Linear Algebra Library Junior C++

Authors:
Juanita Gómez
Santiago Lopez
Oscar Velasco

Presented to:
Julian Jiménez

Algorithms and Data Structures
Universidad del Rosario

May 20, 2019

1 Abstract

In C++ it is difficult to represent and perform operations between mathematical objects, such as vectors and matrices. This project consists on the implementation of a basic linear algebra library, with two C++ classes, one for matrices and one for vectors, in which these objects can be represented easily. This, in order to develop some methods to perform operations between them and find important properties. The classes are useful for performing mathematical operations on vectors and matrices because they include several features that allow the interaction and analysis of these math objects according to their properties.

2 Functionality

The project is divided into two classes, Cmatrix and Cvector, each one with its own constructors and methods. For the Cmatrix class, objects are being constructed using Cvector objects allowing the use of the vector properties and methods in the Cmatrix class.
The methods implemented in both classes allow the representation and interaction of the mathematical objects for linear algebra applications. The tool is designed to allow the following:

1. Construction of matrices and vectors allowing actions like “pull”, “insert”, “erase”, “clear” which are proper of vectors.
2. Operations between matrices, vectors and scalars such as addition, substraction, multiplication and division using regular operators as +, -, * and /.
3. Comparison of matrices and vectors using operators ==, !=, >=, <=, <, >.
4. Operations proper of vectors like dot product, cross product, norm computation, normalization, angle computation and vector projections.
5. Computation of an orthonormal base of vectors using Gram-Schmidt method.
6. Computation of some special matrices such as the transpose and the inverse.
7. Decomposition of matrices in easier representations such as LUP and QR.
8. Computation of matrix main properties such as determinant and eigen values.

3 Description

3.1 Cvector

Cvector is a class implemented intended to represent vectors. It consists on the following attributes:

- **Capacity**: Refers to the number of elements that the object can store. It is expanded whenever the object is full.
- **Length**: Refers to the number of elements present at the object.
- **Array**: Pointer to a dynamic array which stores the elements of the Cvector.

3.1.1 CONSTRUCTORS

- **Empty**: It does not receive any parameter and creates an object of empty type Cvector with length 0 and capacity equal to Initial Capacity.
- **Fill**: Receives a size_t Length and a numberType value and creates an object of type Empty Vector with length Length, capacity equal to Initial Capacity and each of its elements is equal to value.
- **Size**: It receives a size_t length and creates an object of type Cvector with length Length, capacity equal to Initial Capacity with indeterminate elements.
- **Parametric**: It receives a Cvector rhs and creates an object of type Cvector by copying the length, capacity and elements of rhs.

3.1.2 OPERATORS

Class Member Operators

- [ ]: Access elements in vector.
- =: Assign elements in vector.

Comparisson Operators

- (==, !=, >, <, >=, <=): Compares vectors elementwise, returning a vector of bools.

Ostream Operator

- <<: Allows the printing of the vector showing its elements.

Binary Operators

- (+,-,*,/): Operates vectors elementwise when the parameters are two vectors. In the case of * and /, it allows multiplication and division of a vector by an scalar.

3.1.3 CLASS METHODS

Vector Methods

- `toDouble()`: Converts the elements in the Cvector to double precision.
- `push(numberType value)`: Receives a numberType value and inserts it at the end of the vector. It does not return anything.
- `erase(size_t index)`: Receives a size_t index and deletes the vector element corresponding to that index. It does not return anything.
- `insert(size_t index, numberType value)`: Receives a size_t index and a numberType value and inserts it into the position of the vector corresponding to that index. It does not return anything.
- `clear()`: It does not receive parameters and it assigns the length of the vector "this" equal to 0.
- `empty()`: It does not receive parameters and verifies if the vector is empty. It returns a bool.
- `size()`: It does not receive parameters and returns the length of the vector.

Math Methods

- `dot(Cvector<numberType> w)`: It receives a Cvector w and calculates the point product between "this" and the Cvector w. Returns a double type scalar.
- `cross(Cvector<numberType> w)`: It receives a Cvector w and calculates the cross product between "this" and the Cvector w. Returns a Cvector.
- `norm()`: Calculates the norm of the vector and returns a double.
- `normalize()`: Normalizes the vector and returns a Cvector of doubles.
- `angle(Cvector<numberType> &x)`: Calculates the angle in radians between the angle x and "this" returning a double.
- `proj(Cvector<numberType> &x)`: Calculates the proyection of the vector x in the vector "this" and returns a vector of doubles.
- `gram_schmidt()`: Receives a vector of vectors and ortonormalizes the base formed by these vectors, returning another vector of vectors.

Private Methods

- `expandCapacity()`: Expands the capacity of the Cvector by doubling it.
- `Checkrep()`: Asserts that the length of the vector is always greater than 0 and that the length of the vector is always greater or equal than the capacity.

