Warsaw, 21.05.2019.

Assignment B: Approximation of functions #14

Final Report v2.0

Course: ENUME

Author: Kacper Kamieniarz

Index number: 293065

Advisor: Andrzej Miękina

Table of Contents

Task 1 and 2 .... **3**

Overview 3

Functions 3

Results 5

Conclusions 8

Task 3 .... **8**

Overview 8

Functions 8

Results 9

Conclusions 10

Task 4 .... **10**

Overview 10

Functions 10

Results 11

Conclusions 12

References **12**

Task 1 and 2

Overview

Task 1 introduces a function that will be used for the approximation.

$f(x) = \sqrt{1-x^2} \cdot e^{x-\frac{1}{3}}$, the domain of which is $x \in [-1,1]$

Basing on this function's formula, sequences of its values and arguments are created such that.

$$\{y_n = f(x_n) | n = 1,2,\dots,N\}, where\ x = linspace(-1, 1, N)$$

$$for\ N \in \{10, 20, 30\}$$

Task 2 involves designing a procedure for producing a least squares approximation of the function $f(x)$ from task 1. The approximations shall be produced basing on the sequences from the previous task.

$$\{(x_n, y_n) | n = 1,2,\dots N\}$$

For the purpose of approximation we shall use the Chebyshev polynomials defined by the tripart formula:

$$T_0(x) = 1,$$

$$T_1(x) = x,$$

$$T_k(x) = 2x \cdot T_{k-1}(x) - T_{k-2}(x), \text{ for } k = 2,\dots,K$$

Functions

`makeFI(K, N)` – a function that produces the FI matrix, which consists of $N \times K$ values of the Chebyshev polynomials.

$$\mathbf{\Phi} = \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_K(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \cdots & \phi_K(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_N) & \phi_2(x_N) & \cdots & \phi_K(x_N) \end{bmatrix}$$

The function at first fills the first column of the matrix with ones, then it fills the second column with corresponding values of the column vector x. If K is greater than 2, the nested for loop fills the remaining matrix columns, following the tripart formula given in the task.

`makeApp(K, N)` – parameters of this function are K for number of factors of the Chebyshev polynomial in the FI matrix and N is for the size of the column vector x. The goal is to find a vector of parameters p that makes a linear combination with a vector of linearly independent functions.

$$\mathbf{p} = \left[p_1 \dots p_K \right]^T$$

$$\left\{ \phi_k(x) \,\middle|\, k = 1, 2, \dots, K \right\}$$

$$\hat{f}(x;\mathbf{p}) = \sum_{k=1}^{K} p_k \phi_k(x)$$

For which purpose at this step we use an x vector of size 500 so that the obtained curve is smooth.

In order to obtain the vector p, we use the condition

$$\mathbf{\Phi}^T \cdot \mathbf{\Phi} \cdot \mathbf{p} = \mathbf{\Phi}^T \cdot \mathbf{y}$$

, where y is the sequence of values of the function for x defined as $x = linspace(-1, 1, N)$,

Therefore,

$$\hat{\mathbf{p}} = \left(\mathbf{\Phi}^T \cdot \mathbf{\Phi} \right)^{-1} \cdot \mathbf{\Phi}^T \cdot \mathbf{y}$$

The function uses the operator of pseudeinversion “\” implemented in MATLAB in such way.

```
p = (FI' * FI)\(FI'*y);
```

Results

For the purpose of checking the correctness of the program, there are 6 figures presented in the script.
Each figure presents:

- The original function for given N value,
- The approximation for given N and K value,
- The points on which the approximation was based.

The figures are produced for:

$$N = 10$$

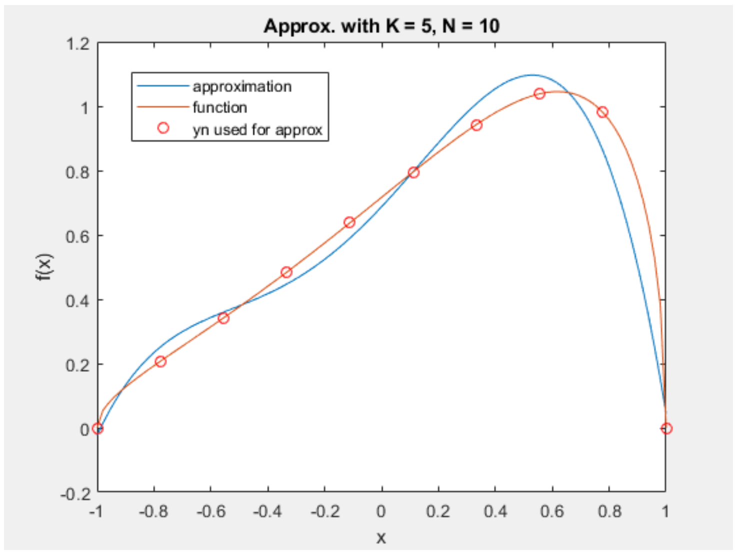

Figure 1.1

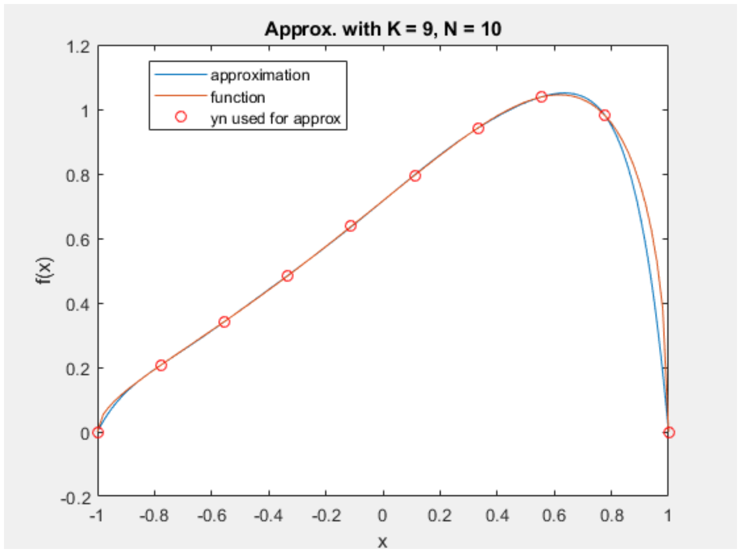

Figure 1.2

In the first case the approximation is visibly imprecise, it has actually got more curves than the original function and at some points the values differ significantly.

The figure for the same N = 10 but greater K = 9 shows, that the approximation is much closer this time, the shape is more similar along with the values being closer.

$N = 20$

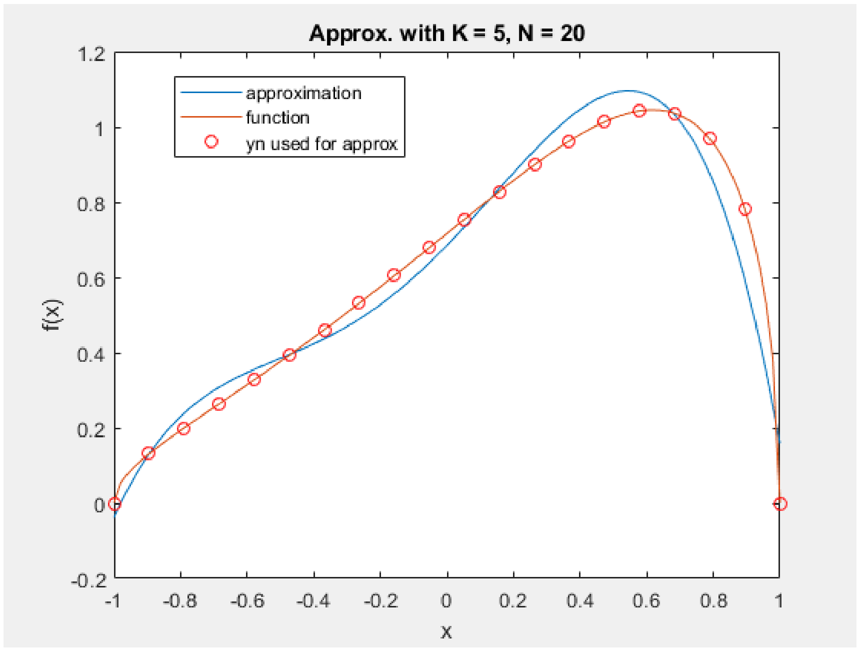

Figure 1.3

The shape of the approximation is very similar to the one from the case for N = 10, K = 5, although it is slightly closer to the original function.

