UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

COS110 Assignment 2

Matrix Arithmetic

Due date: 27 September 2020, 23:30

1 General instructions

- This assignment should be completed individually.
- Be ready to upload your assignment well before the deadline as no extension will be granted.
- If your code does not compile you will be awarded a mark of 0. The output of your program will be primarily considered for marks, although internal structure may also be tested (eg. the presence of certain functions or classes).
- Read the entire assignment thoroughly before you start coding.
- You are not allowed to use any mathematical C/C++ libraries to complete this assignment.
- To ensure that you did not plagiarize, your code will be inspected with the help of dedicated software.
- Note that plagiarism is considered a very serious offence. Plagiarism will not be tolerated and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at `http://www.ais.up.ac.za/plagiarism/index.htm`.

2 Overview

For this assignment, you will create a simplistic mathematical "library" to perform various matrix manipulations and solve systems of linear equations. Operator overloading will be used to implement the matrix arithmetic.

3 Matrices

In mathematics, a matrix is a rectangular array of numbers, symbols, or expressions, arranged in rows and columns. Every element in a matrix is described by two indices `i` and `j`, where `i` is the row number of the element, and `j` is the column number of the element. For example, given a matrix A:

$$A = \begin{bmatrix} 1 & 5 & 7 \\ 8 & 7 & 3 \\ 2 & 9 & 1 \end{bmatrix}$$

The A(2,3) element of A is equal to 3, because that is the value located at row 2 and column 3 of A. Matrices can have any number of rows or columns:

$$\begin{bmatrix}7 & 8 & 9\\2 & 5 & 1\end{bmatrix}, \begin{bmatrix}5 & 3\\4 & 4\\9 & 0\end{bmatrix}, \begin{bmatrix}7\\4\\0\\3\\1\end{bmatrix}, \begin{bmatrix}4 & 4\\9 & 0\end{bmatrix}, \begin{bmatrix}4 & 7 & 8 & 9\\6 & 2 & 5 & 1\\5 & 6 & 7 & 8\end{bmatrix}, \begin{bmatrix}7 & 8 & 9 & 5 & 3 & 5\end{bmatrix}$$

Matrices are used extensively in most scientific fields. In every branch of physics, including classical mechanics, optics, electromagnetism, quantum mechanics, and quantum electrodynamics, they are used to study physical phenomena, such as the motion of rigid bodies. In computer graphics, they are used to project a 3-dimensional image onto a 2-dimensional screen. For this assignment, you will implement simple matrix mathematical operations.

4 Matrix arithmetic

This section describes the matrix mathematical operations that you are required to implement for this assignment.

4.1 Addition, subtraction, scalar multiplication and division

The simplest matrix operation is addition and subtraction of matrices. Only matrices with the same dimensions (equal number of rows and columns) can be added or subtracted from one another. Matrix addition is done element-wise: given two matrices A and B of the same dimension, every element A(i,j) is added to the element B(i,j):

$$\begin{bmatrix}2 & 3 & 1\\3 & 6 & 5\end{bmatrix} + \begin{bmatrix}1 & 5 & 7\\2 & 3 & 4\end{bmatrix} = \begin{bmatrix}2+1 & 3+5 & 1+7\\3+2 & 6+3 & 5+4\end{bmatrix} = \begin{bmatrix}3 & 8 & 8\\5 & 9 & 9\end{bmatrix}$$

The same rule applies to matrix subtraction.

