NUMERICAL METHODS (ENUME 2019) – PROJECT ASSIGNMENT B: APPROXIMATION OF FUNCTIONS

Ernest Pokropek, 293076

Ernest Pokropek
Warsaw, 14/05/2019

Table of contents

I. Introduction

Approximation of functions is one of the most important topics covered in whole applied mathematics. It enables to depict the function knowing only some discrete points on the given plane. Here, the method of **least-squares function approximation** is to be discussed.

Least squares function approximation applies the principle of least squares to function approximation, by means of a weighted sum of other functions. The best approximation can be defined as that which minimises the difference between the original function and the approximation; for a least-squares approach the quality of the approximation is measured in terms of the squared differences between the two. Two types of least squares algorithm are to be distinguished: linear and nonlinear, whereas linear version aims at providing a one straight line which passes through most of the points on x-y graph, the nonlinear one aims to follow as closely as it can to the inflections and turns of a given function.

In this particular case, the non-linear approximation shall be performed using **B-spline functions** (called sometimes basis splines). The B-spline function is a spline function that has minimal support with respect to a given degree, smoothness, and domain partition. Any spline function of given degree can be expressed as a linear combination of B-splines of that degree.

II. Methodology

The given function (from which the indicated sequence of its values will be used for approximation) has the following form:

$$f(x) = \left(x + \frac{1}{3}\right)^2 + e^{-x-2} \text{ for } x \in [-1,1]$$

B-splines are given as follows:

$$Bs_k(x) = B_S\left(2(x - x'_k) + 2\right) \quad B_S(x) = \begin{cases} x^3 & x \in [0,1) \\ -3(x-1)^3 + 3(x-1)^2 + 3(x-1) + 1 & x \in [1,2) \\ 3(x-2)^3 - 6(x-2)^2 + 4 & x \in [2,3) \\ -(x-3)^3 + 3(x-3)^2 - 3(x-3) + 1 & x \in [3,4] \end{cases}$$

$$\text{where } x'_k = -1 + 2\frac{k-1}{K-1} \quad \text{for } k = 1, 2, \ldots, K$$

Concerning the vector of parameter p = [p1, … pk] which is sought for, the vector that makes the linear combination of linearly independent functions { ϕ$_k$(x) | k = 1,2,…, K}:

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

Then, the necessary condition of the minimum has the form:

$$\frac{\partial J_2(\mathbf{p})}{\partial p_k} = 0 \quad \text{for} \quad k = 1, \ldots, K \quad \Leftrightarrow \quad \mathbf{\Phi}^T \cdot \mathbf{\Phi} \cdot \mathbf{p} = \mathbf{\Phi}^T \cdot \mathbf{y}$$

where:

$$\mathbf{\Phi} = \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_K(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \cdots & \phi_K(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_N) & \phi_2(x_N) & \cdots & \phi_K(x_N) \end{bmatrix} \text{ and } \mathbf{y} = \left[f(x_1) \; f(x_2) \; \ldots \; f(x_N) \right]^T$$

Thus finally, the solution may be presented in following form:

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

III. Results

The approximated function, nodes and original functions are presented on figures 1-3, with different pairs of parameter N and K (where N is the number of nodes). For every figure of given N, there are 3 changes in parameter K (captions *a*, *b*, and *c*).

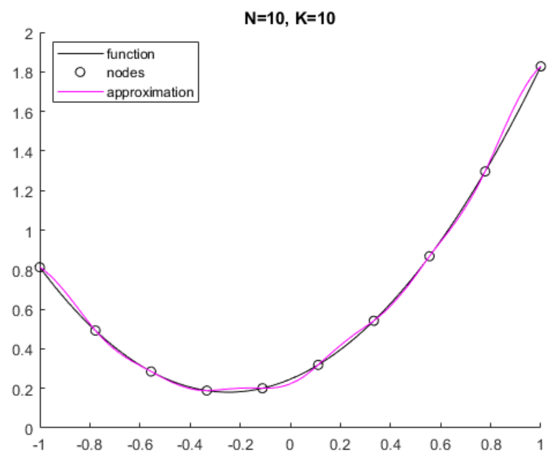

Fig. 1a)

