Project 2
SF2568 Program construction in C++ for Scientific Computing

Aron Andersson aronande@kth.se Harsha HN harshahn@kth.se

Task 1 - Implement the evaluation of the exponential for real numbers.

The exponential of real numbers can be expressed as a simpler version of Taylor series expansion. Our goal is to determine the value for a real number given the error tolerance. Firstly, we need to determine the number of terms required to achieve the required tolerance. So, we apply the rule that if the term *n* is smaller than the error tolerance, then the sum of remaining consecutive terms is smaller than term *n* and the error. Secondly, Horner's scheme has been used to improve the performance. To validate our implementation, we have chosen set of values for real numbers and error tolerance. Error is computed against the function *exp* provided by the standard library *cmath.*

Tol vs. Value	-1	1	5	10	50
0.01	0.00121277	0.00161516	0.00294883	0.00175855	3.14573e+06
0.001	2.22983e-05	2.78602e-05	0.00020803	0.000162384	3.14573e+06
1e-8	1.49839e-10	1.72876e-10	8.31676e-10	9.24047e-10	3.14573e+06
1e-10	7.19647e-13	8.15348e-13	4.17799e-12	1.45519e-11	3.14573e+06

Table 1: Absolute difference between cmath and myexp implementation over a range of *x* (real number) and tolerance.

As can be seen in Table 1 the observed error is within the set tolerance for low *x* values; however, for larger values such as *50* the precision is compromised. Error variable is of type *double* which is a 64 bit floating point with 52 bits mantissa i.e., can store 15 digits of precision. Thus, accuracy upto 15 digits are retained and for values such as *myexp(50)* yields in *e+21* we observe error of e+06 caused by 16th digit onwards.

Task 2 - Construct a matrix class and compute matrix exponential.

The task is to create a matrix A, and then compute the exponential of that matrix, that is *exp(A)*.
We are given this skeleton to follow,

class Matrix {
Public:

Matrix(int m);
Matrix(const Matrix&);
Matrix& operator=(const Matrix&);

```
        Matrix& operator+=(const Matrix&);
        Matrix& operator*=(const Matrix&);
        Matrix& operator*=(const double);
        double norm() const;
        void printMatrix() const;
        void fillMatrix(....);
        ...
};
```

For this task we created three files, *task2MatrixProjectmain.cpp, task2Project.hpp* and *task2Project2.cpp.*

The idea in *task2MatrixProjectmain.cpp* is to let the user create several *mxm* matrices of whichever size he prefers. We have created three matrices of sizes 3x3, 2x2 and 6x6 with random inputs that will result in both big and small exponential values.

#include <vector> enables the user to write matrices in a neat and obvious style of columns and rows. This way of writing the matrice makes it easy to spot a mistake by eye. When all the matrices are created they are put in a library list and run through a *for*-loop. We use an included feature of -<vector> package, *size()*, to get the size of every library matrix we have created.

First line in the for loop, *Matrix mat(element.size()),* will call constructor *Matrix(int m)* and allocate memory for a *pointer-to-a-pointer-to-a-double* list of the size of our matrix. We are aware that writing this in single *C-style array* makes the code run faster, but we chose to do it like this because that is what we were taught in *Lab1* first week of the course.

Now that the memory is allocated, we can fill the *pointer-to-a-pointer-to-a-double* list with the elements of that is in out matrix. This is made by the *fillMatrix()* function that belongs to the *Matrix* class. This function will generate an error message if the user has failed to create an *mxm* matrix as it will go through each row of the library list and check that it corresponds to the size generated by the built-in library function *size()*. For an example, the mistake

```
{{3,2,4},
  {2,3},
  {5,6,3}}
```

or

```
{{3,2},
 {2,3},
 {5,6}}
```

will generate the error message, as they are not *mxm* matrices. The program will stop and go back to the terminal.

To have something to compare our generated *exp(A)* with, we were given the file *8mat_expm1.cpp* which implements *Matlab*'s algorithm for *expm*.

The calling sequence is:

double result = r8mat_expm1(int m, double a[])*

Here, *m* is the dimension of the matrix and *a* denotes a C-style array where the matrix is stored in Fortran-style (column wise). The returned value result is a pointer to a C-style array holding the matrix exponential in Fortran style.

As this takes a C-style array but we have chosen to use the C++ standard library, <vector>, to build our matrices, we have to turn our matrix into a C-style array. To do so we create the function,

getListFromMatrix(a, element, size)

Here *a* is a *pointer-to-a-double* list, the pointer points to an *m by m* size double list, *element* is our created vector<vector<int>> matrix, *size* is the integer m.

This function will generate pointer *a* that points to a created C-style array with the same elements of our vector<vector<int>> matrix. This *a* is then put into the function,

r8mat_expm1(size, a)

which generates the matrix exponential. The elements can be printed out using a simple for loop.