Matrices can also be multiplied or divided by a scalar (i.e. non-matrix) value. To multiply a matrix A by a scalar `b`, every element A(i,j) is multiplied by `b`. Similarly, to divide a matrix A by a scalar `b`, every element A(i,j) is divided by `b`:

$$\begin{bmatrix}2 & 3 & 0\\3 & 6 & 4\end{bmatrix} /2 = \begin{bmatrix}2/2 & 3/2 & 0/2\\3/2 & 6/2 & 4/2\end{bmatrix} = \begin{bmatrix}1 & 1.5 & 0\\1.5 & 3 & 2\end{bmatrix}$$

Scalar multiplication:

$$2 * \begin{bmatrix}3 & 5\\1 & 4\end{bmatrix} = \begin{bmatrix}2*3 & 2*5\\2*1 & 2*4\end{bmatrix} = \begin{bmatrix}6 & 10\\2 & 8\end{bmatrix}$$

4.2 Matrix multiplication

Matrix A can be multiplied by a matrix B, provided that the number of columns in A is equal to the number of rows in B. Such restriction is due to the way matrices are multiplied. Before we discuss matrix multiplication further, let us define the *dot product* of two 1-dimensional matrices. Given a `1 x n` matrix A and a `n x 1` matrix B, their dot product is defined as the sum of products of `A(1,j)` and `B(j,1)` for every `0 < j < n`:

$$A = \begin{bmatrix}a_1 & a_2 & \dots & a_n\end{bmatrix}, B = \begin{bmatrix}b_1\\b_2\\\vdots\\b_n\end{bmatrix}$$

$$A.B = a_1 * b_1 + a_2 * b_2 + \cdots + a_n * b_n$$

For example,

$$A = \begin{bmatrix} 2 & 3 & 1 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 4 \\ 5 \end{bmatrix}$$

$$A.B = 2 * 0 + 3 * 4 + 1 * 5 = 0 + 12 + 5 = 17$$

We can say that every `n x m` matrix A consists of `n` 1-dimensional row matrices A(ri), `0 < i < n`, and `m` 1-dimensional column matrices A(cj), `0 < j < m`. Now, to multiply a `n x p` matrix A by a `p x m` matrix B, we say that every `(i,j)` element of the resulting `n x m` matrix C will be the dot product of A(ri) (i-th row of A) and B(cj) (j-th column of B). Let A(rx) be a **row** of A, and B(cx) a **column** of B:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1p} \\ a_{21} & a_{22} & \dots & a_{2p} \\ \vdots & & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{np} \end{bmatrix}, B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ \vdots & & & \vdots \\ b_{p1} & b_{p2} & \dots & b_{pm} \end{bmatrix}$$

$$C = A * B = \begin{bmatrix} A(r1).B(c1) & A(r1).B(c2) & \dots & A(r1).B(cm) \\ A(r2).B(c1) & A(r2).B(c2) & \dots & A(r2).B(cm) \\ \vdots & & & \vdots \\ A(rn).B(c1) & A(rn).B(c2) & \dots & A(rn).B(cm) \end{bmatrix}$$

For example:

$$A = \begin{bmatrix} 2 & 3 & 1 \\ 5 & 1 & 0 \end{bmatrix}, B = \begin{bmatrix} 4 & 2 \\ 8 & 1 \\ 3 & 4 \end{bmatrix}$$

$$A * B = \begin{bmatrix} A(r1).B(c1) & A(r1).B(c2) \\ A(r2).B(c1) & A(r2).B(c2) \end{bmatrix} = \begin{bmatrix} (2 * 4 + 3 * 8 + 1 * 3) & (2 * 2 + 3 * 1 + 1 * 4) \\ (5 * 4 + 1 * 8 + 0 * 3) & (5 * 2 + 1 * 1 + 0 * 4) \end{bmatrix} =$$

$$= \begin{bmatrix} (8 + 24 + 3) & (4 + 3 + 4) \\ (20 + 8 + 0) & (10 + 1 + 0) \end{bmatrix} = \begin{bmatrix} 35 & 11 \\ 28 & 11 \end{bmatrix}$$

