Computational Physics - Exercise 1: Introdution

Torben Steegmann Chih-Chun Kuo

Introduction

In this first exercise we get comfortable working with matrices and plotting graphs.

In the first part we create different matrices and perform small operations on them.

In the second part we evaluate the 'Chebyshev polynomial' of different degrees and plot the results in a graph.

Simulation model and method

As this exercise serves the purpose of 'warming up' we do not use a specific simulation model or method. Instead, we solve a few smaller tasks in C++:

Exercise 1:

1. Create a 6×6 matrix 'A' with random, but seeded, numbers:
   This we achieve by first using the standard libraries std::srand(wxyz) function where wxyz are the last four digits of out matricle number. In our case that is std::srand(5378). We then write a struct Matrix with a constructor that initializes the cells with random number with the std::rand() function.
2. Search for the largest element and its index in the matrix:
   For this task we create another struct 'index' which we use to keep track of the index of this largest element. As the values of the elements in the matrix never change in this exercise, we can already look for the largest element in the constructor, avoiding extra calculation later. We safe its index so we can look it up and output it later in the program.

3. Create a row vector consisting of the largest elements of each column and a column vector consisting of the largest elements in each row of the matrix. What is the result of their multiplication?:
   As mentioned, the values of the matrix will not change after creation. Similarly to finding the largest element we can also look for the largest element of each row and column and safe them in vectors. We can also already multiply them and store the result for output late in the program.

4. Create a second random 6x6 matrix B with elements having values between −5.0 and 5.0 and calculate $C = A * B$ and $D = B * A$:
   To create B we can reuse the constructor for A. For C and D we overload the matrix constructor, using two matrices as its parameters. We multiply those together and store the result. Then we only need to call this constructor twice, using A and B as the parameter in different orders each time.

Exercise 2: Chebyshev polynomials

The Chebyshev polynomials $T_n(x) = \cos(n \arccos(x)), -1 \leq x \leq 1$ can be recursively defined by:

$$T_0(x) = 1$$
$$T_1(x) = x$$
$$T_{n+1}(x) = 2xT_{n(x)} - T_{n-1}(x), n \geq 1$$

1. In the first task of this exercise we implement the recursive definition of the polynomials in a function cheby(x, N), where x is a vector of floats between −1.0 and 1.0, while N is the maximal polynomial degree we want to calculate. In the program we initialize x with 200 values starting at −1.0 and incremented by 0.1 in each step up to 1.0. We store the results in a matrix with the rows being the evaluations of the data points in x for the polynomial of degree n, where n equals the row number.

2. In the second taks of this exercise we plot the resulting matrix in a graph, as shown in Figure 1. To achieve this we use the library 'matplotlib' (available here: https://github.com/lava/matplotlib-cpp) which is a C++ wrapper of the python library matplotlib. Each row of the matrix results in a line in the graph.

Simulation results

This section shows the results of the Tasks desribed in the previous chapter. To maintain a clean layout, we round all decimal numbers to the second decimal place.

Exercise 1:

Matrix A with seeded random numbers in each cell:

$$\begin{pmatrix} 2.36 & 4.97 & -1.48 & -2.28 & -4.68 & 0.11 \\ 1.61 & -0.32 & -2.25 & -4.75 & 0.59 & 3.80 \\ -2.12 & -3.11 & 4.25 & 0.13 & -2.09 & -0.37 \\ 1.64 & 2.46 & -4.25 & 0.13 & 4.05 & -2.28 \\ 1.53 & 4.00 & -1.85 & 0.04 & -4.96 & -1.10 \\ 4.73 & 0.18 & 4.73 & 0.59 & -0.39 & 3.57 \end{pmatrix}$$

The largest element has a value of 4.97 and is in row 0 and column 1.

The row vector containing the largest element of each column becomes

$$(4.73 \;\; 4.97 \;\; 4.73 \;\; 0.59 \;\; 4.05 \;\; 3.80).$$

The column vector containing the largest element of each row becomes,

$$(4.97 \;\; 3.80 \;\; 4.25 \;\; 4.05 \;\; 4.00 \;\; 4.73).$$

The result of the vector multiplication is 99.09.