3.2 Cmatrix

Cmatrix is a class implemented for C++ language, intended to represent matrices. It consists on the following attributes:

- **Capacity**: Refers to the number of elements that the object can store. It is expanded whenever the object is full.
- **nRows, nCols**: Refers to the number of rows and columns respectively present at the object.
- **array**: Pointer to a dynamic array of Cvectors which stores the elements of the matrix (Cvectors).

3.2.1 CONSTRUCTORS

- **Empty**: It does not receive any parameter and creates an object of empty type Cmatrix with length 0 and capacity equal to Initial Capacity.
- **Fill**: Receives a size_t Length and a numberType value and creates an object of type Empty matrix with length length, capacity equal to Initial Capacity and each of its elements is equal to value.
- **Parametric**: It receives a Cmatrix rhs and creates an object of type Cmatrix by copying the length, capacity and elements of rhs.
- **Specialized**: It receives a size_t row and a size_t col and a bool. If the bool is true, and the values of row and col are equal, it constructs an identity matrix. If the bool is false it constructs a matrix of size $row \times col$ filled with zeros.

3.2.2 OPERATORS

Class Member Operators

- [ ], (): Access elements in matrix.

Binary Operators

- *: Depending on the parameters computes Matrix-Matrix multiplication, Vector-Matrix Multiplication, Escalar-Matrix Multiplication.

3.2.3 CLASS METHODS

Matrix Methods

- `toDouble()`: Converts the elements in the Cmatrix to double precision.
- `push(const Cvector<numberType> &value, bool axis = 0)`: Receives a numberType value and inserts it at the end of the matrix. It does not return anything.
- `erase(size_t index, bool axis)`: Receives a size_t index and deletes the matrix element corresponding to that index. It does not return anything.
- `insert(size_t index, const Cvector<numberType> &value, bool axis = 0)`: Receives a size_t index and a numberType value and inserts it into the position of the matrix corresponding to that index. It does not return anything.
- `numberRows()`: It does not receive parameters and returns the number of rows of the matrix.
- `numberCols()`: It does not receive parameters and returns the number of columns of the matrix.
- `access(size_t row, size_t cod)`: It receives a size_t row and a size_t cod and it access to the element row, cod of the matrix.

Easy matrices

- `eye(size_t N)`: Creates an identity matrix of size $N \times N$
- `zeros(size_t rows, size_t cols)`: Creates a matrix of zeros of size $rows \times cols$
- `ones(size_t rows, size_t cols)`: Creates a matrix of ones of size $rows \times cols$
- `random(size_t rows, size_t cols)`: Creates a matrix of random numbers of size $rows \times cols$
- `diagonalize(const Cvector<numberType> &rhs)`: Creates a matrix with a Cvector v as diagonal and the rest of its elements equal to 0.
- `diagonal(Cmatrix<numberType> &m)`: Creates a vector with the elements of the diagonal of a matrix.
- `permutationMatrix(Cvector<numberType> &v)`: Creates a permutation matrix from a vector where each element of the vector indicates a "1" in the corresponding column index.

Modifiers

- `swap_r(size_t row1, size_t row2)`: Swaps rows row1 and row2.
- `swap_c(size_t col1, size_t col2)`: Swaps columns col1 and col2.
- `appendRows(const Cmatrix<numberType> &rhs)`: Appends the rows of a matrix m to the matrix "this".
- `appendCols(const Cmatrix<numberType> &rhs)`: Appends the columns of a matrix m to the matrix "this".

Special matrices

- `abs()`: Returns a matrix with the absolute value elementwise of "this".
- `transpose()`: Returns the transpose of "this".
- `inverse()`: Returns the inverse of "this".
- `lowerTriangular()`: Returns a lowerTriangular matrix copying the corresponding elements of "this".
- `upperTriangular()`: Returns a upperTriangular matrix copying the corresponding elements of "this".

Matrix Decompositions

- `LUP(double Tol)`: Computes the LUP decomposition of the matrix "this". Returns a tuple composed by the matrix L, the matrix U and a vector P that corresponds to the permutation matrix of the decomposition.
- `QR()`: Computes the QR decomposition of the matrix "this". Returns a tuple composed by the matrix Q, and the matrix R.

Matrix Properties

- `determinant()`: Returns a double corresponding to the determinant of the matrix "this".
- `eigen_values(const double tol)`: Returns a vector containing the eigen values of the matriz by iterating the QR method.
- `Checkrep()`: Check the invariant representation of the class.

Private Methods

- `expandCapacity()`: Expands the number of rows of the Cmatrix by doubling it.

References

[1] Chen, X., *Numerical Methods, Math 375: Numerical Analysis in Fall 2017* https://github.com/Numerical-Analysis/course/blob/master/NumericalMethodsNotes.pdf, 2017