Exercise Sheet № 5 Function Estimation

Task 6.1: Operation Count and Orthogonal Matrices

i) Given a linear system of n equations, calculate the number of multiplications, divisions, additions and subtractions performed by:
   a) for-loop version of the row-oriented forward substitution
   b) for-loop version of the row-oriented backward substitution
   c) for-loop version of the LU decomposition method

ii) Let $\mathrm{Q} \in \mathbb{R}^{n\times n}$ be orthogonal. Prove the following:
   a) $\forall \boldsymbol{x} \in \mathbb{R}^n : \|\mathrm{Q}\boldsymbol{x}\|_2 = \|\boldsymbol{x}\|_2$
   b) $\kappa_2(\mathrm{Q}) = 1$
   c) $\widehat{\mathrm{Q}} \in \mathcal{O}(n) \Rightarrow \widehat{\mathrm{Q}}\mathrm{Q} \in \mathcal{O}(n)$
   d) $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n : \angle \boldsymbol{x}, \boldsymbol{y} = \angle \mathrm{Q}\boldsymbol{x}, \mathrm{Q}\boldsymbol{y}$

Subtask i):

Sub-Subtask a):

Algorithm 1: Forward-Substitution

```
name: forwsubs
input: m × n lower triangular matrix U, b ∈ ℝ^n
output: x ∈ ℝ^m solving Ux = b

forwsubs(U, b):
    x = 0_m
    for j = 1,...,n do
        x[j] = (b[j] − ∑_{k=1}^{j−1} U[j,k]x[k]) · 1/U[j,j]
    end
    return x
```

The sum for $\boldsymbol{x}[j]$ is has $j-1$ multiplications and additions. Thus for $\boldsymbol{x}[j]$ we have $j-1$ additions and multiplications, one subtraction and one division:

$$\sum_{j=1}^{n} j-1 = \sum_{j=1}^{n} j - \sum_{j=1}^{n} 1 = \frac{n(n+1)}{2} - n = \frac{n^2+n-2n}{2} = \frac{n^2-n}{2} = \frac{n(n-1)}{2} \in \mathcal{O}(n^2)$$

Sub-Subtask b):

Algorithm 2: Backward-Substitution

```
name: backsubs
input: m × n upper triangular matrix L, b ∈ ℝ^n
output: x ∈ ℝ^m solving Lx = b

backsubs(L, b):
    x = 0_m
    for j = n,...,1 do
        x[j] = (b[j] − ∑_{k=j+1}^{n} L[j,k]x[k]) · 1/L[j,j]
    end
    return x
```

Similarly to forward-substitution, we get $j-1$ additions and multiplications per iteration, and one subtractions and division.

Sub-Subtask c): The LU decomposition requires additional operations, such as comparisons and swaps. We will ignore them, even though finding maximum entries in a column has a worst-case time of n.

Algorithm 3: LU-Decomposition with partial pivoting

```
name: LUP
input: n × n matrix A
output: n × n LT matrix L, n × n UT matrix U, n × n permutation matrix P

LUP(A):
    U = A
    L = I_n
    P = I_n
```

```

    for j = 1,...,n−1 do
        s = argmax_{k=j,...,n} |U[k,j]|
        if s ≠ j then
            swap(U,s,j)
            swap(P,s,j)
        end

        for i = j+1,...,n do
            L[i,j] = U[i,j]/U[j,j]
            for k = j+1,...,n do
                U[i,k] = U[i,k] − L[i,j]U[j,k]
            end
        end
    end
    return L,U,P
```