Matrix B with seeded random numbers in each cell:

$$\begin{pmatrix} 4.97 & -0.04 & 3.61 & -4.78 & -0.66 & 2.32 \\ -3.63 & -1.92 & -1.20 & 3.43 & 3.21 & -0.80 \\ -0.65 & 3.53 & 2.99 & -4.95 & -0.07 & 0.69 \\ -4.64 & -4.30 & -2.46 & -3.00 & 4.61 & 1.85 \\ 1.44 & 1.18 & -2.83 & 3.31 & -0.26 & -4.93 \\ 0.38 & 0.33 & 0.10 & 0.61 & -4.50 & -4.01 \end{pmatrix}$$

Matrix $C = A * B$

$$\begin{pmatrix} -1.49 & -10.55 & 16.98 & 4.55 & 4.69 & 18.90 \\ 34.99 & 14.99 & 9.86 & 20.87 & -41.07 & -24.47 \\ -5.77 & 17.89 & 14.35 & -29.16 & -6.05 & 12.58 \\ 6.33 & -16.31 & -21.75 & 33.28 & 16.91 & -11.75 \\ -13.47 & -20.69 & 8.99 & -1.55 & 18.35 & 27.98 \\ 17.79 & 14.38 & 31.02 & -46.29 & -16.13 & 2.82 \end{pmatrix}$$

Matrix $D = B * A$

$$\begin{pmatrix} 6.16 & -0.5 & 40.63 & -9.9 & -47.82 & 18.98 \\ -2.39 & 7.41 & -19.73 & 17.32 & 16.71 & -21.46 \\ -7.1 & -26.0 & 30.2 & -15.13 & -21.12 & 26.1 \\ -1.82 & -2.67 & 19.02 & 31.57 & -11.38 & -7.6 \\ -7.04 & 21.76 & -53.73 & -11.78 & 16.48 & -19.16 \\ -23.58 & -15.73 & -14.13 & -4.91 & 24.57 & -9.51 \end{pmatrix}$$

Exercise 2:

As discussed previously, we plot the resulting matrix of the function cheb(x,N), which describes the Chebychev Polynomials, as a graph in Figure 1. For x we use a vector of 200 data points between –1.0 and 1.0 and a value of 5 for N. These chosen values are used as an example. We could also use more or less data points, or calculate polynomials of higher or lower degrees.

