Matrix Calc

Stefano Nordio

Email: stefanonordio96@gmail.com
LinkedIn: https://www.linkedin.com/in/stefano-nordio

Indice

1 Premessa

La cartella “MatrixCalc” contiene il progetto c++ con il relativo file MatrixCalc.pro necessario alla generazione automatica tramite qmake del Makefile.

2 Abstract

Il progetto si prefigge lo scopo di creare una calcolatrice che permette l’uso di operazioni e calcoli sulle matrici. Le matrici inserite per i calcoli potranno essere di due tipi: quadrate e non quadrate.
Nella calcolatrice sono presenti sia operazioni che comportano l’utilizzo di due matrici sia operazioni che riguardano una singola matrice; la prima tipologia include le classiche addizione, sottrazione e moltiplicazione, mentre nella seconda compaiono prodotto per scalare, matrice trasposta, traccia della matrice, eliminazione di Gauss, determinante della matrice, rango della matrice e matrice inversa.
Le operazioni permesse ovviamente dipenderanno dal tipo di matrici inserite (non sarà ad esempio possibile calcolare la traccia se la matrice inserita non è quadrata) e dai vincoli teorici delle singole operazioni (non sarà ad esempio possibile sommare due matrici se queste non presentano le medesime dimensioni).
Le matrici inserite su cui effettuare i calcoli potranno essere riempite con numeri interi, razionali e complessi. Questi ultimi potranno includere, nella parte reale e immaginaria, sia elementi interi sia razionali.
Tutti i tipi inseribili potranno comparire contemporaneamente nella stessa matrice.

3 Gerarchie di Tipi Usate

Il progetto presenta due gerarchie nella parte logica, una che rappresenta l’elemento cardine della calcolatrice, ossia le matrici, e una invece riguardante le eccezioni che possono essere lanciate dal programma.

