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

Caroline Eriksson Sophie Malmliden

October 2019

Task 1

The goal of this part of the exercise is to implement the evaluation of the exponential function for real numbers. The exponential function has the following series representation:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + ...$$

The calling sequence to be used is following:

```
double myexp(double x, double tol = 1e-10);
```

The parameter *tol* indicates the required accuracy. The routine is to be checked against the exponential function from the standard library.

It will not be possible to compute the exponential function using an infinite amount of terms from the series representation of the exponential function. Therefore an upper limit N, for the amount of terms included in the evaluation will be needed. Let us denote the approximation of the exponential function, e^x, with S_N:

$$S_N = \sum_{n=0}^{N} \frac{x^n}{n!} = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + ... + \frac{x^N}{N!}$$

For the approximation to satisfy the required accuracy, the error term, t_{N+1}, for the approximation, S_N, has to be smaller than the tolerance:

$$abs(t_{N+1}) = abs\Big(\frac{x^{N+1}}{(N+1)!}\Big) < tol = 1e-10$$

The evaluation is implemented using polynomials and created through a for loop. In the code we will let S be the variable for S_n where n will change from 0 to the number of terms N that is needed to satisfy the required accuracy and t will be the variable for t_{n+1} where n goes from 0 to N, where N is the number of terms from the series needed for the approximation to fulfill the required accuracy.

The code starts off with assigning the variable S the value 1 and the variable t with the value x. This will correspond to the approximation value S_0 and the error term t_{0+1}, i.e. here corresponding to $n = 0$. Afterwards in the code, the for loop is to be found. Each round in the loop correspond to a different n, starting off with $n = 1$, computing the corresponding S_n and t_{n+1} values. In each round it will be checked wether the approximation is accurate enough by checking if the error term satisfies the accuracy or not.

As long as the required accuracy is not fulfilled the for loop will do a new round where n will be the integer one larger than it was in the loop round just before. The variable S_{n+1} is computed using the error term from the previous loop added to S_n from the previous loop, and the the error term t_{n+2} is computed multiplying t_{n+1} from the previous loop with the fraction $\frac{x}{n+2}$:

$$S_{n+1} = S_n + \frac{x^{n+1}}{(n+1)!}, \quad t_{n+2} = t_{n+1}\frac{x}{n+2}$$

If the required accuracy is fulfilled, the for loop will be stopped and the desired approximation is found. For further details, see Appendix 1: C++ code for Task 1.

To explore the relationship between the input value x and the number of terms, N, from the series respresentation for e^x, several values for x are tested. See in the table below. From this table one can observe that the number of terms, N, required to achieve the requested tolerance seem to increase for an increasing value of the absolute value of x. Further the routine is to be validated against the exponential function from the standard library. Using the data type double, the same function values will be returned from our routine as from the exponential function from the standard library. See table 1 below.

Table 1. Relation between input values x and terms N from the series representation for e^x

x	-3	-1	0	1	3	5	8	10	20
N	21	13	1	13	21	28	37	43	71
S_N	0.0497871	0.367879	1	2.71828	20.0855	148.413	2980.96	22026.5	4.85165e+08
e^x	0.0497871	0.367879	1	2.71828	20.0855	148.413	2980.96	22026.5	4.85165e+08

Task 2

The goal of this task is to compute the matrix exponential using the series:

$$exp(A) = \sum_{n=0}^{\infty} \frac{A^n}{n!} = \frac{A^0}{0!} + \frac{A^1}{1!} + \frac{A^2}{2!} + \frac{A^3}{3!} + \frac{A^4}{4!} + \frac{A^5}{5!} + ...$$

For this task a general class for matrices is to be implemented in order to provide all necessary functionality for computing elementary functions for matrices. These functions will also be needed for the evaluation of the matrix exponential. The skeleton for the matrix class is to be constructed as follows:

```
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(....);
          ...
};
```

This skeleton for the matrix class is placed in a header file and the functions as well as the main part of the code are constructed in a main file. For the matrix operation functions as well as further details, see Appendix 2: Header file for Task 2 and Appendix 3: Main code for Task 2.