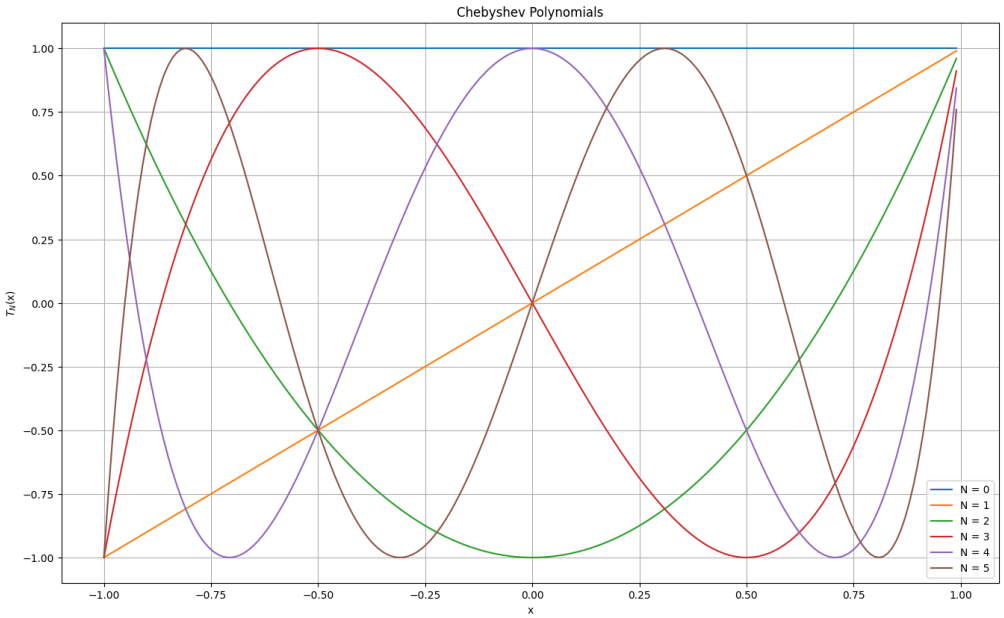

Figure 1: The Chebyshev polynomials of degrees $N = 0$ to $N = 5$. The x-axis displays the data values x between −1.0 and 1.0. The y-axis displays their evaulated values between −1.0 and 1.0.

Discussion

As this Exercise served as a 'warm up' there is not much we can discuss at this point. We include this section for completness.

In exercise 1, we can validate our result by using the `np.matmul()` function or similar implementations and from matrix C and D we see that matrix multiplication is not commutative, which is a fact that we have learned from school.

In exercise 2, we discover that the result from the recursive relation matches the original definition of the Chebyshev polynomial.

