Michele Iannello

Relazione per l’esame di Elementi di Programmazione per la Fisica

Il codice che ho elaborato, e che viene illustrato dalla presente relazione, si propone di calcolare la matrice inversa di una matrice quadrata; per raggiungere questo risultato, ho utilizzato il metodo di Gauss, appreso durante le lezioni di Algebra Lineare.

Il metodo di Gauss

Il metodo di Gauss viene applicato in Algebra, tra le altre cose, per determinare l’inversa di una matrice quadrata; in questa sezione lo illustro brevemente.
Per comodità di trattazione, sia A la matrice data e sia I la matrice identità (la matrice, di dimensione arbitraria, i cui elementi sulla diagonale sono tutti uguali a 1, mentre i restanti elementi sono tutti uguali a 0). Una matrice A è invertibile solamente se il suo determinante è diverso da 0; in questo caso, il prodotto *righe per colonne* di A per la sua inversa restituisce l’identità I.
Il metodo prevede innanzitutto di impostare una matrice rettangolare scrivendo I accanto ad A, ottenendo così una matrice rettangolare con un numero di colonne doppio rispetto al numero delle righe.
Prima di procedere, occorre controllare il primo numero della prima riga di A. Nel caso in cui sia 0, è necessario scambiare la prima riga con un’altra riga qualsiasi, a patto che il primo elemento di quest’ultima sia diverso da 0. Nel caso in cui il primo elemento di ogni riga sia uguale a 0, un *Teorema* ci garantisce che il determinante della matrice è uguale a 0, per cui l’inversa non esiste e non si procede con il metodo.
In caso contrario, si possono eseguire sulla matrice rettangolare le cosiddette operazioni elementari di riga:

- scambio di due righe;
- moltiplicazione di una riga per un numero reale diverso da 0;
- sostituzione della riga j-esima con la somma della riga i-esima e della j-esima moltiplicata per un numero reale qualsiasi.

L’obiettivo è ottenere la matrice identità laddove originariamente è stata scritta A, poiché a questa condizione il metodo garantisce che, laddove era stata scritta I, si ottiene l’inversa di A.
Nel caso in cui, applicando il metodo, si giunga alla situazione in cui almeno una riga si annulla (è composta di soli zeri), il *Teorema* citato precedentemente garantisce che il determinante della matrice è nullo, e dunque la sua matrice inversa non esiste.

Il codice

Il codice che ho elaborato consente di calcolare l’inversa di una matrice quadrata di dimensione arbitraria (per ragionevolezza ho fissato il massimo a 10) il cui determinante sia diverso da 0.
Per prima cosa ho creato una classe che ho chiamato “Matrice”, utilizzando la parola di vocabolario *struct*.
All’interno della classe ho dichiarato un puntatore a puntatori a *double* e ho definito due metodi *void* e un costruttore (di quest’ultimo disquisirò più avanti).
Ogni puntatore a *double* serve a puntare i valori, inseriti da tastiera, di una riga della matrice.
Il primo metodo (*inserita*) si serve di due cicli *for*, uno dentro l’altro, per stampare sul terminale la matrice inserita; il secondo metodo (*inversa*) utilizza la stessa procedura per stampare sul terminale, in caso di successo, la matrice inversa. Entrambi i metodi fanno uso di due manipolatori di output stream: *setprecision()* e *setw()*, che ho utilizzato per incolonnare ordinatamente i valori e rendere leggibili le matrici.
Nell’ambito globale ho definito tre funzioni *void* volte ad eseguire le operazioni di riga, anch’esse dai nomi parlanti: *scambia*, *dividi* e *sottrai*. Le prime due ricevono 3 argomenti, la terza ne riceve 4.
Il terzo argomento di tutte e tre le funzioni, di tipo *int*, è il numero di righe della matrice (e dunque la metà del numero di colonne della matrice rettangolare composta da matrice inserita e matrice identità).