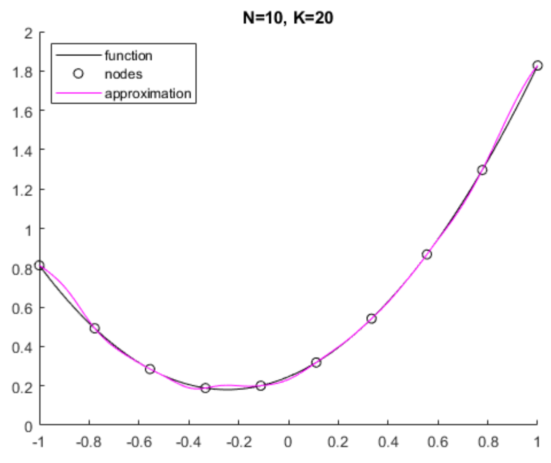

Fig. 1b)

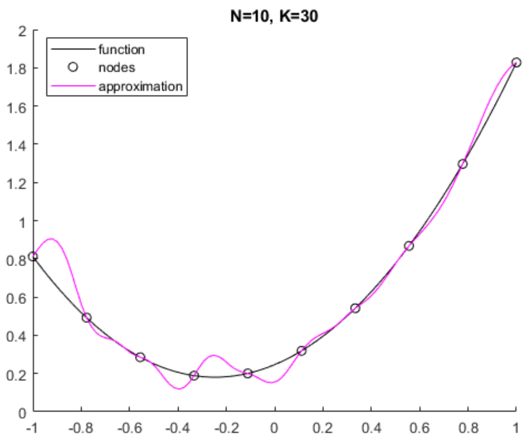

Fig. 1c)

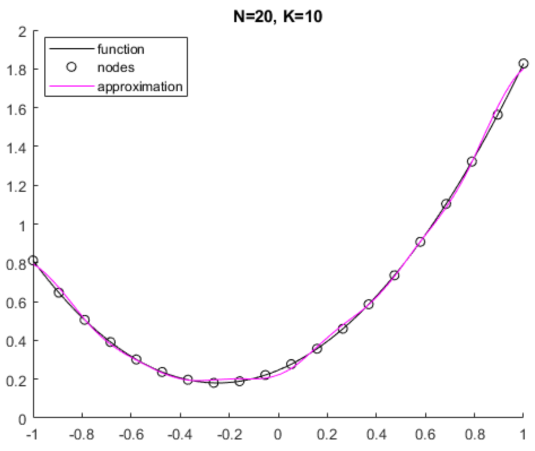

Fig. 2a)

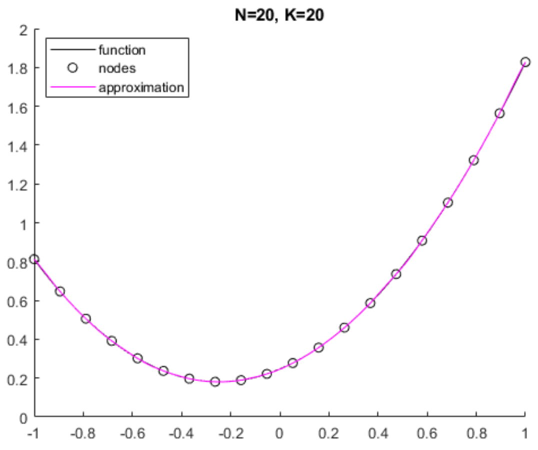

Fig. 2b)

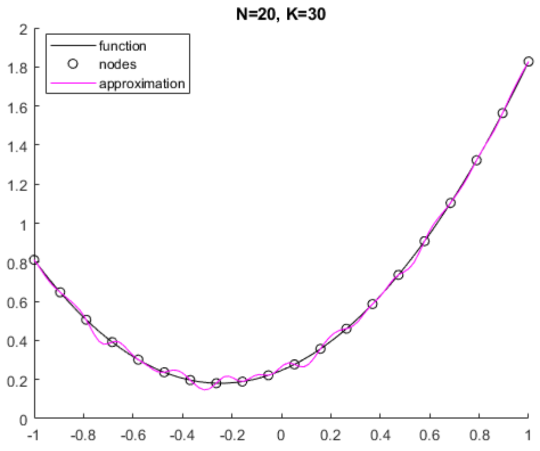

Fig. 2c)