Per j we have $n-j$ divisions for L and per i we get one multiplication and one subtraction for U, thus:

$$\sum_{j=1}^{n-1}\sum_{i=j+1}^{n}\sum_{k=j+1}^{n}1=\sum_{j=1}^{n-1}\sum_{i=j+1}^{n}n-j=\sum_{j=1}^{n-1}(n-j)^2=\frac{1}{6}n(2n^2-3n+1)\in\mathcal{O}(n^3)$$

$$\sum_{j=1}^{n-1}n-j=\frac{1}{2}n(n-1)\in\mathcal{O}(n^2)$$

algorithm	***additions***	***multiplications***	***subtractions***	***divisions***
forward-substitution	$\frac{n(n-1)}{2}$	$\frac{n(n-1)}{2}$	n	n
backward-substitution	$\frac{n(n-1)}{2}$	$\frac{n(n-1)}{2}$	n	n
LU decomposition	0	$\frac{1}{6}n(2n^2-3n+1)$	$\frac{1}{6}n(2n^2-3n+1)$	$\frac{1}{2}n(n-1)$

Table 1: Number of float operations for various algorithms

Subtask ii):

We denote the set of all orthogonal matrices as $\mathcal{O}(n)=\{\mathrm{A}\in\mathbb{R}^{n\times n}\colon \mathrm{A}^{-1}=\mathrm{A}^T\}$.

Sub-Subtask a): Let $\boldsymbol{x}\in\mathbb{R}^n$ and $\mathrm{Q}\in\mathcal{O}(n)$, then:

$$\|\mathrm{Q}\boldsymbol{x}\|_2=\sqrt{\langle\mathrm{Q}\boldsymbol{x},\mathrm{Q}\boldsymbol{x}\rangle}=\sqrt{\boldsymbol{x}^T\mathrm{Q}^T\mathrm{Q}\boldsymbol{x}}=\sqrt{\boldsymbol{x}^T\mathrm{I}\boldsymbol{x}}=\sqrt{\langle\boldsymbol{x},\boldsymbol{x}\rangle}=\|\boldsymbol{x}\|_2$$

Sub-Subtask b): Let $\boldsymbol{x}\in\mathbb{S}^{n-1}$:

$$\kappa_2(\mathrm{Q})=\frac{\max\|\mathrm{Q}\boldsymbol{x}\|_2}{\min\|\mathrm{Q}\boldsymbol{x}\|_2}=\frac{\max\|\boldsymbol{x}\|_2}{\min\|\boldsymbol{x}\|_2}=1$$

Sub-Subtask c): Let $\widehat{\mathrm{Q}}\in\mathcal{O}(n)$:

$$(\widehat{\mathrm{Q}}\mathrm{Q})^T\widehat{\mathrm{Q}}\mathrm{Q}=\mathrm{Q}^T\widehat{\mathrm{Q}}^T\widehat{\mathrm{Q}}\mathrm{Q}=\mathrm{Q}^T\mathrm{I}\mathrm{Q}=\mathrm{Q}^T\mathrm{Q}=\mathrm{I}$$
$$\Rightarrow(\widehat{\mathrm{Q}}\mathrm{Q})^{-1}=(\widehat{\mathrm{Q}}\mathrm{Q})^T\Leftrightarrow\widehat{\mathrm{Q}}\mathrm{Q}\in\mathcal{O}(n)$$

Sub-Subtask d): Let $\boldsymbol{x},\boldsymbol{y}\in\mathbb{R}^n$:

$$\angle\boldsymbol{x},\boldsymbol{y}=\arccos\left(\frac{\langle\boldsymbol{x},\boldsymbol{y}\rangle}{\|\boldsymbol{x}\|_2\cdot\|\boldsymbol{y}\|_2}\right)$$
$$\angle\mathrm{Q}\boldsymbol{x},\mathrm{Q}\boldsymbol{y}=\arccos\left(\frac{\langle\mathrm{Q}\boldsymbol{x},\mathrm{Q}\boldsymbol{y}\rangle}{\|\mathrm{Q}\boldsymbol{x}\|_2\cdot\|\mathrm{Q}\boldsymbol{y}\|_2}\right)=\arccos\left(\frac{\boldsymbol{x}^T\mathrm{Q}^T\mathrm{Q}\boldsymbol{y}}{\|\boldsymbol{x}\|_2\cdot\|\boldsymbol{y}\|_2}\right)$$
$$=\arccos\left(\frac{\boldsymbol{x}^T\mathrm{I}\boldsymbol{y}}{\|\boldsymbol{x}\|_2\cdot\|\boldsymbol{y}\|_2}\right)=\arccos\left(\frac{\langle\boldsymbol{x},\boldsymbol{y}\rangle}{\|\boldsymbol{x}\|_2\cdot\|\boldsymbol{y}\|_2}\right)$$
$$=\angle\boldsymbol{x},\boldsymbol{y}$$