The evaluation of the matrix exponential is implemented using polynomials and a for loop, with the same structure as in Task 1:

Let us denote the approximation of the matrix exponential, e^A, with s_N:

$$s_N = \sum_{n=0}^{N} \frac{A^n}{n!} = \frac{A^0}{0!} + \frac{A^1}{1!} + \frac{A^2}{2!} + \frac{A^3}{3!} + \frac{A^4}{4!} + \frac{A^5}{5!} + ... + \frac{A^N}{N!}$$

For the approximation to satisfy the required accuracy, the error term, t_{N+1}, for the approximation, s_N, has to be smaller than the tolerance. The error term t_{N+1}, will for Task 2 always be a matrix, and to compare this matrix with the accuracy, we will here use the matrix norm operator. The matrix norm chosen for our matrix norm operator is the following. Where A is the square matrix, m is the size of the matrix, j is the variable for iterating the column values for each row i.

$$\|A\|_\infty = \max_{1 \leq i \leq m} \sum_{j=1}^{m} |a_{ij}|$$

Hence the accuracy requirement can be written:

$$\|t_{N+1}\|_\infty = \|\frac{A^{N+1}}{(N+1)!}\|_\infty < tol = 1e-10$$

The evaluation is implemented using polynomials and created through a for loop. In the code we will let s be the notation for the matrix s_n where n will change from 0 to the number of terms N that is needed to satisfy the required accuracy and t will be the notation for the matrix t_{n+1} where n goes from 0 to N, where N is the number of terms from the series needed for the approximation to fulfill the required accuracy.

The code starts off with defining s as the identity matrix and t as the matrix A. This will correspond to the approximation value s_0 and the error term t_{0+1}, i.e. here corresponding to $n = 0$, i.e. here the zeroth term from the representation series. Afterwards in the code, the for loop is to be found. Each round in the loop correspond to a different n, starting off with $n = 1$, computing the corresponding s_n and t_{n+1} values. In each round it will be checked wether the approximation is accurate enough by checking if the error term satisfies the accuracy or not.

The different matrices used to test the matrix exponential are:

$$B = \begin{bmatrix} 5 & 4 \\ 2 & 6 \end{bmatrix}, C = \begin{bmatrix} 12 & -7 \\ -3 & 18 \end{bmatrix}, D = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 6 & 4 \\ 1 & 4 & 2 \end{bmatrix}, E = \begin{bmatrix} 1.3 & 2.4 & 5.1 \\ 6.7 & 4.3 & 8.5 \\ 2.6 & 9.2 & 6.8 \end{bmatrix}$$

The relation between the input matrices, the norm of the error term, $\|t_{N+1}\|_\infty$, the error estimation ϵ and the number of terms N from the series representation for e^A required are presented in table 2 below. The variable ϵ represents the difference between the approximated matrix exponential, s_N, and the matrix exponential from the given routine, R, which implements Matlab's algorithm for expm. ϵ is computed as:

$$\epsilon = abs(\|s_N\|_\infty - \|R\|_\infty)$$

Table 2. The results for the different matrices used in the series representation.				
$Matrix$	B	C	D	E
N	39	73	43	63
$\|t_{N+1}\|_\infty$	5.3255e-11	5.9080e-11	7.7877e-11	6.6107e-11
ϵ	6.7303e-11	1.9670e-6	1.4552e-11	1.1548e-7

Discussion

From table 2 one can see that the approximation of the matrix exponential is able to handle all of the four different matrices tested. It is also apparent that more terms are required in the sum in order for the accuracy condition to be achieved when having negative numbers or floats in the matrices, as in matrix C and E. It is also worth noting that the resulting difference between the approximation and the given routine is larger for the aforementioned matrices. It would also be interesting to compute other types of errors and see how the result differs for different matrices, for example one could extend this task by computing the error in the following way:

$$\epsilon_2 = \|s_N - R\|_\infty$$

It would also be interesting to se how N, $\|t_{N+1}\|_\infty$, ϵ and ϵ_2 would be affected for a larger amount of different matrices with different properties.