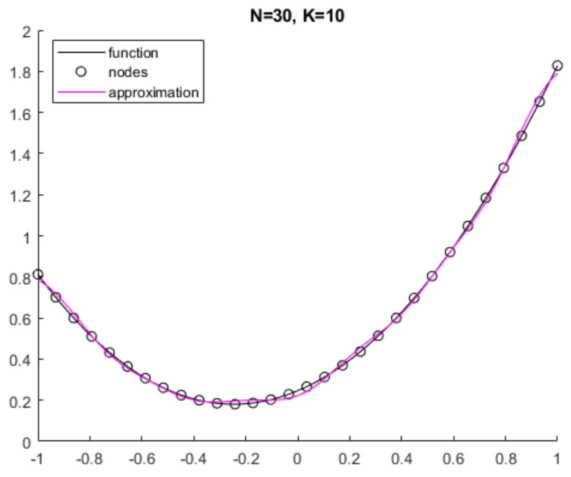

Fig. 3a)

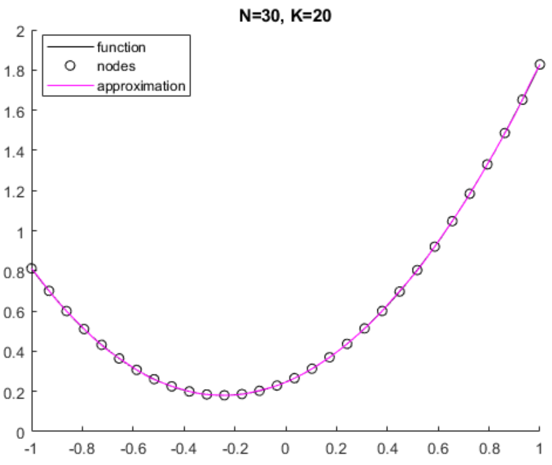

Fig. 3b)

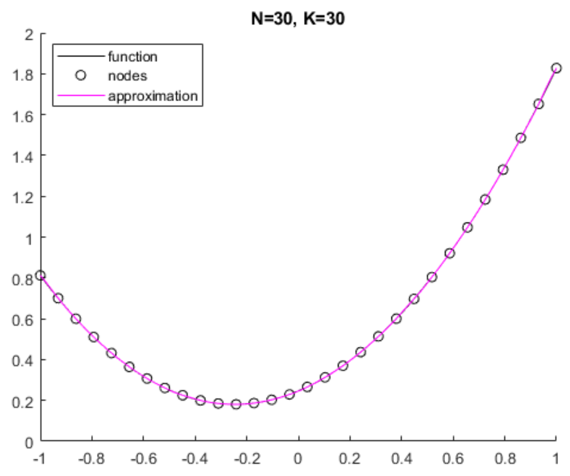

Fig. 3c)

From the figures two scenarios of bad approximation can be depicted:

- Poor fit, caused by small values of parameter K compared to number of nodes (K << N) [Figures 1b), 2a), 3a)], or nodes being generally too dispersed [Figure 1a)]
- Overfit, caused by too big parameter K compared to number of nodes (N >> K) [Figures 1b), 1c), 2c)]

The best approximation results were obtained using parameters K and N being close to each other in their magnitudes.

Furthermore, the systematic investigation of the dependence of the accuracy of approximation on values N and K was carried, using two following accuracy indicators:

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

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

Where f(x; K, N) is an approximating function obtained for N and K. The results of the investigation are presented on figures 2a) and 2b).