Task 6.2: Overdetermined Systems

In the following, let

$$A = \begin{bmatrix} 1 & -1 & 4 \\ 1 & 4 & -2 \\ 1 & 4 & 2 \\ 1 & -1 & 0 \end{bmatrix} \qquad \boldsymbol{b} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \tag{1}$$

i) Verify that the given matrix A has full column rank and compute its QR decomposition using householder reflections
ii) Given a matrix $A \in \mathbb{R}^{m \times n}$ with full column rank and a vector $\boldsymbol{b} \in \mathbb{R}^m$ where $m \geq n$, write a python script that returns the least squares solution of the linear system $A\boldsymbol{x} = \boldsymbol{b}$ using the Householder QR decomposition method. Test your script by using A and $\boldsymbol{b}$ from Equation 1

Subtask i): We apply Gaussian Elimination to bring A into row-echelon form:

$$A \xrightarrow{II-I, III-I, IV-I} \begin{bmatrix} 1 & -1 & 4 \\ 0 & 5 & -6 \\ 0 & 5 & -2 \\ 0 & 0 & -4 \end{bmatrix} \xrightarrow{III-II} \begin{bmatrix} 1 & -1 & 4 \\ 0 & 5 & -6 \\ 0 & 0 & 4 \\ 0 & 0 & -4 \end{bmatrix} \xrightarrow{IV+III} \begin{bmatrix} 1 & -1 & 4 \\ 0 & 5 & -6 \\ 0 & 0 & 4 \\ 0 & 0 & 0 \end{bmatrix}$$

Thus A has column-rank 3, i.e. it has full-column rank. Computing the QR-decomposition yields:

$$Q = \begin{bmatrix} -0.5 & -0.5 & -0.5 & -0.5 \\ -0.5 & 0.83 & -0.16 & -0.16 \\ -0.5 & -0.16 & 0.83 & -0.16 \\ -0.5 & -0.16 & -0.16 & 0.83 \end{bmatrix} \qquad R = \begin{bmatrix} -2 & -3. & -2 \\ 0 & 3.33 & -4 \\ 0 & 3.33 & 0 \\ 0 & -1.66 & -2 \end{bmatrix}$$

$$Q = \begin{bmatrix} -0.5 & 0.5 & -0.1 & -0.7 \\ -0.5 & -0.5 & -0.7 & 0.1 \\ -0.5 & -0.5 & 0.7 & -0.1 \\ -0.5 & 0.5 & 0.1 & 0.7 \end{bmatrix} \qquad R = \begin{bmatrix} -2 & -3 & -2 \\ 0 & -5 & 2 \\ 0 & 0 & 2.4 \\ 0 & 0 & -3.2 \end{bmatrix}$$

$$Q = \begin{bmatrix} -0.5 & 0.5 & -0.5 & -0.5 \\ -0.5 & -0.5 & 0.5 & -0.5 \\ -0.5 & -0.5 & -0.5 & 0.5 \\ -0.5 & 0.5 & 0.5 & 0.5 \end{bmatrix} \qquad R = \begin{bmatrix} -2 & -3 & -2 \\ 0 & -5 & 2 \\ 0 & 0 & -4 \\ 0 & 0 & 0 \end{bmatrix}$$

Subtask ii):

