Matrix manipulation and Matrix equation printing

Contents

Matrix Driver

Basic manipulation of matrices is implemented with this driver.

Exceptions

IncompatibleSizeException

The exception is thrown when an operation is being attempted on matrices that do not have the correct size to allow the operation to take place.

UndefinedMatrixException

The exception is thrown when the size of the matrix has not been defined. The number of rows and columns is 0.

InvalidOperandException

The exception is thrown when the given operand is invalid. The size of the matrix is not allowed by the operation or the determinant is 0 for computing the invers.

Basic operations for matrices

The driver provides basic mathematical operations for matrices like addition, subtraction, multiplication between matrices and multiplication of a scalar with a matrix.

Headers

- `Matrix* operator+(Matrix& a, Matrix& b);`
- `Matrix* operator-(Matrix& a, Matrix& b);`
- `Matrix* operator*(Matrix& a, Matrix& b);`
- `Matrix* operator*(float a, Matrix& b);`

MatrixIrregular header

MatrixIrregular::pseudoinvers

Description

The pseudoinverse of a given irregular matrix is computed depending on the independencies of the rows or columns. The method implements the two formulars.

Linearly independent columns: $A^{+} = (A^{t}A)^{-1}A^{t}$

Linearly independent rows: $A^{+} = A^{t}(AA^{t})^{-1}$

The given matrix will remain unchanged.

Parameters			**Description**
Field	**Name**	**Type**	
Input	a	`Matrix*`	The matrix of which the pseudoinverse is wanted
Output	pseudo_invers	`Matrix*`	The pseudoinverse of the matrix given

Exceptions	
Name	**Description**
`UndefinedMatrixException`	See Exceptions

Header

```
Matrix* pseudoinvers(Matrix* a);
```

Example

```
// a is a previous defined and populated matrix (pointer)
Matrix* pseudoinverse_mat;
Pseudoinverse_mat = a->pseudoinverse(a);
```

MatrixIrregular::determinant

Description

The method will throw an exception because determinant of an irregular matrix cannot be computed.

Parameters			**Description**
Field	**Name**	**Type**	

Exceptions	
Name	**Description**
`InvalidOperandException`	See Exceptions

Header

```
float determinant(Matrix* a);
```

MatrixIrregular::inverse

Description

The method computes the invers of a square matrix using the method presented in the reference [1].

Before computing the matrix, a copy of the given matrix is made so that it will remain unchanged.

Parameters			Description
Field	**Name**	**Type**	
Input	a	`Matrix*`	The matrix of which the invers is wanted
Output	invers	`Matrix*`	The invers of the matrix given

Exceptions	
Name	**Description**
`InvalidOperandException`	See Exceptions
`UndefinedMatrixException`	

Header

```
Matrix* invers(Matrix* a);
```

Example

```
// a is a previous defined and populated matrix (pointer)
Matrix* invers_mat;
invers_mat = a->invers(a);
```

MatrixIrregular::transpose

Description

The method will return the transpose of the given matrix by transposing each row to a column.

The original matrix will remain unchanged.

Parameters			Description
Field	**Name**	**Type**	
Output	transpose	`Matrix*`	The transpose of the matrix given

Exceptions	
Name	**Description**
`UndefinedMatrixException`	See Exceptions

Header

```
Matrix* transpose();
```

Example

```
// a is a previous defined and populated matrix (pointer)
Matrix* transpose_mat;
iranspose_mat = a->transpose();
```

MatrixIrregular::print

Description

The method overwrites the `print` method from the `InterfacePrintInOut` in order to print the matrix in the correct manner. A left and a right square bracket will be printed before and after the matrix.

Parameters			Description
Field	**Name**	**Type**	
Input	os	`std::ostream`	The stream to be outputted to the console

Exceptions	
Name	**Description**

Header

```
void print(std::ostream &os) const;
```

Example

```
MatrixIrregular a;
std::vector<std::vector<float>> mat_a{ { 6, 0, 5 },{ 3, 6, 3 },{ 0, 8, 1 } };
a.populate(mat_a);
std::cout << a << std::endl;
```

MatrixIrregular::populate

Description

The method will populate the matrix with the elements given as a parameter.

