Computational Physics - Exercise 1

Joel Schumacher, 309231, joel.schumacher@rwth-aachen.de

May 3, 2017

Task 1

1.

With my matricle number being 309231, I chose the seed of the Mersenne Twister random number generator used by the numpy module as 9231. The convenience function *np.random.rand* will return uniform random samples in the interval $[0, 1)$, so that I have to rescale into $[-5.0, 5.0)$.

The resulting matrix A is then:

$$A = \begin{pmatrix} 2.2029 & 4.9743 & -2.0779 & 3.1163 & 1.2233 & -2.0957 \\ -4.1435 & 1.4772 & 0.3867 & 2.7847 & -0.1969 & -3.7180 \\ 4.5648 & -4.4702 & -0.8552 & 3.4828 & 0.4080 & -3.1573 \\ 0.1776 & -4.0575 & -1.2090 & 2.4640 & 4.7895 & -3.2285 \\ 4.8563 & 2.5141 & 1.2010 & -1.8702 & 1.9321 & 2.6607 \\ -2.6221 & 1.5498 & -0.3413 & 4.3567 & -1.2784 & 2.6796 \end{pmatrix}$$

2.

The largest element of the matrix was then found at the index $0, 1$. In regular mathematical notation this corresponds to the element A_{12}, which is equal to $A_{12} = 4.9743$.

3.

The vectors containing the largest value for each column r, a row-vector and c a column-vector containing the largest value for each row will be determined using the extra argument of numpy's *max* function. The results are:

$$r = \begin{pmatrix} 4.8563 & 4.9743 & 1.2010 & 4.3567 & 4.7895 & 2.6796 \end{pmatrix}$$

$$c = \begin{pmatrix} 4.9743 \\ 2.7847 \\ 4.5648 \\ 4.7895 \\ 4.8563 \\ 4.3567 \end{pmatrix}$$

The product of these vectors, assuming we are not supposed to calculate $c \cdot r$ is then

$$r \cdot c \approx 99.29$$

Which is above what would have been expected, since the expected value of the maximum of N samples of a uniform distribution between $[0, 1]$ is $\frac{N}{N+1}$. For a proof, see the appendix Expectation value of the maximum of N samples of a uniform distribution in $[0, 1]$. The rescaled expected value for the maximum of 6 independent samples is then $m = \frac{6}{7} * 10 - 5 \approx 3.57$. We would then expect a scalar product of $< r \cdot c >= 6m^2 \approx 76.53$.

4.

Our second random matrix is:

$$B = \begin{pmatrix} 4.8686 & -3.3602 & -1.8808 & 4.2009 & -1.6982 & -3.8247 \\ 2.5127 & -4.7676 & -2.5390 & -2.4462 & -0.7683 & 3.8182 \\ -0.5627 & 1.4668 & 1.6576 & -1.4767 & -2.8466 & -4.3560 \\ 4.2442 & 1.5349 & -2.6651 & -4.3131 & -4.8585 & -2.9110 \\ -1.6806 & 3.0978 & -1.4647 & -3.4098 & -2.9560 & 1.5230 \\ -3.3280 & -2.2861 & 3.8720 & 0.8065 & 1.3259 & -2.3052 \end{pmatrix}$$

Using the function *np.dot* we can calculate the matrix products $C = A \cdot B$ and $D = B \cdot A$:

$$C = A \cdot B = \begin{pmatrix} 42.5379 & -20.8025 & -38.4292 & -19.1478 & -23.1825 & 17.2414 \\ 7.8440 & 19.6115 & -16.8457 & -35.9287 & -13.0759 & 19.9684 \\ 36.0774 & 18.5460 & -20.7583 & 12.4152 & -24.1965 & -33.0410 \\ 4.5027 & 42.9743 & -18.1186 & -17.1054 & -24.1526 & -3.3417 \\ 9.2449 & -29.5105 & -1.0696 & 16.1004 & -6.6947 & -11.9526 \\ 3.0418 & -2.4778 & 1.0678 & -26.5728 & -9.6015 & -3.3734 \end{pmatrix}$$

$$D = B \cdot A = \begin{pmatrix} 18.5909 & 0.4196 & -15.6203 & -3.8718 & 27.5784 & -20.1008 \\ -0.4773 & 30.7177 & -4.1614 & -2.2452 & -15.1054 & 36.5615 \\ -2.4147 & -15.9583 & 0.1718 & -9.1885 & -7.3050 & -23.9865 \\ -25.9029 & 36.0671 & -5.5736 & -6.0057 & -22.5206 & -12.9891 \\ -42.1783 & 11.5273 & 5.9952 & 2.0493 & -27.2530 & 3.8533 \\ 32.4428 & -40.7519 & 4.1238 & -13.7866 & 7.3300 & -2.0041 \end{pmatrix}$$