Algorithm 4: Full QR-decomposition using Householder Reflections

```
name: QR
input: m × n matrix A
output: m × m orthogonal matrix Q, m × n upper triangular matrix R

QR(A):
    R = A
    Q = I_m

    for k = 1, ..., n do
        a_k = [R[k, k]  ···  R[k, m]]^T
        u_k = [0_k   a_k + sign(R[k, k])‖a_k‖_2 e_{1,m−k}]^T
        R = R − (2/‖u_k‖_2^2) u_k R^T u_k^T
        Q = Q H_{u_k}
    end

    return Q, R
```

Solving the system yields the solution

$$\boldsymbol{x} = \begin{bmatrix} 2.9 \\ -0.1 \\ -0.25 \end{bmatrix}$$

The computation of this solution can be found in the submitted Jupyter Notebook `ex_2.ipynb`.

The implementation supplied in `QR.py` contains the function `QR(A,b,mode)`, where mode is a supported keyword-argument that specifies wether the full QR-decomposition should be computed, or the least-squares solution of $A\boldsymbol{x} = \boldsymbol{b}$ should be computed. If mode is set to „full", then the full QR-decomposition is computed, which is the default. If it is set to „solve" then only the solution vector is computed.

Task 6.3: Curve Fitting

Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ be an arbitrary dataset to be fitted with a polynomial of degree $m \in \mathbb{N}$. Write a python-script containing the following functions:

i) `PolyFit(x_data, y_data, m)` that sets up the normal equations for the coefficients of a polynomial of degree m and returns the vector $\boldsymbol{c}$ of coefficients. Test your script using the given data from Table 2 and $m = 1, 2, 3$.

ii) `StdDev(c, x_data, y_data)` which computes the standard deviation of $f(x)$ and the data. Test your script using the data from Table 2.

iii) `PlotPoly(x_data, y_data, c)` wich plots the data points and the fitting polynomial. Test your script using the data from Table 2 and $m = 1, 2, 3$.

iv) Write a program, that fits a polynomial of arbitrary degree m to the data points from Table 2. The program should be able to determine the polynomial degree m that „best“ fits the data in the least squares sense using the standard deviation as best fit measure. Provide a visualization of the given data and the fitting polynomials in one figure frame.

$\boldsymbol{x}$	-0.04	0.93	1.95	2.90	3.83	5.00	5.98	7.05	8.21	9.08	10.09
$\boldsymbol{y}$	-8.66	-6.44	-4.36	-3.27	-0.88	0.87	3.31	4.63	6.19	7.40	8.85

Table 2: The dataset for testing

Bonus: Explain the difference between curve fitting and polynomial interpolation

First a bit of theory:

Let $f(x) = f(x, \boldsymbol{a})$, $\boldsymbol{a} \in \mathbb{R}^{m+1}$, be the function that is to be fitted to the $n+1$ data points (x_i, y_i) for $i = 0, \ldots, n$, where the function f contains $m+1$ variable parameters with $m < n$. If the measurement error is confined to the y-coordinate, the most commonly used measure to determine the „best“ fit is the least squares fit, which minimizes the function

$$S(\boldsymbol{a}) = \sum_{i=0}^{n} r_i^2 \tag{2}$$

where $r_i = y_i - f(x_i)$, are called the *residuals*, with respect to each parameter a_j. The optimal values of the parameters are given by the solution of

$$\frac{\partial S}{\partial a_k} = 0 \tag{3}$$

The spread of the data about the fitting curve is quantified by the ***standard deviation*** defined as

$$\sigma = \sqrt{\frac{S}{n-m}}$$

Consider the linear form $f(x) = \sum_{i=0}^{m} a_i f_i(x)$, where each $f_i(x)$ is a predetermined function of x, called a *basis function*. Then Equation 2 is given by:

$$S = \sum_{i=0}^{n} \left(y_i - \sum_{j=0}^{m} a_j f_j(x_i) \right)^2$$

And Equation 3 becomes

$$\sum_{j=0}^{m} a_j \sum_{i=0}^{n} f_j(x_i) f_k(x_i) = \sum_{i=0}^{n} f_k(x_i) y_i$$

or in matrix notation, we have the so called *normal equations* of the least square fit $\mathrm{A}\boldsymbol{a} = \boldsymbol{b}$, where

$$\mathrm{A}_{kj} = \sum_{i=0}^{n} f_j(x_i) f_k(x_i) \qquad \boldsymbol{b}_k = \sum_{i=0}^{n} f_k(x_i) y_i$$