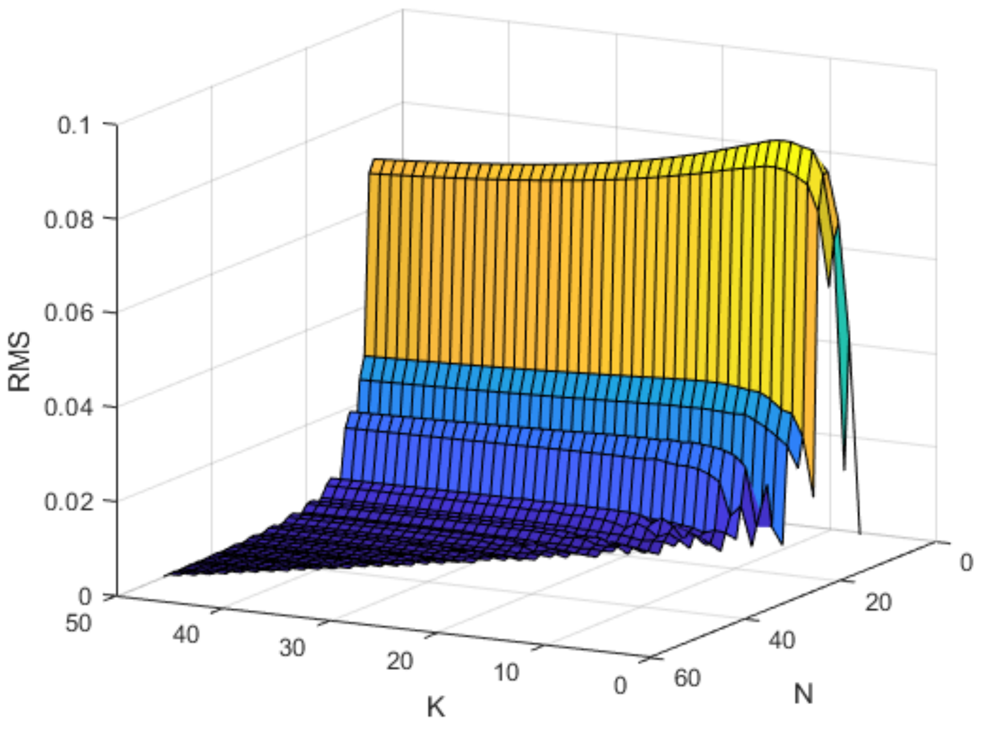

Fig. 2a) Root-mean-square error (RMS) value on parameters K and N

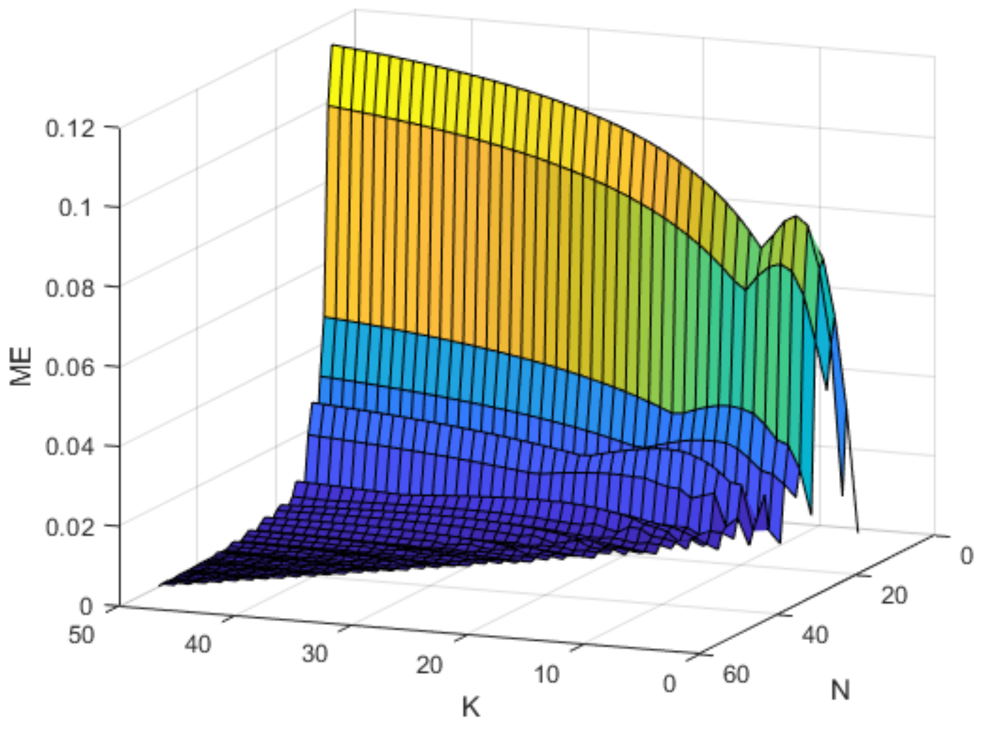

Fig. 2b) Maximum error (ME) value on parameters K and N

From figures 2a) and 2b) it can be depicted, that both measures behave similarly for very large number of nodes (N>20), as the error values get smaller in their magnitudes. The most visible change in the error's behaviour is visible for N϶[0, 20], as the RMS value is almost stable for various parameters of K, while the ME visibly changes. Furthermore, RMS achieves its maximum value approximately in such N and K, that ME reaches its minimum, for common parameter N.

To simulate approximation method on real data, the systematic investigation of the dependence of the indicators (norms) on the approximation was inspected. Points were corrupted according to the formula:

$$\tilde{y}_n = y_n + \Delta\tilde{y}_n \text{ for } n = 1, \ldots, N$$

where Δy_n are pseudorandom numbers following the zero-mean normal distribution with the variance σ^2 ($\sigma\epsilon[10^{-5}, 10^{-1}]$), obtained by means of the MATLAB operator ***randn.*** For each value of the standard deviation, N and K values minimising norms were determined, and then pairs of form (σ, norm(K,N)) were used to approximate the sequence using MATLAB operator ***polyfit***. The results of the described actions are presented on figures 3a) and 3b).