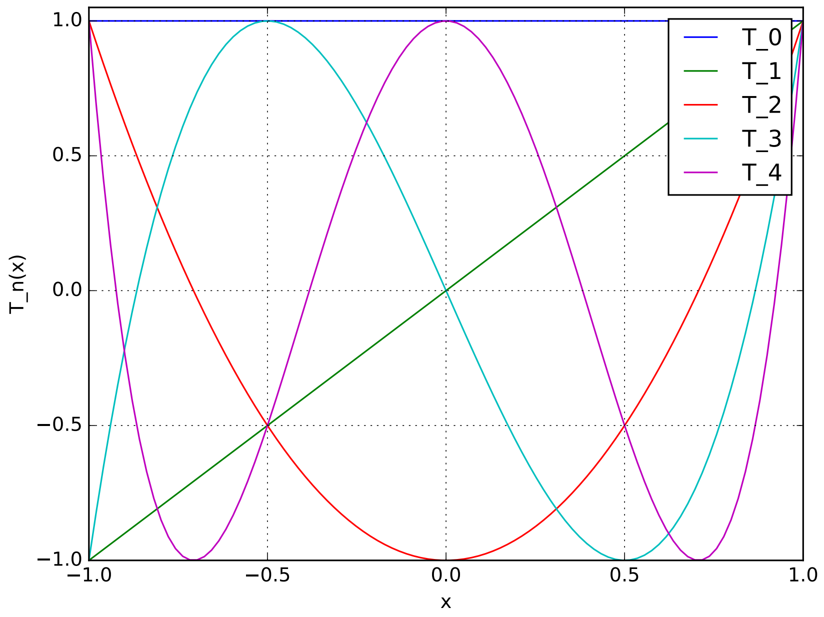

Figure 1: The first 5 Chebychev polynomials of the first kind

Task 2

In this task we write a function that evaluates and then plot the first 5 Chebychev polynomials of the first kind. They are the solution to the second order ordinary differential equations:

$$(1-x^2)y'' - xy' + n^2y = 0$$

$$(1-x^2)y'' - 3xy' + n(n+2)y = 0$$

They are orthogonal and have properties that lend themselves to being used for interpolation.

For evaluating the polynomials in this task, I chose the recursion on the work sheet for ease of implementation, though a more numerically stable and more efficient algorithm, the Clenshaw algorithm would be a better alternative.

A plot of the first 5 Chebychev polynomials can be found in figure figure 1

Task 3

When incrementing an unsigned 8-bit integer 300 times, I noticed the value of the variable starting over after 255 at 0. This is known as an integer overflow and is a result of the finite representation of numbers in a machine. Formally after incrementing 255, which in binary is represented as 11111111 the result is 100000000, which has no bits not equal to zero that fit into the 8-bit variable, which will result in the variable starting over at 0. The first bit (1), being a carry over into the least significant bit of a more significant word, is called the carry bit and can, if handled in a programming language with adequate access, be queried from the CPU as the "carry flag". Naturally this can occur even for bigger (any) word sizes, such as the 64-Bit integer on now commonly used 64-Bit x86 machines. If dealing with a signed variable, the overflow will also result in a sign change.

It should be noted that in Python this phenomenon can usually not be observed since regular Python variables have arbitrary integer precision and grow as needed internally. For this exercise the workaround was to use a numpy array, which wrap C-style arrays with C datatypes, containing only one element.

Appendix

Task 1 program listing

```
import numpy as np

# Sorry for not commenting these properly
def np_mat2tex(m):
    return " \\\\ \n".join(" & ".join("{:.4f}".format(elem)
        for elem in row) for row in m.tolist())

def np_vec2tex(m, delim=" \\\\ "):
    return delim.join("{:.4f}".format(elem)
        for elem in m.tolist())

# 1.
# 309231
np.random.seed(9231)

# Rescale random values from [0,1]
A = np.random.rand(6,6) * 10.0 - 5.0
print("A = ", np_mat2tex(A), "\n")

# 2.
max_value = A.max()
max_index = np.where(A == max_value)
print(max_index, max_value, "\n")

# 3.
row_max = A.max(axis=0)
```

```
col_max = A.max(axis=1)
print("r = ", np_vec2tex(row_max, " & "))
print("c = ", np_vec2tex(col_max), "\n")

# I hope row_max * col_max is fine (instead of the other way round)
print(np.dot(row_max, col_max))

# 4.
B = np.random.rand(6,6) * 10.0 - 5.0
C = np.dot(A, B)
D = np.dot(B, A)

print("B = ", np_mat2tex(B))
print("C = ", np_mat2tex(C))
print("D = ", np_mat2tex(D))
```

Task 2 program listing

```
import numpy as np
import matplotlib.pyplot as plt

def cheby(x, N):
        ret = np.zeros((N+1, len(x)))
        ret[0] = np.ones(len(x))
        ret[1] = np.copy(x)
        for n in range(2,N+1):
                ret[n] = 2*x*ret[n-1] - ret[n-2]
        return ret

x = np.linspace(-1, 1, 100)
T = cheby(x, 4)
for i in range(len(T)):
        plt.plot(x, T[i], label="T_" + str(i))
plt.legend()
plt.grid(True)
plt.xlabel("x")
plt.ylabel("T_n(x)")
plt.ylim(-1.0, 1.05)
plt.savefig("task2.pdf")
```

Task 3 program listing

```
import numpy as np

i = np.arange(1, dtype=np.uint8)
print(i[0])
```

```
for j in range(0, 300):
        i[0] = i[0] + 1
        print(i[0])
```

Expectation value of the maximum of N samples of a uniform distribution in $[0, 1]$

Let $X_i \in [0, 1]$ be $N \in \mathbb{N}$ independent uniformly distributed random variables. Let $M = max(X_{1,}, X_2, ..., X_N)$ be the maximum in question. The probability of M being smaller than some $x \in \mathbb{R}$ is:

$$P(M \leq x) = P(X_1 \leq x, X_2 \leq x, ..., X_N \leq x) = P(X_1 \leq x) \cdot P(X_2 \leq x) \cdot ... \cdot P(X_N \leq x) = x^N$$

Since all random variables are independent the factorization in the second step is allowed.

$P(M \leq x)$ being the definition for the cumulative distribution function, we can obtain the probability distribution for M by differentiation:

$$\rho_M(x) = N \cdot x^{N-1}$$

And it follows:

$$E[M] = \int_0^1 x \cdot N \cdot x^{N-1} dx = \frac{N}{N+1}$$