Note that A*B is not equal to B*A:

$$B * A = \begin{bmatrix} B(r1).A(c1) & B(r1).A(c2) & B(r1).A(c3) \\ B(r2).A(c1) & B(r2).A(c2) & B(r2).A(c3) \\ B(r3).A(c1) & B(r3).A(c2) & B(r3).A(c3) \end{bmatrix} =$$

$$= \begin{bmatrix} (4 * 2 + 2 * 5) & (4 * 3 + 2 * 1) & (4 * 1 + 2 * 0) \\ (8 * 2 + 1 * 5) & (8 * 3 + 1 * 1) & (8 * 1 + 1 * 0) \\ (3 * 2 + 4 * 5) & (3 * 3 + 4 * 1) & (3 * 1 + 4 * 0) \end{bmatrix} = \begin{bmatrix} (8 + 10) & (12 + 2) & (4 + 0) \\ (16 + 5) & (24 + 1) & (8 + 0) \\ (6 + 20) & (9 + 4) & (3 + 0) \end{bmatrix} =$$

$$= \begin{bmatrix} 18 & 14 & 4 \\ 21 & 25 & 8 \\ 26 & 13 & 3 \end{bmatrix}$$

For more information on matrix multiplication, refer to `http://www.mathsisfun.com/algebra/matrix-multiplying.html`.

4.3 Powers of a matrix

Given a non-negative integer k, a k-th power of a square (`n x n`) matrix A can be calculated by multiplying A by itself k times. If k is zero, the matrix must be transformed into an **identity** matrix, i.e. a square matrix of all zeros, with ones only on the `{i,i}` diagonal:

$$A = \begin{bmatrix} 2 & 3 \\ 5 & 1 \end{bmatrix}, A^0 = I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

General case for `k = 0`:

$$A = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & & & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nn} \end{bmatrix}, A^0 = I = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & & & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

5 Systems of linear equations

Now that you know how matrices work, let us consider how matrices can be used to solve systems of linear equations.

A linear equation is an equation that describes a straight line. Recall that it is very easy to solve a linear equation with a single unknown:

$$\begin{aligned} 2x + 3 &= 6 \\ 2x &= 6 - 3 = 3 \\ \therefore x &= 3/2 = 1.5 \end{aligned}$$

Now consider a linear equation with two unknowns:

$$2x + 3y = 12$$

Clearly, no obvious solution to the above equation exists. However, if we were given a *system* of two linear equations, we could have expressed one unknown in terms of the other one, and find out the values of both unknowns by substitution:

$$\begin{cases} 2x + 3y & = 12 \\ x + 2y & = 4 \end{cases} \Rightarrow \begin{cases} 2x + 3y = 12 \\ x = 4 - 2y \end{cases} \Rightarrow \begin{cases} 2(4 - 2y) + 3y = 12 \\ x = 4 - 2y \end{cases} \Rightarrow$$

$$\begin{cases} 8 - 4y + 3y = 12 \\ x = 4 - 2y \end{cases} \Rightarrow \begin{cases} 8 - y = 12 \\ x = 4 - 2y \end{cases} \Rightarrow \begin{cases} y = 8 - 12 = -4 \\ x = 4 - 2y = 4 - 2 * (-4) = 4 + 8 = 12 \end{cases}$$

Solving a small linear system by hand is easy, but imagine having to deal with a dozen or a hundred of unknowns. For such cases, a more efficient solution is necessary. One way to automatically solve a system of linear equations is to use the *Cramer's Rule* algorithm. Cramer's Rule solves systems of linear equations using determinants.

A determinant is a real number that can be very useful in mathematics because it has multiple applications, such as calculating area, volume, and other quantities. Here, we will use determinants to reveal whether a matrix is invertible by using the entries of a square matrix to determine whether there is a solution to the system of equations.

5.1 Using Cramer's Rule to Solve a System of Two Equations

The determinant of a 2×2 matrix, given

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

is defined as

$$\det(A) = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - cb$$

Notice the change in notation. There are several ways to indicate the determinant, including det(A) and replacing the brackets in a matrix with straight lines, |A|.

To understand Cramer's Rule, let's look closely at how we solve systems of linear equations using basic row operations.

Consider the following system of equations:

$$\begin{cases} 12x + 3y & = 15 \\ 2x - 3y & = 13 \end{cases}$$

where coefficients are in red.