Parameters			Description
Field	**Name**	**Type**	
Input	_matrix	`std::vector<std::vector>`	The elements to be populated in the matrix

Exceptions	
Name	**Description**
`UndefinedMatrixException`	See Exceptions

Header

```
void populate(std::vector<std::vector<float>> _matrix);
```

Example

```
MatrixIrregular a;
std::vector<std::vector<float>> mat_a{ { 1, 3, 6 },{ 2, 1, 7 },{ 9, 2, 1 },{ 5, 2, 2 } };
a.populate(mat_a);
```

MatrixIrregular::decompose

Description

The method will return a new matrix from the one stored in the object. The matrix will have a specific dimension and the start point, in the stored matrix, is known (upper left corner).

The index of the starting point begin from 0.

Parameters			Description
Field	**Name**	**Type**	
Input	i	`int`	The row where the new matrix starts
	j	`int`	The column where the new matrix starts
	rows	`int`	The number of rows of the new matrix
	columns	`int`	The number of columns of the new matrix
Output	decomposition	`Matrix*`	The new matrix

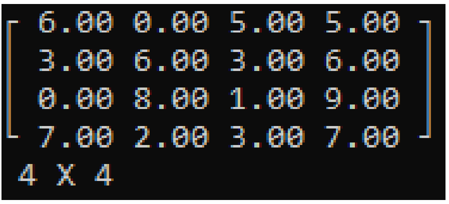
Exceptions	
Name	**Description**
`InvalidOperandException`	See Exceptions
`UndefinedMatrixException`	

Header

```
Matrix* decompose(int i, int j, int rows, int columns);
```

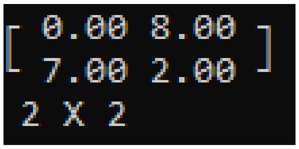
Example

```
MatrixIrregular a;
std::vector<std::vector<float>> mat_a{ { 6,0,5,5 },{ 3,6,3,6 },{ 0,8,1,9 },{ 7,2,3,7 } };
a.populate(mat_a);
std::cout << a << std::endl;
Matrix* decomposition;
decomposition = a.decompose(2, 0, 2, 2);
std::cout << *decomposition << std::endl;
```



```
┌ 6.00 0.00 5.00 5.00 ┐
│ 3.00 6.00 3.00 6.00 │
│ 0.00 8.00 1.00 9.00 │
└ 7.00 2.00 3.00 7.00 ┘
4 X 4
```

Figure 2 Initial matrix



```
┌ 0.00 8.00 ┐
└ 7.00 2.00 ┘
2 X 2
```

Figure 1 Decomposed matrix

MatrixIrregular::compose

Description

The method will merge four matrices of different size into a single matrix. The second one is placed to the right of the first one, the fourth to the right of the third and merge the two resulting matrices on top of each other.

Conditions:

1. First and second matrices must have the same number of rows
2. Third and fourth matrices must have the same number of rows
3. The sum of the columns of the first two matrices must be equal to the sum of columns of the third and fourth matrices

Parameters			**Description**
Field	**Name**	**Type**	
Input	list	`std::vector`	The list of the four matrices
Output	composition	`Matrix*`	The composed matrix

Exceptions	
Name	**Description**
`InvalidOperandException`	See Exceptions

Header

```
Matrix* compose(std::vector<Matrix*> list);
```

Example

```
std::vector<Matrix*> list;

Matrix *a = new MatrixIrregular();
Matrix *b = new MatrixIrregular();
Matrix *c = new MatrixIrregular();
Matrix *d = new MatrixIrregular();

std::vector<std::vector<float>> mat_a{ { 1,3 },{ 3,5 } };
std::vector<std::vector<float>> mat_b{ { 1,2,1 },{ 2,3,4 } };
std::vector<std::vector<float>> mat_c{ { 3 },{ 7 },{ 8 } };
std::vector<std::vector<float>> mat_d{ { 5,2,4,6 },{ 1,3,2,9 },{ 1,4,2,9 } };

a->populate(mat_a); b->populate(mat_b);
c->populate(mat_c); d->populate(mat_d);
list.push_back(a); list.push_back(b);
list.push_back(c); list.push_back(d);

Matrix* result;
try {
	result = a->compose(list);
	std::cout << *result << std::endl;
}
catch (const std::exception& e) {
	std::cout << e.what() << std::endl;
}
```