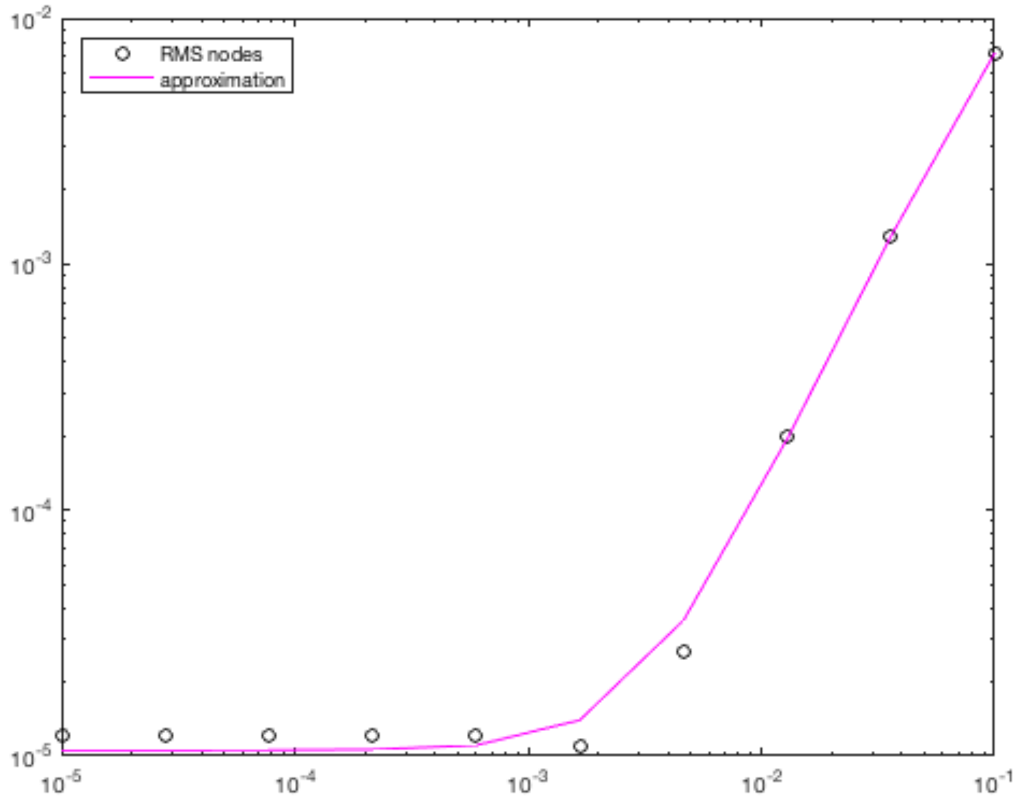

Fig. 3a) RMS on random sigma relation

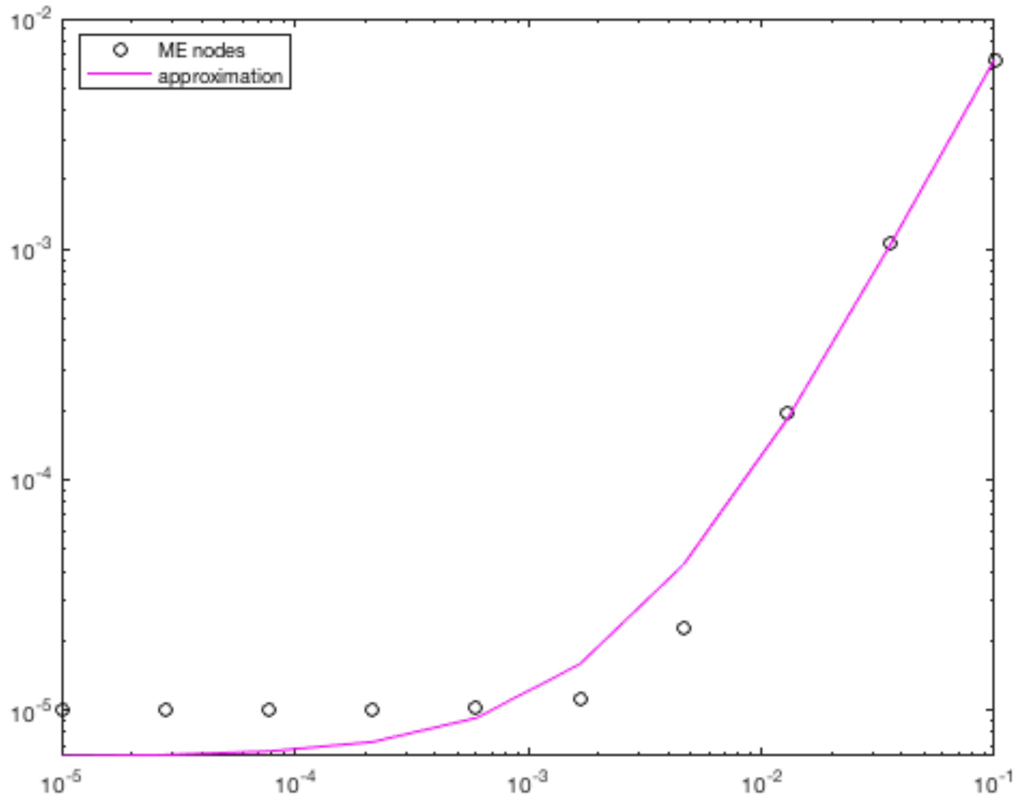

Fig. 3b) ME on random sigma relation

From the Fig. 3a) and Fig. 3b) it can be depicted, that approximation performs well for data corrupted with standardized errors. For the larger values of data corruption, the approximation error also enlarges.

IV. Conclusions

Least squares approximation is one of the most commonly used approximation methods. Its advantages are:

- Simplicity: It is very easy to explain and to understand
- Applicability: There are hardly any applications where least squares doesn't make sense
- Theoretical Underpinning: It is the maximum-likelihood solution and, if the Gauss-Markov conditions apply, the best linear unbiased estimator

Meanwhile, the disadvantages present as:

- Sensitivity to outliers
- Test statistics might be unreliable when the data is not normally distributed (but with many datapoints that problem gets mitigated)
- Tendency to overfit data (LASSO or Ridge Regression might be advantageous)

Concluding, the least square approximation for non-linear functions is as good as the one using it, making it a great tool in hands of a skilled person.