Solve for `D`

$$D = \begin{vmatrix} 12 & 3 \\ 2 & -3 \end{vmatrix} = (12)(-3) - (2)(3) = -36 - 6 = -42$$

Solve for `x`

$$x = \frac{D_x}{D} = \frac{\begin{vmatrix} 15 & 3 \\ 13 & -3 \end{vmatrix}}{-42} = \frac{-45 - 39}{-42} = \frac{-84}{-42} = 2$$

Solve for `y`

$$y = \frac{D_y}{D} = \frac{\begin{vmatrix} 12 & 15 \\ 2 & 13 \end{vmatrix}}{-42} = \frac{-156 - 30}{-42} = \frac{-126}{-42} = -3$$

Thus, the solution is (2,-3), where `x` = 2 and `y` = -3.

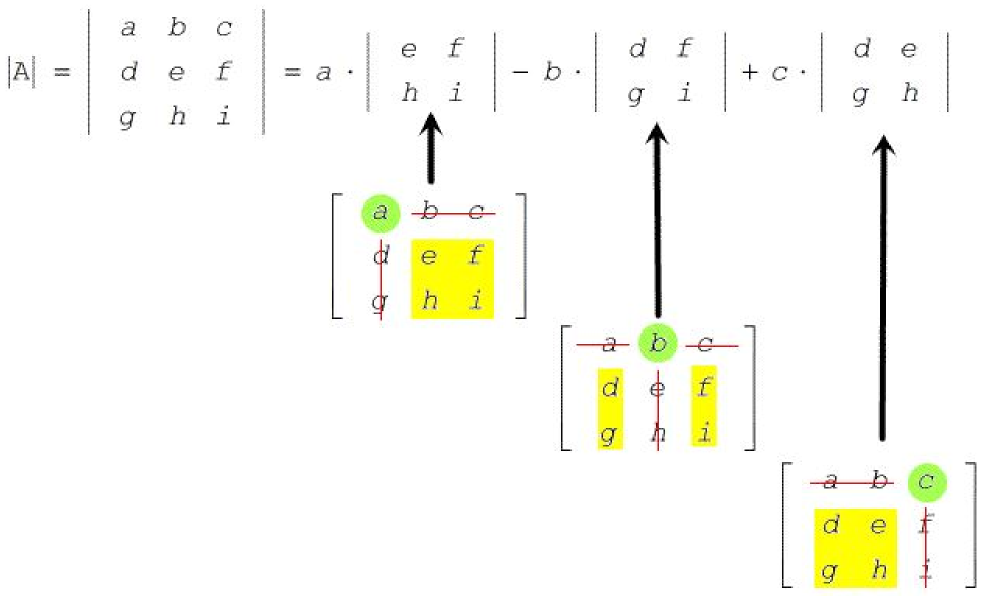
Moreover, D_x and D_y are the determinants for `x` and `y`, respectively.

5.2 Using Cramer's Rule to Solve a System of Three Equations

Finding the determinant of a 2×2 matrix is straightforward, but finding the determinant of a 3×3 matrix is more complicated. Evaluating the determinant of a 3 × 3 matrix, given

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

is defined as


|A| = | a b c ; d e f ; g h i | = a · | e f ; h i | − b · | d f ; g i | + c · | d e ; g h |
a b c d e f g h i
a b c d e f g h i
a b c d e f g h i

Consider the following system of equations:

$$\begin{cases} -2x - y - 3z & = 3 \\ 2x - 3y + z & = -13 \\ 2x \ \ - 3z & = 14 \end{cases}$$

Solve for `D`:

$$D = \begin{vmatrix} -2 & -1 & -3 \\ 2 & -3 & 1 \\ 2 & 0 & -3 \end{vmatrix} = (-2)\begin{vmatrix} -3 & 1 \\ 0 & -3 \end{vmatrix} - (-1)\begin{vmatrix} 2 & 1 \\ 2 & -3 \end{vmatrix} + (-3)\begin{vmatrix} 2 & -3 \\ 2 & 0 \end{vmatrix}$$

