**

ATTENZIONE: MATLAB si accorge da solo se la matrice che gli passate è simmetrica e definita positiva. Questo non succede (di solito) per le librerie open source: dovete stare attenti ad utilizzare algoritmi specifici per matrici simmetriche e definite positive. Se non lo fate, potreste ottenere dei risultati molto peggiori.

Analizzate tutte le matrici che riuscite, iniziando da quelle più piccole.

In ciascuno dei due ambienti di programmazione, utilizzate un solutore diretto per matrici simmetriche e definite positive sparse risolvendo il sistema lineare `Ax=b` dove il termine noto `b` è scelto in modo che la soluzione esatta sia il vettore `xe=[1 1 1 1 1 1 ...]`, avente tutte le componenti uguali a 1: in altre parole, `b=A*xe`.

Per ognuna delle matrici occorre determinare:

- il tempo necessario per calcolare la soluzione `x`;

- l'errore relativo tra la soluzione calcolata `x` e la soluzione esatta `xe`, definito da:

$$\texttt{errore relativo} = \frac{\|\texttt{x} - \texttt{xe}\|_2}{\|\texttt{xe}\|_2}$$

  dove $\|\texttt{v}\|_2$ è la norma euclidea del vettore `v`;

- la memoria necessaria per risolvere il sistema, ovvero grosso modo l'aumento della dimensione del programma in memoria da subito dopo aver letto la matrice a dopo aver risolto il sistema.

Riportate in un grafico queste tre quantità mettendo in ascissa la dimensione della matrice e in ordinata tempo, errore e memoria. Sarà opportuno che la scala delle ordinate sia logaritmica, dato che i valori da rappresentare saranno di ordini di grandezza diversi.

Riassunto

Riassumendo, la relazione dovrà contenere:

- Per ogni ambiente di programmazione (MATLAB + open source diversa da Octave a vostra scelta) una breve descrizione della libreria usata mettendo in evidenza le sue caratteristiche e se è documentata e mantenuta;
- Per ogni ambiente di programmazione (MATLAB e open source), per ogni sistema operativo (Linux e Windows) e per ogni parametro (velocità, precisione e occupazione di memoria), eseguire gli algoritmi sulla stessa macchina e riportare i grafici come descritto sopra;
- Grafici riassuntivi che permettano il confronto tra le varie situazioni e consentano di prendere una decisione operativa;
- Il listato del codice.

Qualche nota logistica:

- All'esame "portate" un computer in modo che possiamo far girare il vostro programma.
- La relazione dovrà essere consegnata almeno 3 giorni (= 72 ore) prima dell'esame.
- Non cambiate i componenti del gruppo tra il primo e il secondo progetto (tutto lo stesso gruppo deve affrontare entrambi i progetti e partecipare all'orale nella stessa data).

Se avete dei dubbi mandate pure una email.