V. Bibliography

[1] Roman Z. Morawski - ENUME 2019 – Lecture notes
[2] Jerzy Krupka, Andrzej Miękina, Roman Z. Morawski, Leszek J. Opalski – Wstęp do metod numerycznych dla studentów elektroniki i technik informacyjnych, praca zbiorowa pod redakcją Romana Z. Morawskiego
[3] Uri M. Ascher, Chen Greif – A First Course on Numerical Methods

VI. Appendix

MATLAB R2018b

```
clear
close all

% Tasks 1 and 2 - approximation of function

% initial setup
nodes_num=100;
approx = zeros(1,nodes_num);
x = linspace(-1,1,nodes_num);

i = 1;
for N=10:10:30
    for K=10:10:30
        y_t = zeros(1,N);
        index=N/10;

        % performing approximation
        for s=1:step
            approx(s)=approximate(x(s),N,K);
        end
        figure(i)
        i = i+1;
        hold on;
        % original function
        plot(x, f(x), 'k');

        % nodes
        x_n=generate_xn(N);
        y=generate_y(x_n);
        plot(x_n, y,'ko')

        % approximated function
        plot(x, approx,'m')
        title(strcat('N=', num2str(N),', K=', num2str(K)));
        legend('function','nodes', 'approximation');
        hold off
    end
end

% Task 3 - Systematic investigation of the dependence of the accuracy
% of approximation on the values of N and K

[N,K]=meshgrid(5:50,5:50);
rms = nan(length(N));
me = nan(length(N));

for k=4:50
```

```
    % Satisfying the given K < N condition
    for n=k+1:50
        rms(n-4,k-3)= RMS(n,k);
        me(n-4,k-3)= ME(n,k);
    end
end


figure(10)
surf(K, N, rms);
xlabel("K")
ylabel("N")
zlabel("RMS")
figure(11)
surf(K, N, me);
xlabel("K")
ylabel("N")
zlabel("ME")

% Task 4 - systematic investigation of the dependence of norms on the
% standard deviation of random errors

% initial setup
N_const = 20;
nodes_num = 10;

n_min_rms = zeros(1,nodes_num);
k_min_rms = zeros(1,nodes_num);

n_min_me = zeros(1, nodes_num);
k_min_me = zeros(1, nodes_num);

[N,K] = meshgrid(1:N_const,1:N_const);
space = logspace(-5,-1,nodes_num);

roms = nan(N_const);
roms_min = ones(1, nodes_num);
mes = nan(N_const);
mes_min = ones(1, nodes_num);

% performing the investigation
for i=1:nodes_num
    for n=5:N_const
        for k=5:N_const
            if(k<n)
                % computing errors on corrupted data
                roms(n-4, k-4) = RMS_corrupt(n, k, space(i));
                mes(n-4, k-4) = ME_corrupt(n, k, space(i));
                % selecting pairs
                if(roms(n-4,k-4)<roms_min(i))    % RMS
                    roms_min(i)=roms(n-4,k-4);
                    n_min_rms(i)=n;
                    k_min_rms(i)=k;
                end
                if(mes(n-4,k-4)<mes_min(i))      % ME
                    mes_min(i)=mes(n-4,k-4);
                    n_min_me(i)=n;
                    k_min_me(i)=k;
                end

            else
                continue;
            end
        end
    end
end
```

```
% fitting the data
p_rms = polyfit(space, roms_min, 3);     % RMS
RMSE_map = polyval(p_rms, space);

p_me = polyfit(space, mes_min, 3);       % ME
ME_map = polyval(p_me, space);

% plotting
figure % RMS
loglog(space, roms_min, "ko");
hold on
loglog(space, RMSE_map,'m');
legend("RMS nodes", "approximation")
hold off

figure % ME
loglog(space, mes_min, "ko");
hold on
loglog(space, ME_map,'m');
legend("ME nodes", "approximation")