- s*cambia* riceve due puntatori a *double* e un *int*, e come ci si potrebbe aspettare si occupa di scambiare i due puntatori (che rappresentano le due righe), ovvero di assegnare al primo il valore del secondo e viceversa (per cui i due puntatori devono essere ricevuti per riferimento);
- *dividi* riceve un puntatore a *double* (anch'esso per riferimento), un *double* e un *int*, e sostituisce ogni elemento del puntatore con il risultato del rapporto tra questo e il *double*;
- *sottrai* riceve due puntatori a *double* (solo il secondo per riferimento), un *int* e un *double*, e sostituisce ogni elemento i-esimo del secondo puntatore con la differenza tra questo e il prodotto tra il *double* e l'elemento i-esimo del primo puntatore, per ogni i che va da 0 al doppio dell'*int* ricevuto (per maggiore chiarezza: detto **a** il primo puntatore, **b** il secondo, **n** l'*int* e **k** il *double*, la funzione sostituisce **b[i]** con **b[i] – k*a[i]** per ogni **i** compresa tra 0 e **2n**).

A questo punto procedo ad illustrare l'ambito della funzione *main*.
Il programma, dopo essersi presentato con una stringa di output, entra immediatamente in un ciclo *do* all'interno del quale si trova la quasi totalità delle istruzioni contenute in *main*, allo scopo di permettere molteplici calcoli di matrici inverse all'interno della stessa esecuzione del codice.
Tramite output viene ora richiesto all'esecutore di decidere il numero di righe della matrice (e di colonne, poiché si parla di matrici quadrate) e di inserirlo per mezzo della tastiera, seguìto dal tasto *Invio*. La gestione dell'input accetta soltanto numeri naturali compresi tra 1 e 10, e alla digitazione di un qualsiasi altro carattere, o di due numeri separati da spazi, richiede nuovamente l'inserimento. Questo risultato è raggiunto grazie a un segmento di codice che utilizza:

- un ciclo *do/while* per permettere un nuovo input, quando quello digitato non rispondesse alle richieste di cui sopra;
- due istruzioni di controllo *if*, che rilevano rispettivamente il fallimento nella lettura dell'input e la presenza di spazi o caratteri dopo il numero letto in assenza di errore. Al termine dell'ambito di ognuna delle due si trova la parola di vocabolario *continue*, in modo che l'intero segmento di codice che separa l'istruzione *if* dalla fine dell'ambito di *do* venga ignorato e il programma prosegua con l'iterazione successiva;
- un segmento di codice, volto alla gestione delle eccezioni, che utilizza le parole di vocabolario *try*, *throw* e *catch* per ben tre tipi diversi di digitazione: numeri non positivi, numeri maggiori di 10 e la digitazione del numero 1; nel caso in cui il programma entri nell'ambito di quest'ultimo *catch*, viene subito richiesta la digitazione dell'unico numero che costituisce la matrice, e nel caso in cui sia diverso da 0 ne viene stampato il reciproco sul terminale. Diversamente, l'esecutore viene invitato a una nuova digitazione e il ciclo viene trasferito all'iterazione successiva mediante la parola *continue.*
- un'ultima istruzione *if* che, nel caso in cui la lettura sia andata a buon fine, interrompe il ciclo tramite la parola *break.*

A questo punto si può procedere alla dichiarazione di un array di puntatori a *double* e contestualmente all'allocazione della memoria che sarà necessaria a contenere gli elementi della matrice rettangolare. Nello specifico, detto **n** il numero di righe della matrice, ho dichiarato un array che contiene **n** puntatori, e con un ciclo *range for* ho allocato ad ognuno di essi memoria pari a **2n*sizeof(double)** bytes, in modo da utilizzare le operazioni di riga per gestire contemporaneamente gli elementi della matrice di cui il programma si propone di calcolare l'inversa e gli elementi della matrice identità.
Il programma richiede ora l'inserimento dei singoli numeri che andranno a costituire la matrice **n*n,** assegnando i valori digitati ad ogni elemento di ogni puntatore per mezzo di due cicli *for.* Un sistema di gestione dell'input analogo a quello descritto poc'anzi garantisce l'inserimento di soli numeri reali, pena la necessità di dovere eseguire una nuova digitazione (e la comparsa di un commento stizzito sulla linea di output). Al termine dell'inserimento, il programma assegna automaticamente il valore corretto agli elementi puntati che devono costituire la matrice identità, utilizzando un ulteriore ciclo *for*.
Ora può essere invocato l'unico costruttore di Matrice, la cui presentazione ho in un primo momento posticipato per questioni di chiarezza. Questo riceve come argomento un puntatore a puntatore di *double* e inizializza con il suo valore la variabile membro (dello stesso tipo) dell'oggetto da creare.
Il programma prosegue mostrando in output la matrice inserita dall'esecutore, grazie all'invocazione del metodo *inserita* dell'oggetto appena creato.