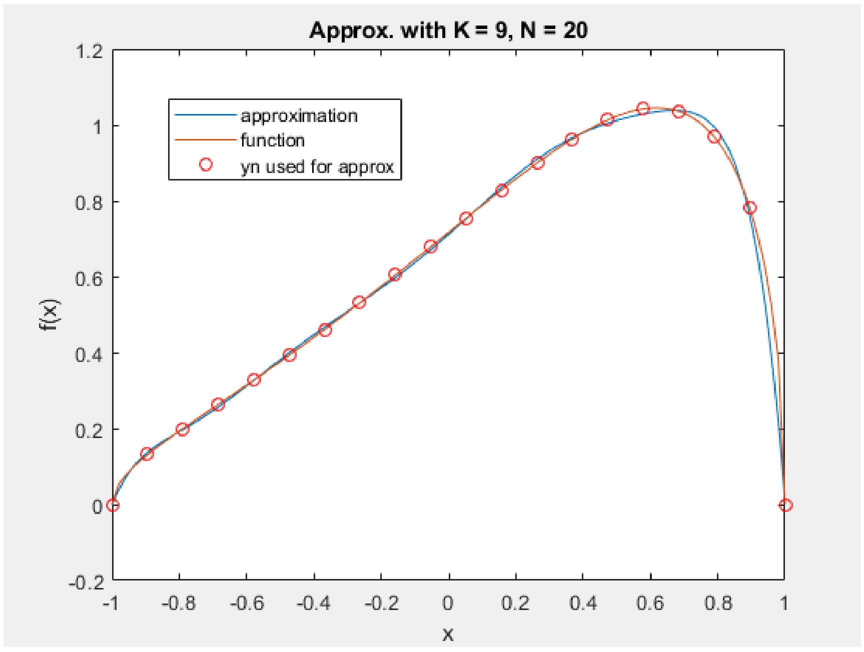

Figure 1.4

Using greater value of K = 9 again helped to make the approximation more precise, the shape is almost identical to the original function, values are close.

$N = 30$

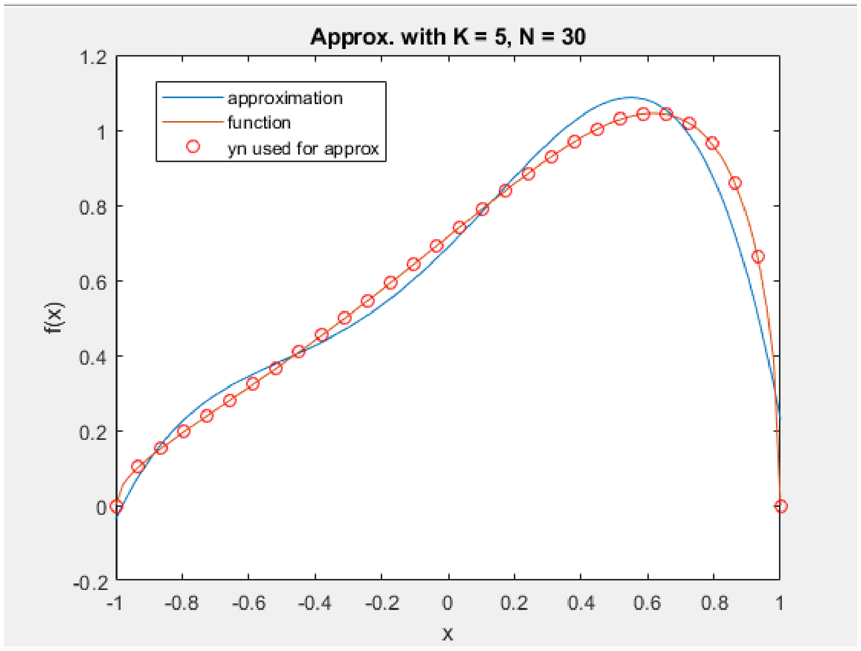

Figure 1.5

Again, as in the previous cases, where K =5 the shape of the approximation has more curves than the original function but also values are a bit closer than in the cases with smaller N value.