MatrixIrregular::border_matrix

Description

The method will border the first matrix with one or more matrices on the right of it or on the bottom..

The border can be:

1. To the right of the first matrix (`border_side: true`)
   a. The matrices must have the same number of rows
2. To the bottom of the first matrix (`border_side: false`)
   a. The matrices must have the same number of columns

Parameters			Description
Field	**Name**	**Type**	
Input	list	`std::vector`	The list of the matrices
	border_side	`bool`	The side on which the border should happen
Output	composition	`Matrix*`	The composed matrix

Exceptions	
Name	**Description**
`InvalidOperandException`	See Exceptions
`IncompatibleSizeException`	

Header

```
Matrix* border_matrix(std::vector<Matrix*> list, bool board_side);
```

Example

```
std::vector<Matrix*> list;

Matrix *a = new MatrixIrregular();
Matrix *b = new MatrixIrregular();
Matrix *c = new MatrixIrregular();
Matrix *d = new MatrixIrregular();

std::vector<std::vector<float>> mat_a{ { 1,2,1 },{ 2,3,4 },{ 1,2,4 } };
std::vector<std::vector<float>> mat_b{ { 3 },{ 7 },{ 8 } };
std::vector<std::vector<float>> mat_c{ { 5,2,4,6 },{ 1,3,2,9 },{ 1,4,2,9 } };
std::vector<std::vector<float>> mat_d{ { 2,5 },{ 8,9 },{ 2,3 } };

a->populate(mat_a);   b->populate(mat_b);
c->populate(mat_c);   d->populate(mat_d);

list.push_back(a);   list.push_back(b);
list.push_back(c);   list.push_back(d);

Matrix* result;
try {
       result = a->border_matrix(list, true);
       std::cout << *result << std::endl;
}
catch (const std::exception& e) {
       std::cout << e.what() << std::endl;
}
```

MatrixSquare header

MatrixSquare::invers

Description

The method will compute the invers of the given matrix.

See MatrixIrregular::invers for more details.

MatrixSquare::determinant

Description

The method will return the determinant of a square matrix by making all the entries below the main diagonal 0 using row operations.

Parameters			Description
Field	**Name**	**Type**	
Input	a	`Matrix*`	The matrix of which the determinant is wanted
Output	determinant	`float`	The determinant of the matrix

Exceptions	
Name	**Description**

Header

```
float determinant(Matrix* a);
```

Example

```
MatrixSquare a;
std::vector<std::vector<float>> mat_a{ { 1,2,1,8 },{ 2,3,4,6 },{ 1,2,4,7 },{ 3,1,7,5 } };
a.populate(mat_a);
std::cout << a.determinant(&a) << std::endl;
```

MatrixDiagonal header

MatrixDiagonal::invers

Description

In comparison with the MatrixIrregular::invers this method is a particular case and the invers is computed significantly faster because the invers is also a diagonal matrix.

Parameters			Description
Field	Name	Type	
Input	a	`Matrix*`	The matrix of which the determinant is wanted
Output	determinant	`float`	The determinant of the matrix

Exceptions	
Name	Description
`InvalidOperandException`	See Exceptions

Header

```
Matrix* invers(Matrix * a);
```

Example

```
MatrixDiagonal a;
std::vector<std::vector<float>> mat_a{ { 1,0,0,0 },{ 0,3,0,0 },{ 0,0,4,0 },{ 0,0,0,5 } };
a.populate(mat_a);
std::cout << *a.invers(&a) << std::endl;
```

MatrixDiagonal::determinant

Description

The method will return the determinant of a square matrix by multiplying all the entries of the main diagonal.