Use the algorithm described in section 5.1 to evaluate the determinant of a 2x2 matrix

$$= (-2)[-9] - (-1)[-8] + (-3)[-6] = -44$$

Solve for `x`:

$$x = \frac{D_x}{D} = \frac{\begin{vmatrix} 3 & -1 & -3 \\ -13 & -3 & 1 \\ -11 & 0 & -3 \end{vmatrix}}{-44} = \frac{176}{-44} = -4$$

Solve for `y`:

$$y = \frac{D_y}{D} = \frac{\begin{vmatrix} -2 & 3 & -3 \\ 2 & -13 & 1 \\ 2 & -11 & -3 \end{vmatrix}}{-44} = \frac{-88}{-44} = 2$$

Solve for `z`:

$$z = \frac{D_z}{D} = \frac{\begin{vmatrix} -2 & -1 & 3 \\ 2 & -3 & -13 \\ 2 & 0 & -11 \end{vmatrix}}{-44} = \frac{-44}{-44} = 1$$

The solution is (-4,2,1), i.e. `x = -4, y = 2, z = 1`.

To summarise, solving a system of linear equations with matrices and Cramer's Rule consists of two steps:

1. Find the determinant of the denominator D and the determinants for all variables, x and y for `2x2` matrix, or x, y and z for `3x3` matrix.
2. Divide each variable's determinant with the denominator D.

There are 2 cases:

Case I : When D is not 0. In this case we have, x = D_x/D, y = D_y/D, z = D_z/D. Hence unique value of x, y, z will be obtained.

Case II : When D = 0

1. When at least one of D_x, D_y and D_z is non zero: Then no solution is possible and hence system of equations will be inconsistent.
2. When D = 0 and $D_x = D_y = D_z = 0$: Then the system of equations will be consistent and it will have infinitely many solutions.

6 Your task

This assignment consists of three tasks:

1. Implementing the Matrix class with the basic input/output functionality.
2. Implementing the matrix arithmetic using operator overloading.
3. Implementing Cramer's Rule algorithm to solve systems of linear equations using the Matrix class for both two and three equations.

6.1 Task 1: Implementing the Matrix

File `matrix.h` contains an incomplete header of the Matrix class that you will implement for this task. A two-dimensional dynamic array of **`double`** values is used to store the matrix data. The dimensions of the array are stored in two member variables: `rows` and `cols`, which stand for rows and columns, respectively.

6.1.1 Constructors

Matrix constructor takes two **`unsigned int`** parameters, representing the number of rows and columns in the matrix. Upon receiving the dimension parameters, the constructor must allocate a dynamic array of the given size, and fill it with zeros. Remember to implement a destructor to deallocate the memory when the program exits. In addition to the constructor, implement an appropriate copy constructor, and overload the assignment operator `=` to enable assigning Matrix objects to one another.

6.1.2 Data input and output

To populate a matrix object with data, we will use operator overloading. Overload the stream insertion operator `>>` to read matrix data from either the keyboard or a file stream. Make sure you can use `>>` to input a n `x` n matrix stored in a text file in the following format:

```
2 4 3
5 2 3
7 1 0
```

Overload the stream extraction operator `<<` to output a matrix to the screen or file in the following format:

```
        2       4       3
        5       2       3
        7       1       0
```

Every element should occupy exactly 10 character spaces on the screen. The precision (number of decimal points) should be set to 3. Hint: use `setw()` and `setprecision()` functions of the `iomanip` library. Do not add any additional empty lines: just print the data.

To access individual elements of the Matrix, overload the parenthesis operator `()` to take two **`int`** parameters such that the following is possible (note that matrix indices start from 0,0):

```
Matrix mat(2,3);          // create a 2 x 3 matrix
mat(0,0) = 5;             // set element 0,0 to 5
cout << mat(0,0) << endl; // output element 0,0 to screen
```

Overloading `()` is very similar to overloading the subscript operator `[]`. Note that operator `()` has to take two **`unsigned int`** input parameters.

