Linear Algebra

A simple library for Linear Algebra with C++.

Have you interest in? Mail me!

Class Matrix

Example

```
#include <iostream>
#include "./matrix.cpp"
using namespace std;

int main()
{

  Matrix::MATRIX_VERBOSE = false;     //shows some useful informations when
true
  Matrix::MATRIX_PRINT_PRECISION = 4; //the precision of numbers when
printed
  int i, j;
  Matrix mymatrix(3, 5), secondmatrix(3, 5); //declaring my matrix objects

  for (i = 0; i < 3; i++)
  {
    for (j = 0; j < 3; j++)
    {
      mymatrix.set(i, j, (double)i * j + i - j);
      secondmatrix.set(i, j, (double)i * j * j - i);
    }
  }
  mymatrix.set(1, 1, 3.1415926535); //setting values into the object

  //"cout" prints a matrix
  cout << mymatrix;     // printing the matrix
  cout << mymatrix.T(); // transposing the matrix

  //operations with scalar
  cout << mymatrix + 5; //summing a scalar value
  cout << 5 + mymatrix; //summing a scalar value
  cout << mymatrix - 5; //subtracting a scalar value
  cout << 5 - mymatrix; //subtracted by scalar value
  cout << mymatrix * 5; //multiplying by a scalar value
  cout << 5 * mymatrix; //multiplying by a scalar value
  cout << mymatrix / 5; //dividing by a scalar value
  cout << 5 / mymatrix; //dividing a scalar value

  //operations with matrices "entry-to-entry"
```

```
    cout << mymatrix + secondmatrix; //summing a matrix
    cout << mymatrix - secondmatrix; //subtracting a matrix
    cout << mymatrix * secondmatrix; //multiplying by a matrix
    cout << mymatrix / secondmatrix; //dividing by a matrix

    //matrix multiplication of type "row dot column" with the "%" operator
    cout << mymatrix % secondmatrix.T(); //matrix multiplication with "%"

    //changing the sign of the matrix entries
    cout << -mymatrix;

    //All the boolean operators "!=, ==, >=, <=, >, <" are implemented
    //as examples, see below
    cout << (mymatrix >= secondmatrix); //1 when true, 0 when false
    cout << (4 >= mymatrix);
    cout << (mymatrix >= 4);
    return 0;
}
```

Indexing

I implemented the same indexing as in the Python language. For instance, let us declare a matrix `A(3,3)`. To get the **last row**, you need just to access the `(-1)` index as well as the **last column**.

```
int m=3, n=6;
Matrix A(m, n);
cout << A(-1,-1); //prints the (last row, last column) entry: (2,5)
cout << A(-2,-2); //prints the (last-1 row, last-1 column) entry: (1,4)
cout << A(-3,-2); //ERROR: you must not access the "-m" index or less
//the access range allowed is [from (-m+1) until (m-1)]
```

To clarify what I just explained, see the allowed access:

- declaring a matrix as A(1, 5).

column indices	**[0,**	**1,**	**2,**	**3,**	**4]**
access allowed	[0,	1,	2,	3,	4]
access allowed	[0,	-4,	-3,	-2,	-1]
access denied	[**-5**,	-4,	-3,	-2,	-1]

Transaction Interface

To get values from or to set values into the matrix, use the `.get()` and `.set()` methods.

```
Matrix my(3,3); //creating a zero matrix
cout << my.get(2, 2); //prints the (3rd, 3rd) entry of the matrix
//remember the indexing starts from zero!
my.set(2, 2, 3.1415); // setting the PI number to the (3rd, 3rd) entry
cout << my; // prints the matrix with the PI number
```

Functions

- `eye(int m, int n, int offset=0)`

  > it returns an identity matrix, with the main diagonal of ones when `offset==0`; upper diagonal when `offset>=1`; lower diagonal when `offset<=-1`.

- `eye(int n)`

  > it returns a square identity matrix (`offset=0`).

Methods

- `this.get(int row, int col)`

  > returns the (row-th, col-th) entry of the matrix.

- `this.set(int row, int col, double value)`

  > insert the value into the (row-th, col-th) entry of the matrix.