Appendix 1: C++ code for Task 1

```
#include <iostream>
#include <cmath>
#include <math.h>
#include <vector>
using namespace std;

// declaration of functions before use
double myexp(double, double);

// Creating function myexp(x, tol) that calculates the approximation of the
// exponential function using the ampunt of terms from the Maclaurin series
// that lets the approximation satisfy the required accuracy.
double myexp(double x, double tol)
{
    double S = 1; //define first term from series
    double t = x; //define first error term

    for (int i = 1; true; i++){

        S += t; //adding new terms from Maclaurin series
        t *= x/(i+1); //computing updated error term

        if (abs(t) < tol) {

            cout << "The number of terms required: " << i << endl;
            break; //checking required accuracy
        }
    }
    return S;
}

int main()
{
        //declaration of variable before use, here choose input
        //variable for the approximations
        double x = 10;
        double tol = 1e-10;
        //printing out the exact and approximate values and corresponding errors
        cout << "The input value x is: " << x << endl;
        double F = myexp(x, tol);
        cout << "The approximated exponential from our routine: " << F << endl;
        double e = exp(x);
        cout << "The exact exponential: " << e << endl;
        return 1;

}
```

Appendix 2: Header file for Task 2

```
#ifndef Matrix_h
#define Matrix_h
#include <iostream>

class Matrix {

public:
        int m;
        double** Mat;
        int* K;
        Matrix(int m);
        Matrix(const Matrix&);
        ~Matrix();
        Matrix& operator=(const Matrix&);
        Matrix& operator+=(const Matrix&);
        Matrix& operator*=(const Matrix&);
        Matrix& operator*=(const double);
        Matrix operator*(const double&) const;
        double norm() const;
        void printMatrix() const;
        void fillMatrix(const int, const Matrix& Q);

};

#endif
```