The subscript operator `[]` will be used to extract separate rows of a Matrix. Extracted rows have to be independent Matrix objects of `1 x n` dimension. Overload `[]` to take one **`unsigned int`** input parameter such that the following is possible:

```
Matrix mat(3,3);        // 3 x 3 matrix
Matrix matRow = mat[0]; // set matRow equal to the first row of mat
                        // dimension of matRow must be 1 x 3
```

You should also implement getter functions to return the number of rows and the number of columns in a matrix.

6.2 Task 2: Matrix arithmetic

All matrix arithmetic as described in Section 4 is to be implemented via operator overloading. Error checking has to be performed where appropriate, and if invalid input is given, output the appropriate error message (details in table 1) and return the left hand side operand (or right hand side operand if the left hand side is not of the appropriate type), or exit if the function is void. **Note:** only the operators with the = symbol should modify **`this`** object.

Table 1 summarises the operators that you have to overload. Assume that `a` and `b` are objects of the `Matrix` class, and `n` is of type **`double`**.

6.3 Task 3: Implementing Cramer's Rule

Cramer's Rule described in Section 5 is to be implemented for this part of the assignment. A system of linear equations will be represented by two separate Matrix objects: an `nxn` matrix `A` and a corresponding `nx1` matrix `b`.

Operator overloading will be used to implement Cramer's Rule.

Firstly, operator ~ will be used to find and set the determinant of `A` as follows:

```
Matrix A(2,2) // create a matrix
infile >> A; // read matrix data from file
double det = ~A; // calculate and return the determinant of A (see Sections 5.1 and 5.2)
```

For the context of this assignment, the determinant can only be calculated for `2x2` and `3x3` matrices. If a matrix of any other dimensionality is provided, print the message `"Error: invalid matrix dimensions"` and return zero. Note that the operator should not only return the determinant, but also store it in the private variable **`double`** `denominator`. Thus, matrix `A` is modified by the operator.

For the system of equations in Section 5.1, `A` = $\begin{bmatrix} 12 & 3 \\ 2 & -3 \end{bmatrix}$, thus `~A` must return `-42`.

Secondly, to perform Cramer's Rule, operator `|` has to be overloaded. Operator `|` should calculate and return the numerators (determinants) of `A` given `b`. Return type: dynamic array of

Table 1: Matrix arithmetic operators

Operator	Usage	Message	Example
`*`	matrix * n	none	`a*2`
`*`	n * matrix	none	`2*a`
`/`	matrix / n	If n is 0: "`Error: division by zero`"	`a/2`
`+`	matrix + matrix	If row/column dimensions do not match: "`Error: adding matrices of different dimensionality`"	`a + b`
`+=`	matrix += matrix	If row/column dimensions do not match: "`Error: adding matrices of different dimensionality`"	`a += b`
`-`	matrix - matrix	If row/column dimensions do not match: "`Error: subtracting matrices of different dimensionality`"	`a - b`
`-=`	matrix -= matrix	If row/column dimensions do not match: "`Error: subtracting matrices of different dimensionality`"	`a -= b`
`*`	matrix * matrix	If row/column dimensions are incorrect: "`Error: invalid matrix multiplication`"	`a * b`
`*=`	matrix *= matrix	If row/column dimensions are incorrect: "`Error: invalid matrix multiplication`"	`a *= b`
`^`	matrix to the power of n, matrix^n	If the matrix is not square (`n x n`): "`Error: non-square matrix provided`" If the power is negative: "`Error: negative power is not supported`"	`a^2`
`^=`	matrix to the power of n, matrix^=n	If the matrix is not square: "`Error: non-square matrix provided`" If the power is negative: "`Error: negative power is not supported`"	`a^=2`

double values, **`double *`**. You will return an array of two doubles for systems of two equations, and an array of three doubles for systems of three equations. In case of invalid input, print an error message as specified in Table 2, are return NULL.