hold off



% ----------- FUNCTION DEFINITIONS -------------


% approximation of corrupted data
function y = approx_corrupt(x, N, K, sigma)
    phi = generate_phi(N,K);
    x_n = generate_xn(N);
    y = generate_y_corrupted(x_n,sigma);
    p = phi.' * phi\phi.' * y.';
    y = 0;
    K = length(p);
    for i=1:K
        y = y + p(i) * Bsk(x,i,K);
    end
end

% root mean square error of corrupted data
function y = RMS_corrupt(N,K,sigma)
    nom = zeros(1,N);
    denom = zeros(1,N);
    x_n=generate_xn(N);
    for i=1:N
        nom(1, i) = approx_corrupt(x_n(i), N, K, sigma) - f(x_n(i));
        denom(1, i) = f(x_n(i));
    end
    y= norm(nom) / norm(denom);
end

% maximum error of corrupted data
function y = ME_corrupt(N,K,sigma)
    nom = zeros(1,N);
    denom = zeros(1,N);
    x_n = generate_xn(N);
    for i=1:N
        nom(1, i) = approx_corrupt(x_n(i), N, K, sigma) - f(x_n(i));
        denom(1,i) = f(x_n(i));
    end
    y= norm(nom,Inf) / norm(denom, Inf);
end

% generation of corrupted y
```

```
function y=generate_y_corrupted(x, sigma)
    N = length(x);
    yn = zeros(1,N);
     for n=1:N
       x_n = -1+ 2*(n-1) / (N-1);
       yn(n) = f(x_n) + randn()*sigma^2;
     end
    y=yn;
end

% Root-mean-square error (2 norm)
function y = RMS(N, K)
    nom = zeros(1, N);
    denom = zeros(1, N);
    x_n = generate_xn(N);

    for i=1:N
        nom(1, i) = approximate(x_n(i), N, K)-f(x_n(i));
        denom(1, i) = f(x_n(i));
    end

    y=norm(nom) / norm(denom);
end

% Maximum error (infinity norm)
function y = ME(N, K)
    nom = zeros(1, N);
    denom = zeros(1, N);
    x_n = generate_xn(N);

    for i=1:N
        nom(1, i) = approximate(x_n(i), N, K) - f(x_n(i));
        denom(1, i) = f(x_n(i));
    end

    y= norm(nom, Inf) / norm(denom, Inf);
end

% generation of phi for approximation
function y = generate_phi(N,K)
    phi = zeros(N,K);
    for n=1:N
        x_n = -1+2*(n-1)/(N-1);
        for k=1:K
            phi(n, k) = Bsk(x_n,k,K);
        end
    end
    y = phi;
end

% generate y coordinates of the function
function y = generate_y(x)
    N = length(x);
    yn = zeros(1, N);
     for n=1:N
       x_n = -1+2*(n-1)/(N-1);
       yn(n) = f(x_n);
     end
    y = yn;
end

% generate x coordinates of the function
function y = generate_xn(N)
    x_n = zeros(1,N);
    for n=1:N
       x_n(n) = -1 + 2*(n-1) / (N-1);
    end
    y=x_n;
end

% calculating xk points in given B-splines
function y = Bsk(x, k, K)
    xk = -1 + 2*((k-1) / (K-1));
    y = Bs(2 * (x - xk) + 2);
end

% approximating function
function y = approximate(x, N, K)
    Fi = generate_phi(N, K);
    x_n = generate_xn(N);
    y = generate_y(x_n);
    p = Fi.' * Fi \ Fi.' * y.';
    y=0;
    K=length(p);
    for i=1:K
        y=y + p(i) * Bsk(x, i, K);
    end
end

% initial function
function y = f(x)
    y = (x+(1/3)).^2 + exp(-x-2);
end

% B_spline functions definition
function y=Bs(x)
    % x [0, 1)
    if (x>=0 && x<1)
        y = x^3;
    % x [1, 2)
    elseif (x>=1 && x<2)
        y = -3*((x-1)^3) + 3*((x-1)^2) + 3*(x-1) + 1;
    % x [2, 3)
    elseif (x>=2 && x<3)
        y = 3*((x-2)^3) - 6*((x-2)^2) + 4;
    % x [3, 4]
    elseif (x>=3 && x<=4 )
        y = -((x-3)^3) + 3*((x-3)^2) - 3*(x-3) + 1;
    % x doesn't belong to [0, 4]
    else
        y=0;
    end
end
```