The function, *exp(double* expm, double tol)*, turns our matrix object into an exponential matrix. The idea of this function is that the elements of our exponential matrix will continuously be compared, for every added term in *equation 1*, to what we know is the correct exponential matrix created by *r8mat_expm1(size, a).*

When creating our exponential of a matrix, we do not want to change our original matrix, A, as this matrix will in the while-loop, in *exp(double* expm, double tol),* be used to multiplicate previous A^i with A. So A has to be copied so it can be changed to A^i, where i is the number of the term we are at in *equation 1* when looping.. The copying is made by calling the copy constructor *Matrix(const Matrix& Other).*

Matrix object *resultMatrix* is our exponential matrix that will for every loop in *exp(double* expm, double tol)* come closer to the exponential matrix from *r8mat_expm1(size, a)*. By examining,

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = \frac{A^0}{0!} + \frac{A^1}{1!} + \ldots \qquad (1)$$

We see that the first term is the identity matrix and the second term is just our regular A matrix. Therefore, before we enter the while-loop these two terms are added together with the help of the assignment *operator+()*.

The function *identity()* is creating an identity matrix of size *mxm*

Matrix object *subMatrix* will be used for tolerance check in the while loop. It takes the most recent exponential matrix and subtracts the elements with elements of the the correct exponential matrix created from *r8mat_expm1(size, a),* as we here refer to with pointer *ptr.* It can do so by using the added *subtraction()* function. Then we use the *norm()* function to calculate the norm-2 as is suggested in the paper *Nineteen dubious ways to compute the exponential of a matrix, twenty five years later.* As it takes a lot of coding to calculate the norm in the norm-2 way, we downloaded a package *Eigen 3.3.7*. With the help of the function *eigenvalues(),* the eigenvalues of the transpose matrix could then be easily calculated.

To implement this package, we did the following steps.
1. Downloaded the package from www.eigen.tuxfamily.org
2. To run it with GNU we typed in the terminal,

```
g++ -Wall -I /Users/HP/Downloads/eigen-eigen-323c052e1731/eigen-eigen-323c052e1731/ task2MatrixProject2main.cpp task2Project2.cpp r8lib.cpp r8mat_expm1.cpp -o output
```

The matrix object *facMatrix* is, for every loop in *exp(double* expm, double tol)* , the latest term of *equation 1.*

So in summary, we have additionally to the functions given to us in the skeleton, also added the following functions,

```
void identity();
void subtraction(double* expA);
Matrix exp(double* ptr, double tolerance) const;
void getListFromMatrix(double* a, std::vector<std::vector<int>> matrix, int sizeOfMatrix);
```

The matrices we used are,

```
         {2 0 1                        {5 6
 A =  -1 1 0                 B =   4 2}
-         3 3 0}

         {5 4 4 3 3 7
           0 2 2 3 4 1
  C =   9 8 3 2 2 2
          0 2 2 3 4 1
          0 2 2 3 4 1
          0 2 2 3 4 1}
```

Calculating the exponential of a matrix when implementing
Matlab's algorithm for *expm* is,

```
The expm(A) =
{0.954389  3.25803 1.60621
-2.69222    1.5202 -1.08601
-8.07665    1.5606 -2.25803}

The expm(B)  =
{3594.36 3255.84
2170.59 1966.47}

expm(C) =
{1.72794e+6  2.74494e+6  2.08739e+6  2.4757e+6   3.10438e+6  1.65379e+6
653561       1.03825e+6  789530      936414      1.17421e+6  625510
1.91898e+6   3.0484e+6   2.31815e+6  2.74938e+6  3.44756e+6  1.83664e+6
653561       1.03825e+6  789530      936415      1.17421e+6  625510
653561       1.03825e+6  789530      936414      1.17422e+6  625510
653561       1.03825e+6  789530      936414      1.17421e+6  625511}
```

Now, using our function exp(double* ptr, double tolerance) with tolerance 0.1, we get

```
exp(A) =
{0.893056  3.30417  1.59861
 -2.7       1.49722  -1.10139
 -8.1       1.49167  -2.30417}

exp(B) =
{3594.36  3255.84
  2170.56  1966.44}

exp(C) =
{1.72794e+6  2.74494e+6  2.08739e+6  2.4757e+6   3.10438e+6  1.65379e+6
653561       1.03825e+6  789530      936414      1.17421e+6  625510
1.91898e+6   3.0484e+6   2.31815e+6  2.74938e+6  3.44756e+6  1.83664e+6
653561       1.03825e+6  789530      936415      1.17421e+6  625510
653561       1.03825e+6  789530      936414      1.17422e+6  625510
653561       1.03825e+6  789530      936414      1.17421e+6  625511}
```

References:

1. Nineteen dubious ways to compute the exponential of a matrix, twenty five years later. SIAM Review 45(2003)1, 3–49