Appendix 3: Main code for Task 2

```
#include "Klass.h"
#include <iostream>
#include "r8lib.h"
#include <vector>
#include "r8mat_expm1.cpp"
#include "r8mat_expm1.h"
#include "r8lib.cpp"
#include <cmath>
#include <math.h>
using namespace std;



Matrix::Matrix(int s)   // Constructor of class Matrix.
{
        m = s;
        Mat = new double* [m];

        for (int i = 0; i < m; i++)
        {
                Mat[i] = new double[m]();
        // For each row i every column value shall be set to 0.
        }
}

Matrix::Matrix(const Matrix &q)          // Copy constructor.
{
        m = q.m;
        Mat = new double* [q.m];
        for (int i = 0; i < q.m; i++)
        {
                Mat[i] = new double[q.m];
        }
        for (int i = 0; i < q.m; i++) {
                for (int j = 0; j < q.m; j++) {
                        Mat[i][j] = q.Mat[i][j];
                }
        }
}


Matrix::~Matrix()        // Destructor.
{
        for (int i = 0; i < m; i++){
                delete[] Mat[i];
        }
        delete[] Mat;
}


Matrix& Matrix::operator=(const Matrix& q)       // Overload operator.
{
        if (this == &q)
                return(*this);
```

```
        delete[] Mat;
        this->m = q.m;
        this->Mat = new double* [m];
        int ol = 0;

        for (int i = 0; i < m; i++)
        {
                this->Mat[i] = new double[m];
                ol++;
                for (int j = 0; j < m; j++)
                {
                        this->Mat[i][j] = q.Mat[i][j];
                }
        }
        return(*this);
}

Matrix& Matrix::operator+=(const Matrix& Q)
{
        for (int i = 0; i < m; i++)
        {
                for (int j = 0; j < m; j++)
                {
                        this->Mat[i][j] += Q.Mat[i][j];

                }
        }
        return (*this);
}


Matrix& Matrix::operator*=(const Matrix& Q)
{
        Matrix temp(Q.m);
        for (int i = 0; i < temp.m; ++i) {
           for (int j = 0; j < temp.m; ++j) {
                for (int k = 0; k < m; ++k) {
                   temp.Mat[i][j] += (Mat[i][k] * Q.Mat[k][j]);
                }
           }
        }
        return (*this = temp);
}

Matrix& Matrix::operator*=(const double num)
{
        for (int i = 0; i < m; ++i) {
                for (int j = 0; j < m; ++j) {
                        this->Mat[i][j] *= num;
                }
        }
        return *this;
}


Matrix Matrix::operator*(const double& num)const {
        Matrix temp(m);
```

```
        for (int i = 0; i < m; i++) {
                for (int j = 0; j < m; j++) {
                        temp.Mat[i][j] = Mat[i][j] * num;
                }
        }
        return temp;
}

double norm(const Matrix& Q)     // Computes the norm of the matrices.
{
        double *K = new double[Q.m]();
        // Creating the vector K which will consist
        // of each sum of the values in each row.
        double max;

        for (int i = 0; i < Q.m; i++) {
                for (int j = 0; j < Q.m; j++) {
                        K[i] += Q.Mat[i][j];
                }
        }
        for (int i = 0; i < Q.m; i++) {
        // Determining the maximum value in the vector K.
                if (K[0] < K[i]) {
                        K[0] = K[i];
                }
        }

        return K[0];
}

void printMatrix(const Matrix& Q) {
        cout << "\nThe entered matrix is: " << endl;
        for (int i = 0; i < Q.m; i++) {
                for (int j = 0; j < Q.m; j++) {
                        cout << Q.Mat[i][j] << ' ';
                }
                cout << ' ' << endl;
        }
        //cout << "\n" << endl;
}

void fillMatrix(const int m, const Matrix& Q) {
        for (int i = 0; i < m; i++) {
                for (int j = 0; j < m; j++) {
                        cout << "Enter matrix value for row number "
                        << i << " and column number " << j << ": ";
                        cin >> Q.Mat[i][j];
                }
        }
}


int main()
{
        int m = 2;
        double tol = 1e-10;
        Matrix M(m);
```

```
        fillMatrix(m, M);
        printMatrix(M);

        // Creating the identity matrix.
        Matrix s(m);
        for (int i = 0; i < m; i++) {
                for (int j = 0; j < m; j++) {
                        if (j == i) {
                                s.Mat[i][j] = 1;
                        }
                }
        }

        //  Creating the vector to be used as input in the r8mat_expm1 function.
        int asize = m * m;
        double* a;
        a = new double[asize];
        int aelement = 0;

        for (int i = 0; i < m; i++) {
                for (int j = 0; j < m; j++) {
                        a[aelement] = M.Mat[j][i];
                        aelement += 1;
                }
        }

        // Creating a matrix out of the result vector.
        double* result = r8mat_expm1(m, a);
        Matrix R(m);
        int relement = 0;
        for (int i = 0; i < m; i++) {
                for (int j = 0; j < m; j++) {
                        R.Mat[j][i] = result[relement];
                        relement += 1;
                }
        }

        // Calculation of the matrix exponential.
        Matrix t = M;

        for (double i = 1.0; i < 1000; i++) {
                s += t;
                double c = 1.0 / (i + 1.0);
                t *= (M * c);
                if (norm(t) < tol) {
                        cout << "\nThe number of terms required in the sum are: " << endl;
                        cout << i + 1 << endl;
                        cout << "\nThe norm of the error term is: " << endl;
                        cout << norm(t) << endl;
                        cout << "\nThe difference between the norm of the approximation
                        and the norm of the given routine is: " << endl;
                        cout << abs(norm(s) - norm(R)) << endl;
                        break;
                }
        }
        return 0;
}
```

References

The references for this Project is following internet sources:
https://codereview.stackexchange.com/questions/149669/c-operator-overloading-for-matrix-operations-follow-up http://www.cplusplus.com/forum/general/69897/ https://github.com/akalicki/matrix/blob/master/dist/matrix.cpp https://www.programiz.com/cpp-programming/examples/array-largest-element