A commonly used linear form is a polynomial. If the degree of the polynomial is m, then we have $f(x) = \sum_{j=0}^{m} a_j x^j$ and the basis functions are given by $f_j(x) = x^j$.

First we want to verify the transformation of Equation 3 when using a linear form $f(x)$:

$$
\begin{aligned}
&S = \sum_{i=0}^{n} \left(y_i - \sum_{j=0}^{m} a_j f_j(x_i) \right)^2 = \sum_{i=0}^{n} \left(y_i^2 - 2y_i \sum_{j=0}^{m} a_j f_j(x_i) + \left(\sum_{j=0}^{m} a_j f_j(x_i) \right)^2 \right) \\
&G_i = \sum_{j=0}^{m} a_j f_j(x_i) \Rightarrow \frac{\partial G_i^2}{\partial a_k} = 2G_i \frac{\partial G_i}{\partial a_k} = 2G_i a_k f_k(x_i) \\
&S = \sum_{i=0}^{n} y_i^2 - 2y_i G_i + G_i^2 \Rightarrow \frac{\partial S}{\partial a_k} = \sum_{i=0}^{n} -2y_i a_k f_k(x_i) + 2G_i a_k f_k(x_i) \\
&\frac{\partial S}{\partial a_k} = 0 \Leftrightarrow 2a_k \sum_{i=0}^{n} G_i f_k(x_i) - y_i f_k(x_i) = 0 \Leftrightarrow \sum_{i=0}^{n} G_i f_k(x_i) = \sum_{i=0}^{n} y_i f_k(x_i) \\
&\Leftrightarrow \sum_{i=0}^{n} \sum_{j=0}^{m} a_j f_j(x_i) f_k(x_i) = \sum_{i=0}^{n} y_i f_k(x_i) \Leftrightarrow \sum_{j=0}^{m} a_j \sum_{i=0}^{n} f_j(x_i) f_k(x_i) = \sum_{i=0}^{n} y_i f_k(x_i)
\end{aligned}
$$

The submission of the fitting algorithm is supplied in the file `Fitter.py`, which implements a class called `Fitter` with members n, m and $\boldsymbol{x}$, wich represents an x-axis. The idea is to use the same instance for multiple y-values, exploiting the QR-decomposition for the normal equations. The class implements the required functions as methods.

Additionally, the class has a static-method called `find_best` which takes in $\boldsymbol{x}$ and $\boldsymbol{y}$ to find the best polynomial of degree $m = 1, \ldots, n$ with the minimal standard deviation. Since the LU-decomposition is on average longer than Gaussian Elimination, for finding the best fitting polynomial, we use Gaussian Elimination to find the coefficient vector.

The tests for $m = 1, 2, 3$ can be found in the submitted Jupyter-Notebook `ex_3.ipynb` which produces the following plot:

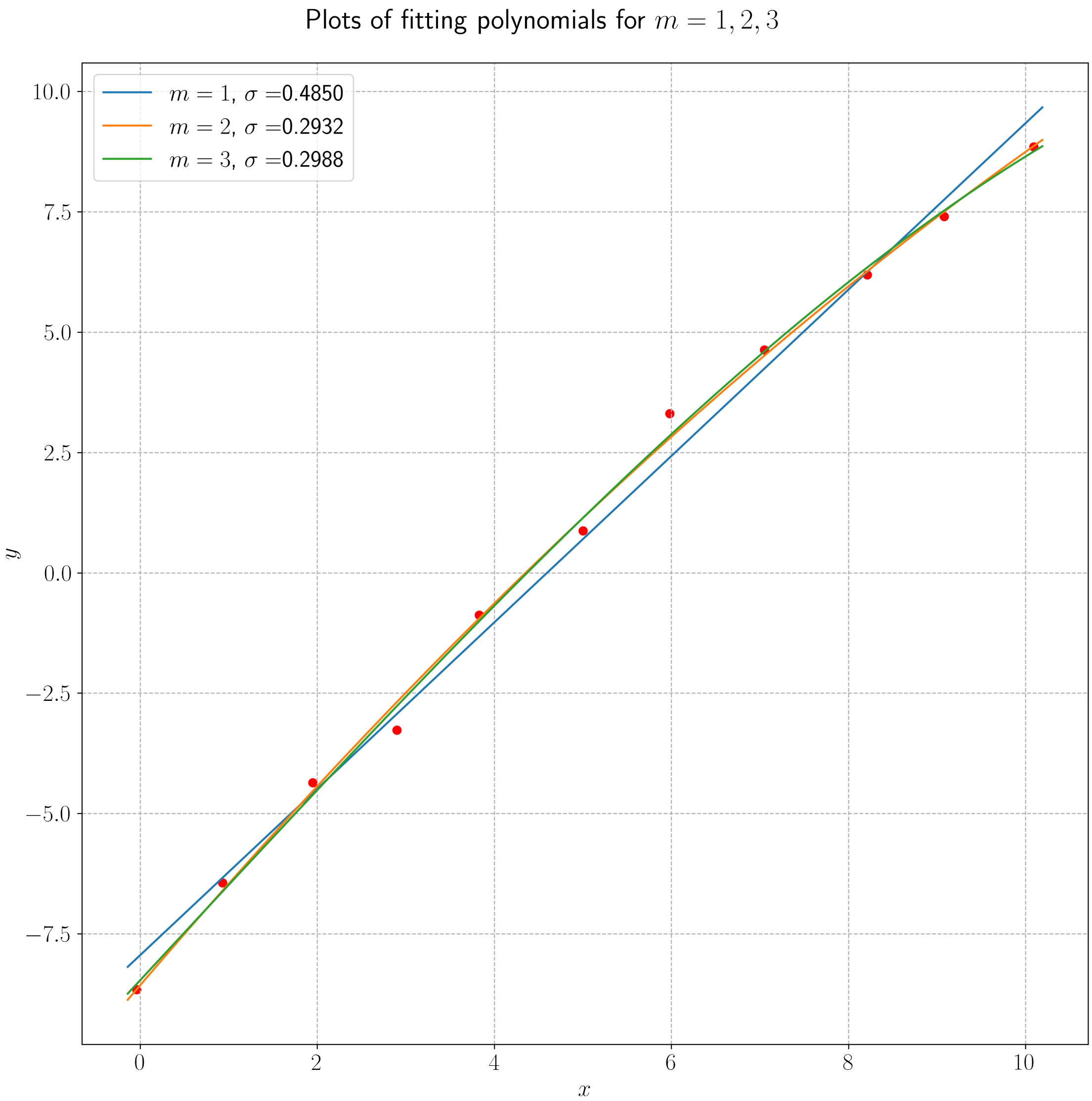

Figure 1: Plot of fitting polynomials for $m = 1, 2, 3$ for data from Table 2

The „best“ fit in terms of standard deviation is computed in the submitted script `find_best.py`, which produces the following plot:

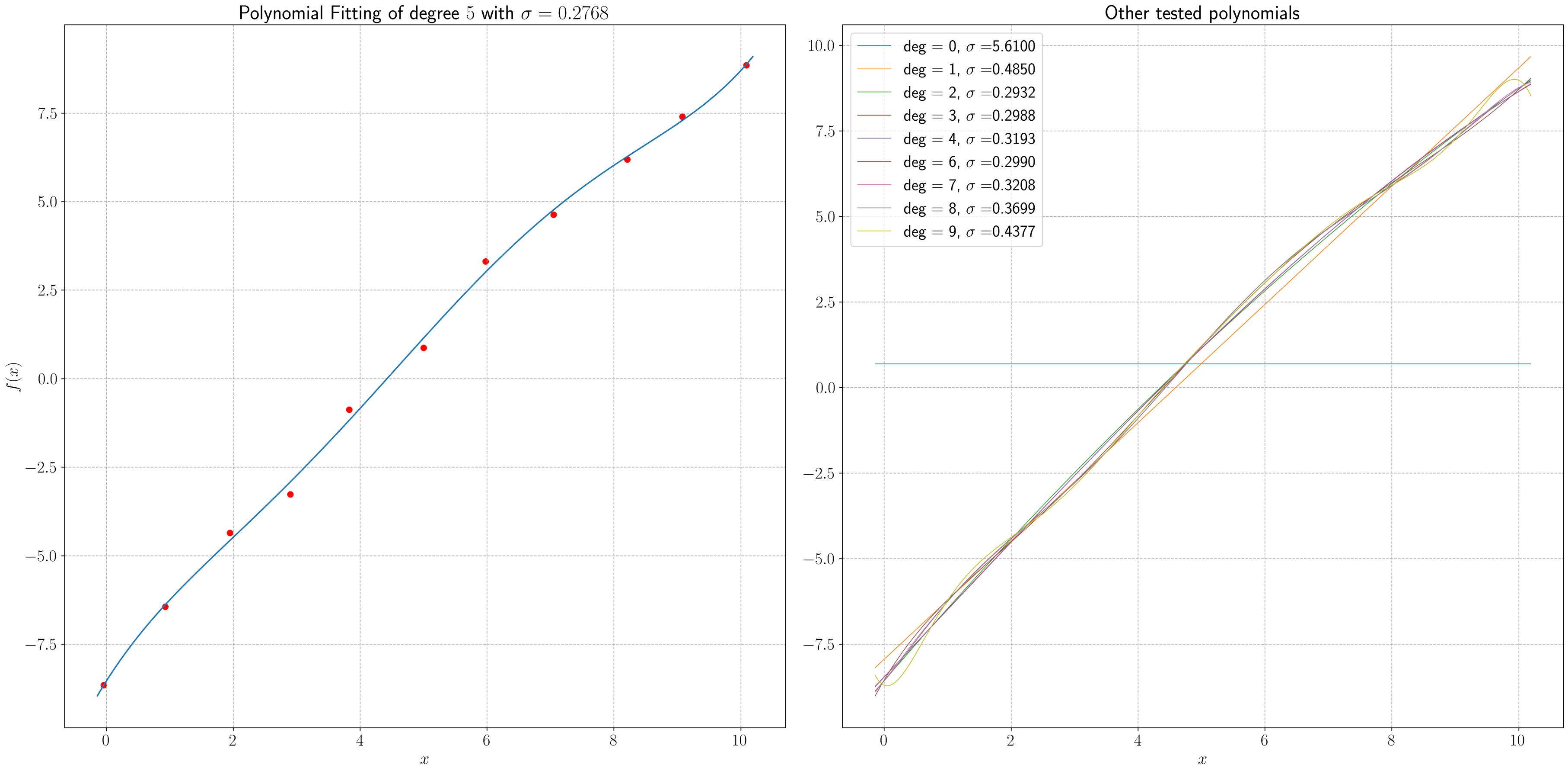

Figure 2: Best fitting polynomial and the other tested polynomials

Bonus: Notice that a fitting function does not have to equal the dataset in the given x-values. However, the advantage of using a fitting algorithm over interpolation is the fact, that a properly fitted function remains closer to the dataset than a interpolated polynomial, on average. This happens because a fitted polynomial usually has a much lower degree than an interpolated one, as interpolation with n data-points produces polynomials of degree n. High degree polynomials tend to wildly oscillate between the data-points, resulting in a worse average distance from the dataset.