Usage:

```
Matrix A(2,2), b(2,1); // declare matrices
infile >> A; // read matrix data from file
infile >> b; // read matrix data from file
double * determinants = A|b; // return Dx, Dy (for 2x2) and Dz (for 3x3)
```

The determinants are also stored in the private variable of Matrix: **`double`**`* linearDeterminants`. Thus, matrix A is modified by this operator.

For the system of equations in Section 5.1, `A` = $\begin{bmatrix} 12 & 3 \\ 2 & -3 \end{bmatrix}$, and `b` = $\begin{bmatrix} 15 \\ 13 \end{bmatrix}$. Thus, `A|b` must return the numerators (**`double`**`* linearDeterminants`) of matrix A containing values $\begin{bmatrix} -84 \\ -126 \end{bmatrix}$.

Finally, operator `|=` will be used to calculate the solution to the system of equations, and return it as a 1-column matrix. Given two matrices `A` and `b` representing a system of linear equations, the `n x 1` matrix of solutions can be obtained as follows:

```
Matrix  s = A |= b;
```

This operator can modify matrix `A`. The algorithm can be summarised as follows: check if the numerators and the denominator have already been calculated for `A`. If not, calculate the relevant determinants. Then, use Cramer's rule to obtain the final solution and return it as a `n x 1` matrix. In case of invalid input, return `b` without modifying `b`.

For the system of equations in Section 5.1, `A` = $\begin{bmatrix} 12 & 3 \\ 2 & -3 \end{bmatrix}$, and `b` = $\begin{bmatrix} 15 \\ 13 \end{bmatrix}$. Thus, `Matrix s = A |= b` must modify `A` such that `A` would contain the solution to the system of equations in **`double`**$*$ `linearSolutions`, `s` = $\begin{bmatrix} 2 \\ -3 \end{bmatrix}$.

Table 2 summarises the three operators that you have to overload for this task. Assume that `a` and `b` are objects of the `Matrix` class.

Table 2: Cramer's Rule operators overload

Operator	**Usage**	**Message**	**Example**
~	~matrix	If the matrix is not `2x2` or `3x3`: "`Error: invalid matrix dimensions`"	`~a`
|	matrix | matrix	If the left hand side matrix is not `n x n`: "`Error: non-square matrix provided`" If the left hand side matrix is not `2x2` or `3x3`: "`Error: incorrect number of variables`" If the right hand side matrix is not `n x 1`: "`Error: incorrect number of variables`"	`a|b`
`|=`	matrix |= matrix	If the left hand side matrix is not `2x2` or `3x3`: "`Error: incorrect number of variables`" If the right hand side matrix is not `n x 1`: "`Error: incorrect number of variables`" If division by zero is encountered during Cramer's Rule: "`Error: division by zero`"	`a|=b`

6.4 Implementation considerations

- Provide your own makefile
- `matrix.h` should be modified to include the declarations of the overloaded operators, as well as any additional helper functions your implementation might require. Note: For the purpose of this assignment, we will NOT overwrite you header files.
- `matrix.cpp` should be used to implement the Matrix class.
- Do not add any other `.h/.cpp` files, do not add extra classes.
- No mathematical C/C++ libraries can be used.
- Input data used to test your program will be space-separated.
- Remember to output error message where necessary.
- All error messages are simple one-line text messages, ending with a `\n` or `endl`. Do not add any additional breaks, new line characters or extra hyphenation.

6.5 Submission

Once you are satisfied that your program is working, compress all of your code, including a working makefile, into a single archive (either .tar.gz, .tar.bz2 or .zip) and submit it for marking to the appropriate upload link before the deadline. No object files or executables should be included in the archive. The archive must contain no folders or subfolders.

Your makefile should compile your code to an executable named `main`. Remember that your makefile must provide a run rule to execute your compiled program. That is:

```
run:
    ./main
```

A correct makefile would compile and link all code when the following command is executed in the terminal:

```
> make
```

A correct makefile would compile, link and run all code when the following command is executed in the terminal:

```
> make run
```

Note: Fitchfork will provide its own main function to test your code. Thus, any `main.cpp` submitted for marking will be overwritten.