Parameters			Description
Field	Name	Type	
Input	a	`Matrix*`	The matrix of which the determinant is wanted
Output	determinant	`float`	The determinant of the matrix

Exceptions	
Name	Description

Header

```
float determinant(Matrix* a);
```

Example

```
MatrixDiagonal a;
std::vector<std::vector<float>> mat_a{ { 1,0,0,0 },{ 0,3,0,0 },{ 0,0,4,0 },{ 0,0,0,5 } };
a.populate(mat_a);
std::cout << a.determinant(&a) << std::endl;
```

Other headers

When it comes to the `MatrixLine` and `MatrixColumn`, these headers implement simple classes that override ether the populate method or the printing method.

PrintingMacros header

In this header are defined macros for the different ASCII characters that will be printed.

The macros `PRINTING_PADDING_ELEMENT` and `PRINTING_PRECISION` used when a matrix is printed alone and dictate the total number of characters that each entry from the matrix will have when printed while the second macro will set the precision of the entry when is printed.

Printing a matrix equation

The driver provides a way to visualize a simple matrix equation and supports the following operations: addition, subtraction, multiplication, and invers.

No operations are being made, the module only prints the equation depending on the operands and operations.

There are several components that are used in printing the equation:

- `matrix_to_print` – the matrix that will be printed at the end of the process
- `aux_print_matrix` – an auxiliary matrix in which each operand is placed
- `list_operands` – the list of operands that need to be printed
- `operations` – the list of operations that are made

The algorithm will go through the `operations` and add the corresponding operand and operation to the `aux_print_matrix` first and then to the final `matrix_to_print`. A flow diagram is presented below that show the main principal on how the algorithm will create the whole equation for printing. Keep in mind that the diagram may differ from the actual implementation.

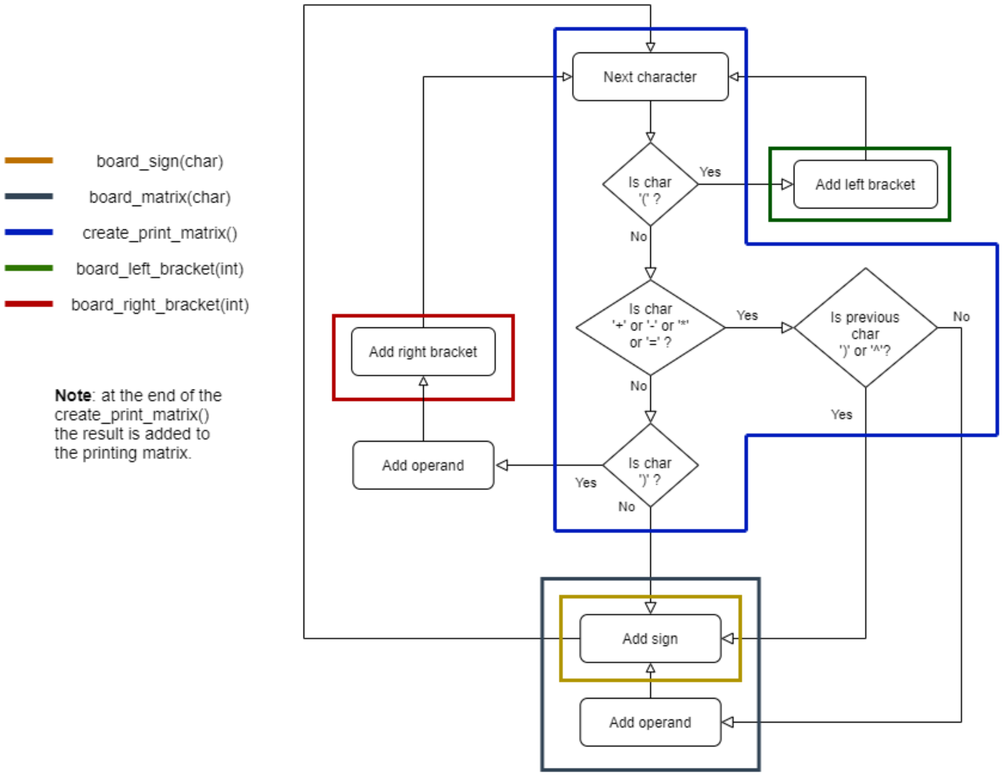


Figure 3 Flow diagram of the printing algorithm

Each method represents a certain block from the diagram.