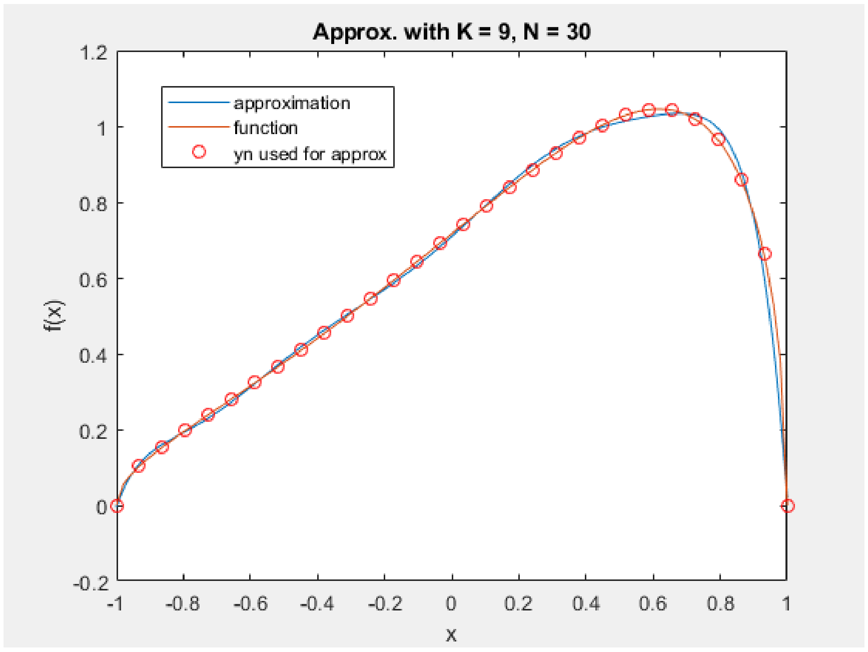

Figure 1.6

Similarly as for N = 10 and N = 20, greater value of K = 9 gave the better result, with the shape of the approximation very close to the original function and values very close.

Conclusions

Comparing the figures for several N and K values revealed that the quality of the approximation strongly depends on the number of points used (N) as well as number of polynomial factors (K). Basing on the figures obtained in this task specifically it shows that the shape of the approximation mainly depends on K value but increasing N also increases the precision of the approximation. Also in terms of the functions designed for this task, the number of points used for the sake of drawing the approximation also influences the curve (bigger number means more smooth curve) for which reason the used number of points is 500.

Task 3

Overview

Task 3 involves investigating the independence of the accuracy of approximation on the values of N and K. For that purpose two functions are introduced `delta2(K,N)` and `deltaInf(K,N)` both of which follow given formulas.

$$\delta_2(K,N) = \frac{\left\| \hat{f}(x;K,N) - f(x) \right\|_2}{\left\| f(x) \right\|_2} \quad \text{(the root-mean-square error)}$$

$$\delta_\infty(K,N) = \frac{\left\| \hat{f}(x;K,N) - f(x) \right\|_\infty}{\left\| f(x) \right\|_\infty} \quad \text{(the maximum error)}$$

Using those functions, two matrices are created, both for the values of $N \in [5,50]$ and $K \in [3, N-1]$.

Functions

`delta2(K, N)` – using MATLAB `norm` operator it follows the formula for the root-mean-square error and computes the error value for approximation with given N and K

`deltaInf(K, N)` – similarly as above it produces the maximum error for approximation with given K and N values.

Size of returned vectors for both these functions are 500 as in function from Task 2.

Results

The figures in this task are drawn using `surf()` function and matrices are scaled by logarithm of base 10 so the function calls are as shown below.

```
surf(log10(error2))
surf(log10(errorInf))
```