Viene ora eseguito il nucleo del codice, ovvero il metodo di Gauss per ottenere la matrice inversa.
Il programma entra in un ambito *try* e controlla se il primo elemento della prima riga è uguale a 0; in caso affermativo, avvia un ciclo *for* fornito di una clausola *if*, che invoca la funzione *scambia* la prima volta che trova una riga il cui primo elemento non sia nullo. Se tutte le righe hanno il primo elemento nullo, viene lanciata da *throw* un'espressione raccolta da *catch*, il quale segnala all'esecutore che la matrice inversa non esiste e chiede di scegliere se procedere con il calcolo di un'altra matrice oppure terminare il programma. Ho utilizzato una gestione dell'input analoga alle precedenti, con l'aggiunta di una variabile di tipo *bool* che consente contemporaneamente l'uscita dal ciclo *do/while* che gestisce questi errori (tramite *break*) e il passaggio all'iterazione successiva del ciclo *do/while* che contiene la quasi totalità delle istruzioni di esecuzione (tramite *continue*).
Nel caso in cui invece l'operazione di scambio sia andata a buon fine, il programma può ottenere ora l'identità dalla matrice di partenza, entrando in due cicli *for*, uno dentro l'altro, e invocando le funzioni *dividi* e *sottrai* con il seguente criterio:

- ad ogni iterazione del ciclo esterno, ogni elemento della riga i-esima viene diviso per l'elemento i-esimo della stessa riga per mezzo della funzione *dividi:* in questo modo, ad ogni iterazione, lo stesso elemento i-esimo diverrà 1;
- ad ogni iterazione del ciclo interno, viene eseguita la funzione *sottrai* (come sopra descritta) solo se le due variabili che gestiscono i due cicli sono diverse tra loro: in questo modo, ad ogni iterazione, viene annullato l'elemento j-esimo di ogni riga, esclusa la riga j-esima.

Alla conclusione di questo segmento di codice, si aprono due possibilità: se almeno una delle righe della matrice quadrata è composta di soli zeri, il determinante della matrice è uguale a 0 e perciò la sua matrice inversa non esiste (si veda trattazione matematica); in caso contrario, è stata ottenuta una matrice identità. Per stabilirlo, ho utilizzato una gestione delle eccezioni simile alla precedente che, nel caso in cui tutti gli elementi di una riga siano nulli, chiede all'esecutore se procedere con il calcolo di una nuova matrice o terminare il programma. La consueta (e leggermente sarcastica) gestione degli eventuali errori di input si occupa di chiedere una nuova digitazione ogni qual volta questa non obbedisse alle richieste del codice.
Se non è stata trovata alcuna riga nulla viene ora mostrata sul canale di output la tanto attesa matrice inversa, per mezzo del metodo membro a lei dedicato e denominato proprio *inversa*.
Infine viene domandato all'esecutore se procedere con il calcolo di una nuova matrice oppure terminare il programma, operazione che richiede un'ultima gestione degli eventuali errori di input.
Nel caso in cui si scelga di terminare il programma, appare un confortante messaggio:
"Programma terminato senza errori".

Bibliografia e sitografia

R.Fioresi, M.Morigi, Introduzione all'Algebra Lineare, Edizioni Ambrosiana, Milano 2015
Dispense didattiche alla pagina http://www.physycom.unibo.it/labinfo/ curate dal professore G.Servizi