3.1 Gerarchia delle Matrici

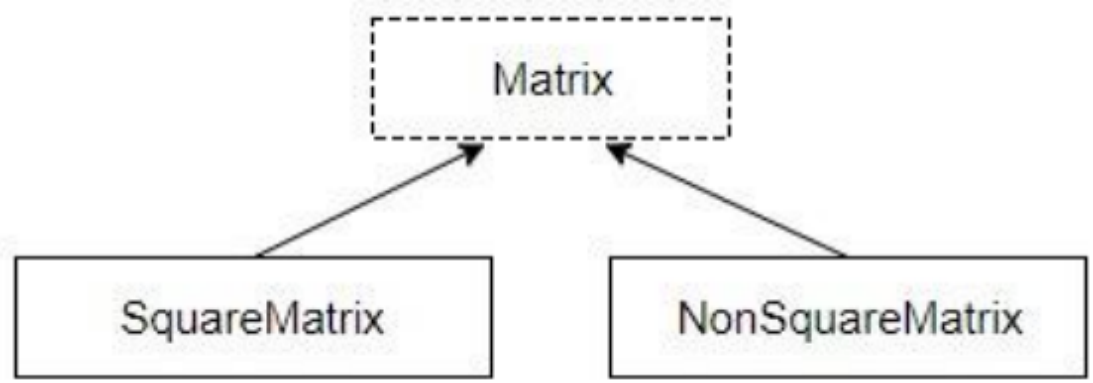

Figura 1: Gerarchia delle Matrici

3.1.1 Matrix

La prima gerarchia riguarda l'implementazione logica delle matrici.
Tutta la gerarchia è stata resa template in modo tale da poter istanziare i valori delle matrici con diversi tipi di dati numerici, primitivi e non.
La scelta di rendere il valore di default del template come double è nata dal fatto che son presenti dei metodi (vedi eliminazione di Gauss) che facendo operazioni di divisione tra gli elementi non permettono di gestire valori interi.
Si parte dalla classe basa astratta Matrix che fornisce l'interfaccia comune ai due tipi derivati. I 3 campi dati rows, columns e values sono stati marchiati protected al fine di renderli accessibili direttamente dalle due classi derivate ma non dall'esterno.
La scelta del vector della libreria STL è stata fatta per non dipendere dal framework QT, per avere un modello istanziabile con diversi tipi di dati e per il fatto che automaticamente gestisce la memoria in caso di distruzione.
Sono poi definiti i costruttori ad uno e due parametri, utilizzati dalle classi derivate, il costruttore di copia avente lo stesso comportamento di quello standard e il distruttore reso virtuale in modo tale da richiamare il giusto distruttore della gerarchia in caso di distruzione.
Tra i metodi implementati direttamente nella classe base ci sono i get di righe e colonne, gli operatori di input e output utilizzati principalmente per testare il corretto funzionamento da riga di comando, gli operatori di accesso agli elementi della matrice, l'operatore di uguaglianza, gli operatori di assegnazione e il metodo per il calcolo dell'eliminazione di Gauss che effettua l'operazione sulla matrice d'invocazione e ritorna un valore intero che se uguale al numero di righe della matrice d'invocazione indica la corretta terminazione dell'algoritmo di eliminazione, altrimenti indica un arresto anticipato in quanto la matrice passata è singolare, cioè l'eliminazione di Gauss è arrivata a trovare una riga composta da tutti 0.
Infine sono presenti i metodi virtuali puri di copia clone(), somma, differenza, prodotto, prodotto per scalare, calcolo del rango e della matrice trasposta che sono implementati nelle classi derivate.
Sono stati resi virtuali puri in quanto hanno bisogno di lavorare con oggetti concreti e quindi implementarli direttamente nella classe base sarebbe stato logicamente errato.
Questa rappresenta quindi l'interfaccia delle matrici quadrate e non quadrate che potranno utilizzare tutto ciò appena descritto.

3.1.2 NonSquareMatrix

La classe derivata concreta NonSquareMatrix implementa il costruttore a due parametri (righe, colonne) e il costruttore di copia utilizzando i relativi costruttori della classe padre.
Oltre poi al distruttore con comportamento standard la classe NonSquareMatrix implementa i metodi virtuali puri definiti in Matrix.

3.1.3 SquareMatrix

La classe derivata concreta SquareMatrix implementa il costruttore a un parametro (righe e colonne sono le medesime) e il costruttore di copia utilizzando i costruttori della classe padre.
Sono poi implementati distruttore e metodi virtuali puri della classe base che a differenza

delle implementazioni nella classe NonSquareMatrix lavoreranno con oggetti relativi a matrici quadrate.
Sono poi implementate operazioni esclusive per le matrici quadrate, ossia il metodo per il calcolo della traccia, del determinante e della matrice inversa.
Il metodo trace() semplicemente ha il compito di sommare gli elementi diagonali della matrice.
Il metodo determinant() calcola il determinante della matrice sfruttando il metodo di Gauss; effettua prima l'eliminazione di Gauss della matrice e poi ne somma gli elementi diagonali.
Il numero di scambi tra righe nell'eliminazione di Gauss deve essere in numero pari altrimenti il determinante cambierebbe di segno; per ovviare a questo dettaglio su cui fare attenzione è stato messo un flag booleano nel metodo di Gauss che ad ogni scambio (swap) muta da true a false o viceversa in modo da poter verificare tramite il valore intero restituito se gli scambi sono stati in numero pari (valore di ritorno positivo) o in numero dispari (valore di ritorno negativo).
Se la matrice tramite il metodo di Gauss risulta singolare (per il motivo descritto sopra) allora il determinante sarà uguale a 0 altrimenti restituisce la somma degli elementi diagonali facendo attenzione al segno del valore ritornato da Gauss.
Infine il metodo inverse() che restituisce la matrice inversa utilizzando il metodo di Gauss-Jordan.
Per farlo viene invocato il metodo inverseExt() che ha il compito di procedere all'eliminazione di Gauss della matrice estesa con la matrice identità e poi effettuare i vari pivoting per arrivare a restituire l'inversa.