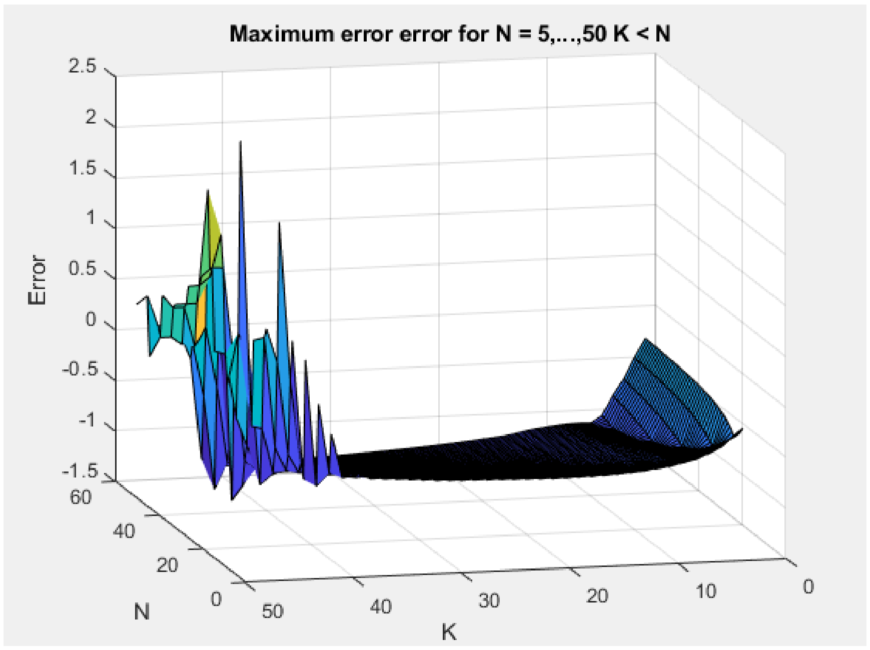

Figure 3.1

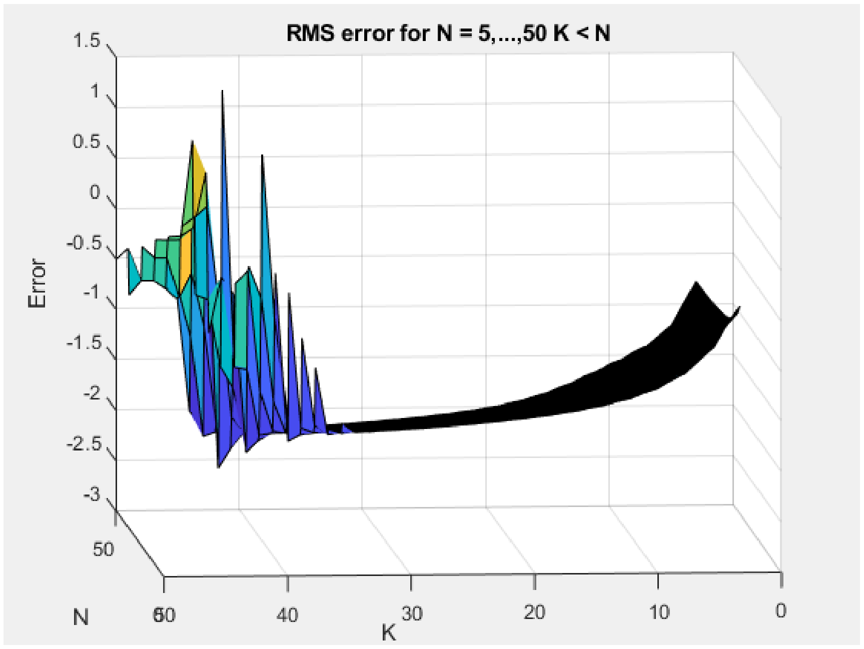

Figure 3.2

Conclusions

Comparing the figures for the root-mean-square error with the maximum error dependency on N and K shows that in both of these cases there are big distortions at the biggest values of N and K (over 40). Examining smaller values reveals that in case of the root-mean-square the surface is more curved for K > 10 which means that the values decrease more rapidly with the increase of N and K than in case of maximum error. Below K = 10 the maximum error falls more rapidly. Level of the surface in case of the maximum error is higher than in case of the root-mean-square (it's mostly between -1 and 0) which indicates that the maximum error values are in fact higher than in case of the root-mean-square.

Task 4

Overview

Vector of corrupted data $\widehat{y_n}$ is created following a formula

$\widehat{y_n} = y_n + \Delta\widehat{y_n}$, where $\widehat{y_n}$ are pseudorandom numbers following the zero-mean normal distribution with the standard deviation $\sigma_y \in [10^{-5}, 10^{-1}]$. For 20 equally spaced values from this interval root-mean-square error as well as maximum error are computed for all $N \in [5,50]$ and $K \in [3, N-1]$. For all of the above minimum values are then computed. For the obtained sequences of σ_y, maximum error and root-mean-square MATLAB operator `polyfit()` is then used in order to approximate those results.

Functions

For the purpose of this task new functions are introduced, first of which is

`errorData(N, power)`– it creates a vector $\widehat{y_n}$ with corrupted data. As parameters it takes N for number of elements and power for power to which we will raise 10 in standard deviation. It uses matlab operator $randn$ for generating pseudorandom numbers.

`errorApp(K, yn)`– works similarly to `makeApp(K, N)` from Task 2 but instead of N it takes y_n to make an approximation basing on the corrupted data vector.

`corrupted2(K, yn)` – analogically it produces root-mean-square error for approximation with given K for the corrupted data vector y_n

`corruptedInf(K, yn)`–similarly as in Task 2 it computes the maximum error for approximation with given K but for the corrupted data vector y_n

Results

After fitting the sequences and their values using `polyfit()` and `polyvalue()` operators with polynomials of 5 degree for root-mean-square and degree 2 for maximum-error, two figures are made.