The equation has a limited number of rows and columns that it can display, a macro (`MAX_PRINTING_ROW_SIZE`) has been defined and can be change in order to display more rows and columns.

PrintingEquation header

PrintingEquation::create_print_matrix

Description

The method will go through the operands and decide how the operand and operation will be added to the `aux_print_matrix` and then to the `matrix_to_print`.

Parameters			Description
Field	Name	Type	

Header

```
void create_print_matrix();
```

Example

```
std::vector<Matrix*> operands;
std::string operations = "*(*)^=";

Matrix *a = new MatrixIrregular();
a->populate(read_matrix());
std::cout << *a << std::endl;

operands.push_back(a->transpose());
operands.push_back(a);
operands.push_back(a->transpose());
operands.push_back(a->pseudoinvers(a));

PrintingEquation pq(operands, operations);
pq.create_print_matrix();
```

PrintingEquation::board_sign

Description

The method will add specific characters to the `aux_print_matrix` to represent the operation that wants to be added.

See PrintingEquation::print_borded_matrix for the exact characters with the exception of the character '#' which means that no sign should be added to `aux_print_matrix`.

Parameters			Description
Field	Name	Type	
Input	sign	`char`	The sign that has to be boarder to the `aux_print_matrix`

Header

```
void board_sign(char sign);
```

PrintingEquation::board_right_bracket

Description

The method will add specific characters to the `aux_print_matrix` to represent a left bracket(‘(’).

The same description and parameter can be said about the `PrintingEquation::board_right_bracket`

Parameters			Description
Field	Name	Type	
Input	size	`int`	The size of the bracket that has to be added

Header

```
void board_left_bracket(int size);
void board_right_bracket(int size);
```

PrintingEquation::print_borded_matrix

Description

The method will translate the specific characters that were added by the other methods and print the correct ASCII character to the console.

Character translation:

Square brackets characters

? → ┌   ┐ → %

@ → │   │ → @

$ → └   ┘ → !

Sign characters

$p \rightarrow +$

$m \rightarrow -$

$i \rightarrow *$

$e \rightarrow =$

Parameters			Description
Field	Name	Type	
Input	size	`int`	The size of the bracket that has to be added

Header

```
void print_borded_matrix(std::vector<std::string> &matrix);
```

PrintingEquation::board_matrix

Description

The method will add the operand, the matrix, with its brackets and if needed a sign.

The elements are added to aux_matrix_order in the following order:

1. a left bracket ('(')
2. the actual matrix
3. a right bracket (')')
4. a sign (if specified)

Both the left and right bracket are the same size as the matrix.

A macro has been defined (`MAX_PRINTING_ROW_SIZE`) that indicates the maximum number of rows and columns that can be displayed. If the matrix has one of the dimensions bigger that the macro a single dot, for columns, will be added or a line of dots for the rows.

Parameters			Description
Field	**Name**	**Type**	
Input	sign	`char`	The sign that has to be boarder to the `aux_print_matrix` besides the operand (matrix)

Header

```
void board_matrix(char sign);
```

Note: The correct number of operands with respect to the operations need to be given to the method PrintingEquation::create_print_matrix or else the printing will fail

References

[1] Ahmad Farooq, Khan Hamid, 'An Efficient and Simple Algorithm for Matrix Inversion', https://www.researchgate.net/publication/220337322_An_Efficient_and_Simple_Algorithm_for_Matrix_Inversion