Appendix

```
#include <iostream>
#include <vector>

// plotting library available here: https://github.com/lava/matplotlib-cpp
#define WITHOUT_NUMPY // setting for matplotlib
#include "matplotlibcpp.h"

namespace task_1
```

```
{
struct Index
{
    int row = 0;
    int column = 0;
};

struct Matrix
{
    // Constructor for a row_length x column_length matrix with random number in its cells
              Matrix(int    row_length,    int    column_length)    :    row_length(row_length),
column_length(column_length)
    {
        // 3: create vectors, initialized at -5 because that is the smallest number
        largest_row = std::vector<float>(column_length, -5);
        largest_column = std::vector<float>(row_length, -5);
        for (int i = 0; i < row_length; ++i)
        {
            matrix.emplace_back();
            for (int j = 0; j < column_length; ++j)
            {
                // 1.:create random element between -5.0 and 5.0
                                  float  element  =  -5.f  +  static_cast<float>(std::rand())  /
(static_cast<float>(RAND_MAX / 11));
                // place element into matrix
                matrix[i].emplace_back(element);
                std::cout << matrix[i][j] << ',';
                // 3.: check if element is largest found in column j so far
                if (element > largest_row[j])
                {
                    // 3.: if so place replace current element
                    largest_row[j] = element;
                }
                 // 3.: check if element smaller or equal to the currently largest element in
row i
                if (element <= largest_column[i])
                {
                       // if so there is no need for the last check as we know it cant be the
largest in the matrix
                    continue;
                }
                // 3.: else replace currently largest row element
                largest_column[i] = element;
                // 2.: check if element is largest in matrix
                if (element > matrix[largest_element_index.row][largest_element_index.column])
                {
                    // 2.: if so replace the index of the largest element
                    largest_element_index.row = i;
                    largest_element_index.column = j;
                }
            }
            std::cout << ";\n";
        }

        // only do the next part if it is mathematically possible (for this exercise it always
does)
        if (row_length != column_length)
            return;

        for (int i = 0; i < row_length; ++i)
        {
            // 3.: multiply row with column vector
```

```
            largest_scalar += largest_row[i] * largest_column[i];
        }
    };

    Matrix(Matrix A, Matrix B) // constructor for 4.
    {
        // make sure it is mathematically possible
        if (A.column_length != B.row_length)
        {
            std::cout << "Invalid Matrix sizes.\n";
            return;
        }

        for (int k = 0; k < A.row_length; ++k)
        {
            matrix.emplace_back();
            for (int i = 0; i < A.column_length; ++i)
            {
                matrix[k].emplace_back();
                for (int j = 0; j < B.row_length; ++j)
                {
                    // standard matrix multiplication
                    matrix[k][i] += A.matrix[k][j] * B.matrix[j][i];
                }
                std::cout << matrix[k][i] << ',';
            }
            std::cout << ";\n";
        }
    };

    // output function for 2. (value never changes in exercise -> no need to recalculate)
    void largest_element()
    {
        std::cout << "The largest element has a value of " << matrix[largest_element_index.row]
[largest_element_index.column] << " and is in row "
                  << largest_element_index.row << " and column " << largest_element_index.column
<< ".\n";
    };

    // output for 3. (value never changes in exercise -> no need to recalculate)
    void largest_scalar_result() { std::cout << "The result of the vector multiplication is "
<< largest_scalar << ".\n"; };

    std::vector<std::vector<float>> matrix;
    int largest_scalar = 0;
    int row_length;
    int column_length;
    Index largest_element_index;
    std::vector<float> largest_row;
    std::vector<float> largest_column;
};

void task_1()
{
    std::srand(5378);

    // 1.: create matrix A
    std::cout << "Matrix A: \n";
    Matrix A(6, 6);
    // 2.: output largest element
    A.largest_element();
    // 3.: output largest scalar
```

```
    A.largest_scalar_result();

    // 4.: Create Matrix B
    std::cout << "\nMatrix B: \n";
    Matrix B(6, 6);

    // 4.: Create Matrix C as A * B
    std::cout << "\nMatrix C: \n";
    Matrix C(A, B);

    // 4.: Create Matrix D as B * A
    std::cout << "\nMatrix D: \n";
    Matrix D(B, A);
}
} // namespace task_1

namespace task_2
{
// 1.: cheby function
auto cheby(std::vector<float> x, int N)
{
    std::vector<std::vector<float>> result(N + 1); // N + 1 rows

    std::vector<float> chebn(N + 1);
    // solves the values of 1 datapoint element for the N+1 chebychev Polynomials recursively
    auto chebx = [&chebn](float element, int n) mutable -> void
    {
        // level of indirection for recursion
          auto chebx_impl = [&chebn](std::vector<float>& p_chebn, float element, int n, auto&
chebx_ref) mutable -> void
        {
            if (n == 0)
            {
                chebn[n] = 1;
                return;
            }
            if (n == 1)
            {
                chebx_ref(chebn, element, n - 1, chebx_ref);
                chebn[n] = element;
                return;
            }

            chebx_ref(chebn, element, n - 1, chebx_ref);
            chebn[n] = 2 * element * chebn[n - 1] - chebn[n - 2];
            return;
        };
        chebx_impl(chebn, element, n, chebx_impl);
        return;
    };

    std::cout << "\nChebyshev Polynomial Matrix:\n";

    for (int i = 0; i < x.size(); ++i)
    {
        // chebx gets called for every element
        chebx(x[i], N);
        for (int j = 0; j <= N; ++j)
        {
            // place value of polynomial j in row j
            result[j].emplace_back(chebn[j]);
        }
```

```
    }

    for (auto const& row : result)
    {
        for (auto const& element : row)
        {
            std::cout << element << '\t';
        }
        std::cout << '\n';
    }

    return result;
}

void task_2()
{
    namespace plt = matplotlibcpp; // convenience

    // create a vector x with 200 data points
    std::vector<float> x(200);
    float e = -1.f;
    for (int i = 0; i < 200; ++i)
    {
        x[i] = e;
        e += 0.01f;
    }
    // 1.: create matrix
    auto cheb = task_2::cheby(x, 5);

    // 2.: Plot Polynomials
    for (int i = 0; i < cheb.size(); ++i)
    {
        std::vector<float> y = cheb[i];

        plt::plot(x, y, {{"label", "N = " + std::to_string(i)}});
    }

    // plot settings
    plt::title("Chebyshev Polynomials");
    plt::xlabel("x");
    plt::ylabel("$T_N$(x)");
    plt::grid(true);
    plt::legend();
    plt::show();
}
} // namespace task_2

int main()
{
    task_1::task_1();

    task_2::task_2();

    return 0;
}
```