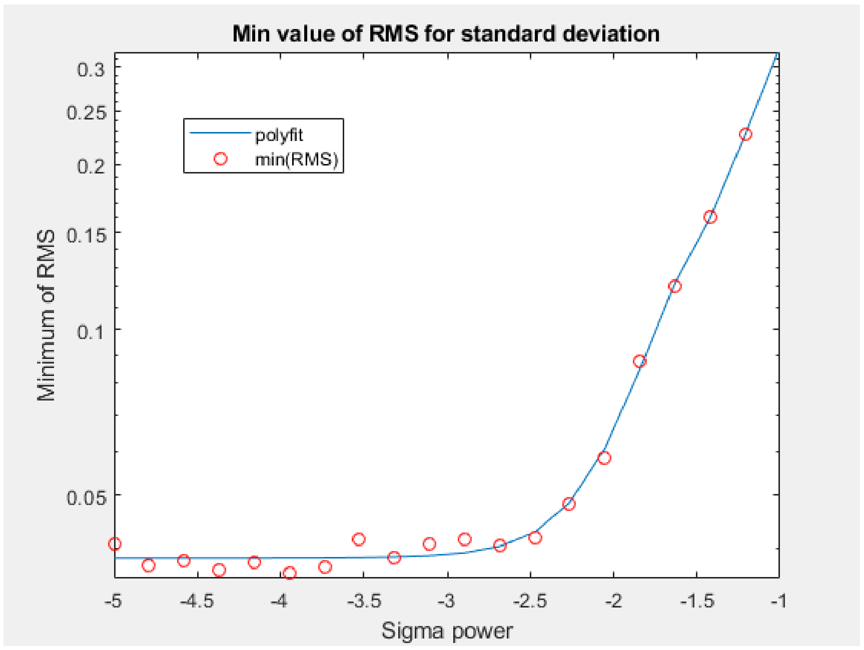

Figure 4.1

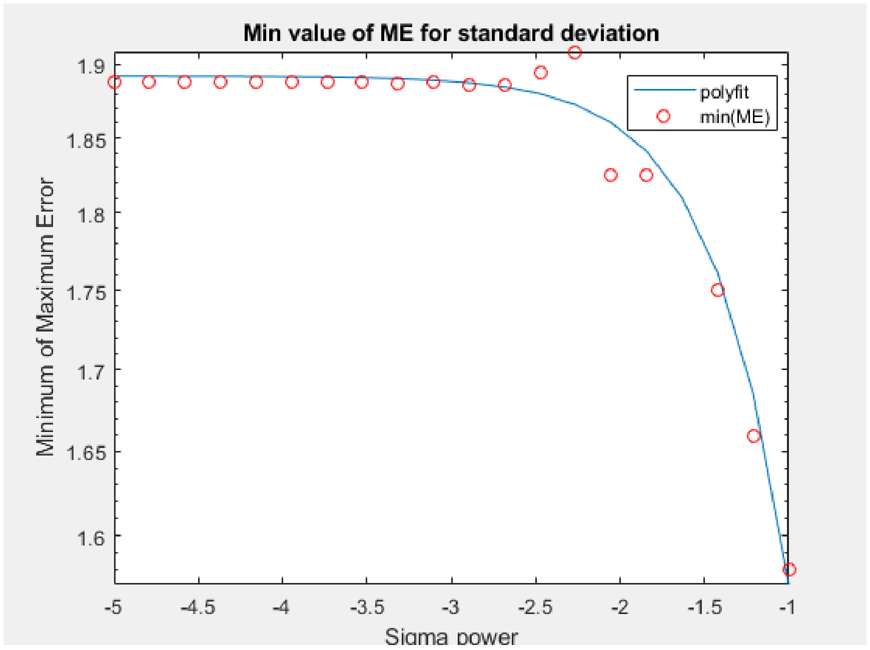

Figure 4.2

For some reason despite of using `semilogy()` function for plotting the figures, the second figure for the maximum error is scaled linearly in the y-axis. Both figures are scales linearly on the x-axis which is the power to which we raise 10 in the standard deviation. The minimum values of the errors are computed using MATLAB `min()` operator.

Conclusions

The best fit for the curves were obtained for degree 5 polynomial in case of root-mean-square and a polynomial of degree 2 in the maximum error. In both cases it still is not perfect though. There are some critical points for which the curve bends away from the values of the minimum. This might be caused with the distortions seen in Task 3 for the high values of N and K.

References

- https://www.mathworks.com/help/matlab/
- Roman Z. Morawski – Numerical Methods (ENUME) Lecture notes for Spring Semester 2018/2019