3.2 Gerarchia delle Eccezioni

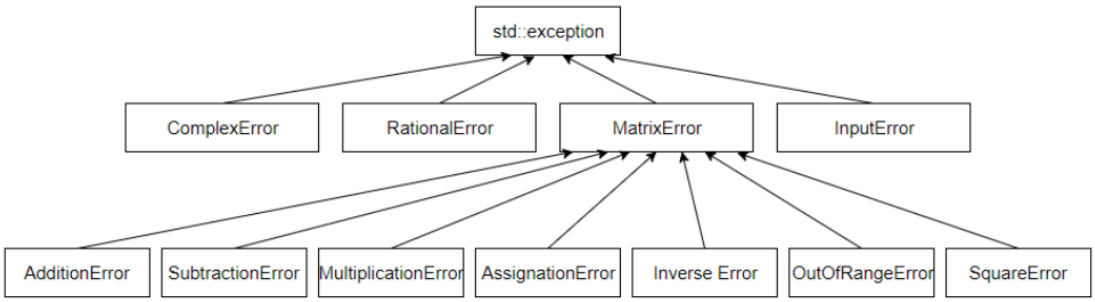

Figura 2: Gerarchia delle Eccezioni

La gerarchia delle eccezioni presenta 4 macro categorie derivate da std::exceptions: ComplexError che riguarda le eccezioni relative ai numeri complessi, RationalError che riguarda le eccezioni dei numeri razionali, MatrixError che gestisce le eccezioni riguardanti le matrici e InputError necessario per lanciare l'eccezione in caso di input errato.
La classe MatrixError dispone di diverse sottoclassi relative a specifiche eccezioni che riguardano le matrici.
Tutte le classi definite dispongono semplicemente della ridefinizione del metodo virtuale what() che permette di mostrare in caso di lancio dell'eccezione il messaggio d'errore definito.

4 Classi dei Tipi Numerici

Sono presenti due classi che definiscono nuovi tipi numerici con il quale è possibile popolare le matrici: Rational e Complex.

4.1 Rational

La classe Rational implementa la logica che permette l'utilizzo di elementi numerici razionali all'interno delle matrici.
Tra i campi dati privati sono presenti num e den ridefiniti in corso d'opera come long long int per permettere ai metodi implementati nelle matrici di lavorare correttamente senza andare in overflow visto che utilizzano diverse moltiplicazioni che potrebbero generare, soprattutto nel caso di numeri complessi, risultati molto grandi.
Tra i campi dati privati ci sono poi una funzione Normalize() che normalizza il razionale nel caso in cui il denominatore sia negativo e un metodo Simplify() che semplifica, se possibile, il razionale sfruttando il metodo statico MCD.
Nella parte pubblica sono presenti il costruttore a due parametri inizializzati di default (con denominatore uguale a 1), costruttore di copia e distruttore con comportamento standard, i due metodi di get di numeratore e denominatore, e tutte le varie operazioni possibili.
Anche se non tutte le operazioni verranno effettivamente utilizzate dal progetto sono state implementate in fase iniziale di progettazione logica in modo da garantire una completa definizione degli elementi razionali.

4.2 Complex

La classe Complex implementa la logica che permette l'utilizzo di elementi numerici complessi all'interno delle matrici.
La classe è stata resa template in modo da poter lasciare libertà sulla scelta della natura numerica della parte reale e immaginaria del numero complesso.
Nel nostro progetto è stata sfruttata la classe Rational così da poter utilizzare numeri complessi con parte reale e immaginaria razionale.
Tra i campi privati sono presenti la parte reale e immaginaria aventi tipo del template mentre nella parte pubblica ci sono i soliti costruttore a due parametri inizializzati di default, costruttore di copia e distruttore con comportamento standard, i due metodi di get per la parte reale e la parte immaginaria, e tutte le operazioni possibili che, come per i razionali, sono state definite nella maniera più completa seppur senza il reale utilizzo nel progetto.

5 Progettazione GUI

Per la progettazione dell'interfaccia grafica è stato adottato il pattern architetturale Model-View-Controller.
Così facendo vengono suddivisi i compiti, la parte logica è interamente gestita dal Model e la View si occupa di presentare la parte logica all'utente.
Il tutto è gestito dal Controller che si occupa di inoltrare le interazioni dell'utente nella View verso il Model cercando di dare sempre una risposta, sia nel caso positivo sia in quello negativo.

5.1 ModelMatrix

Il Model è caratterizzato da tre puntatori alla classe base Matrix che rappresentano le due matrici operando e la matrice risultato.
In questo modo viene sfruttato il polimorfismo che può presentarsi a run-time e, ove possibile, viene invocato il metodo corretto nelle classi figlie.
Nei casi in cui venga richiesta l'invocazione di un metodo non virtuale relativo ad un certo tipo vengono fatti dei cast sia per verificare la possibilità d'invocazione sia per l'invocazione stessa.
La scelta di utilizzare i puntatori a matrici istanziate esplicitamente a Complex<Rational> deriva dal fatto che in questo modo è possibile sfruttare i costruttori di Complex e Rational per poter inserire nelle matrici valori interi, valori razionali e valori complessi.
Tutti i metodi relativi alle singole matrici vengono richieste dalla View con un parametro booleano per indicare se l'operazione deve essere invocata sulla prima matrice (true) oppure sulla seconda (false).

5.2 ControllerMatrix

Il Controller ha la funzione di inoltrare le interazioni utente nella View verso il Model.
È quindi caratterizzato da un puntatore a Model e permette di gestire tutte le eventuali eccezioni che vengono lanciate, mostrandole all'utente tramite delle MessageBox, in modo tale che queste non si propaghino verso i livelli superiori compromettendo la corretta esecuzione dell'applicazione.

5.3 MainWindow

La View è l'interfaccia grafica e ciò che l'utente vede e con il quale può interagire.
È quindi caratterizzata da un puntatore a Controller per permettere la creazione della catena MVC in modo da filtrare le richieste inviate al Model attraverso il Controller.
La creazione dell'interfaccia grafica è stata fatta con l'ausilio dello strumento QtDesigner che ha permesso una più semplice creazione dei vari widget con i relativi slot.
Le matrici vengono mostrate all'utente tramite delle TableWidget ridimensionabili in numero di righe e colonne tramite degli SpinBox mentre tutte le operazioni sono rese cliccabili con dei PushButton.
I metodi setOperand e setValues permettono di istanziare le matrici in base all'input dell'utente nelle tableWidget, il metodo printResult invece permette di mostrare il risultato a video.
Alla chiusura dell'applicazione vengono invocati a catena i distruttori per distruggere Model, Controller e View.

5.4 Parser

La classe Parser è stata implementata al fine di garantire una corretta codifica degli elementi inseriti nelle matrici visibili nell'interfaccia grafica all'utente.
È composta da due metodi statici che hanno comportamento simmetrico, il primo verifica tramite un'espressione regolare che l'input inserito dall'utente sia valido e in caso affermativo lo converte nel rispettivo valore complesso, il secondo invece trasforma i valori complessi

nelle relative stringhe ed è necessario per stampare la matrice risultato che l'utente vede nell'interfaccia grafica.

6 Manuale Utente

L'interfaccia che si presenta all'apertura dell'applicazione è molto semplice ed intuitiva.
Nella parte alta sono presenti le due finestre che conterranno le matrici operando.
È necessario impostare le dimensioni delle matrici, dopodiché è possibile inserire i valori con un doppio click all'interno delle singole celle.
Tra le due matrici si trovano i pulsanti relativi alla somma, alla differenza e al prodotto.
Sotto le matrici si trovano invece i pulsanti per le operazioni sulla singola matrice.
Al click di una qualunque operazione, se questa non può essere applicata verrà mostrato un messaggio d'errore in una finestra pop-up, viceversa se l'operazione va a buon fine verrà mostrato il risultato nella finestra in basso all'applicazione.
Per chiudere l'applicazione basterà premere nel classico pulsante X di chiusura e confermare l'operazione.
Di seguito dei valori d'esempio per l'inserimento nelle matrici:

- Intero positivo: 5
- Intero negativo: -5
- Razionale positivo: 5/2
- Razionale negativo: -5/2
- Complesso con valori interi: 5+5i
- Complesso con valori razionali: 5/2+5/2i
- Complesso con solo parte immaginaria intera: 0+5i
- Complesso con solo parte immaginaria razionale: 0+5/2i

Fare attenzione a non digitare spazi nell'inserimento previo segnalazione d'errore da parte della calcolatrice.

7 Ore Utilizzate

Fase	Durata
Analisi Preliminare del Problema	4h
Progettazione Modello e GUI	15h
Apprendimento Libreria Qt	7h
Codifica Modello e GUI	15h
Debugging	4h
Testing	4h
TOTALE	**49h**

Tabella 1: Tabella Ore Utilizzate

8 Ambiente di Sviluppo

Programma	Versione
Sistema Operativo	Microsoft Windows 10 Education 32-bit 10.0.16299
Compilatore	gcc (MinGW-W64) 5.3.0
QT	5.9.3
Qmake	3.1

Tabella 2: Tabella Ambiente di Sviluppo