Aschaffenburg University of Applied Sciences
Faculty for Engineering
Prof. Dr. Michael Möckel

Email: Michael.Moeckel@th-ab.de
Telephone: +49 (0)6021 / 4206 - 507

Numerical Methods and Simulation
Module MT 03
Winter term 2022/23

https://moodle.th-ab.de/course/view.php?id=624

Problem Set 8

Interpolation and Regression II

Coursework

Exercise 1: Regression with different functions

The least square method minimizes the quadratic difference between the approximately fitted function $f(x)$ and the data points (x_i, y_i).

$$E = \sum_{i=1}^{n} [y_i - f(x_i)]^2$$

The function $f(x) = \sum_j a_j g_j(x)$ can be represented by a linear combination of basis functions $g_j(x)$. A Vandermonde matrix A can be set up. If the number of basis functions is smaller than the number of data points the Vandermonde matrix is not a square matrix and its rank is smaller than n.

Theorem: The minimization of the error is achieved by the coefficient vector $\vec{a}$ iff a normal condition holds for the Vandermonde matrix A, the coefficient vector $\vec{a}$ and the vector of the data points $\vec{y}$ such that

$$A^T A\vec{a} = A^T\vec{y}$$

This *normal condition* (dt Normalengleichung) can be used to solve well-conditioned regression problems efficiently. Unfortunately, it deteriorates the conditioning of the problem from cond(A) to cond(A)^2. For badly conditioned problems other approaches e.g. based on orthogonal matrices are available.

Consider the following data points:

x	1/6	2/6	3/6	4/6	5/6	1
y	1	4	0	-3	3	1

Fit the following functions by setting up the non-square generalized Vandermonde matrix A and solve the normal condition:

(1) $f_1(x) = a_0 + a_1 x + a_2 x^2$

(2) $f_2(x) = b_0 + b_1 \cos(2\pi x) + b_2 \sin(2\pi x)$

(3) $f_3(x) = c_0 + c_1 \cos(2\pi x) + c_2 \sin(2\pi x) + c_3 \cos(4\pi x)$

Exercise 2: Polynomial regression (difficult)

In the least squares approach to linear regression a linear function is fitted to a set of data points such that the squared error function becomes minimal (see Problem Set 7, Exercise 3).

In an analogous way a higher order polynomial can be fitted to the set of data points. Generalize the linear regression example to a polynomial of order n. This requires the optimization of $n+1$ parameters (the coefficients of the polynomial) and leads to a system of $n+1$ coupled linear equations. Sums of powers of data point coordinates (x_i, y_i) are required for all even powers up to $\sum_i x^{2n}$.

Write a MATLAB function which returns a vector $a = [a_0, ...a_n]$ with all polynomial fitting parameters up to a given order n by setting up the system of coupled linear equations and solving it using the \operator.

Exercise 3: Interpolation of data with splines

An alternative to interpolation of n data points $(x_1, y_1)\ \ldots\ (x_n, y_n)$ with a polynomial of high order is based on using piecewise defined polynomials of lower order, so-called *splines*. A spline interpolation $\mathcal{S}(x)$ can be written as a sum of functions which are nonzero only between two data points $\mathcal{S}(x) = \sum_i s_i(x)$ with $s_i(x) = 0\ \forall x \notin [x_i, x_{i+1}]$.

(1) **Linear splines.** The simplest 'spline' connects all data points with lines. Write a little MATLAB function `LinearSpline.m` which calculates the parameters $a_0(i)$ and $a_1(i)$ for all lines $g_i(x) = a_1(i)x + a_0(i)$ connecting the data points (x_i, y_i) and (x_{i+1}, y_{i+1}) and plot the data as well as the connecting lines on their respective intervals. Note that linear splines normally lead to discontinuities of the derivative at the data points.

(2) **Quadratic splines.** Quadratic splines use second order polynomials of the form $h_i(x) = a_2(i)x^2 + a_1(i)x + a_0(i)$. For n data points there are (n-1) splines needed which implies that 3(n-1) parameters need to be fixed. This is done by the following constraints:

   (i) $y_i = h_i(x_i)$ (data point = function value at the first point of interval)

   (ii) $y_{i+1} = h_i(x_{i+1})$ (at the last point of interval). The conditions (i) and (ii) give 2(n-1) constraints.

   (iii) At the interior data points, the derivatives of two splines are matched: $h_i'(x_{i+1}) = h_{i+1}'(x_{i+1})$. This gives another $n-2$ conditions and ensures continuity of the first derivative.

   (iv) The final condition can be set by fixing a boundary condition for one of the outer data points (e.g. $h_1''(x_1) = 0$).

(3) **Cubic splines.** A plausible further demand is to expect continuity of both the first and the second derivative at all (n-2) interior data points. This form of a spline is used in most cases. It consists of cubic polynomials $f_i(x) = a_0(i) + a_1(i)x + a_2(i)x^2 + a_3(i)x^3$ with 4(n-1) parameters. The following conditions are used to determine the parameters:

   (i) $y_i = f_i(x_i)$

   (ii) $y_{i+1} = f_i(x_{i+1})$

   (iii) At the interior data points, the first derivatives of two splines are matched: $f_i'(x_{i+1}) = f_{i+1}'(x_{i+1})$. This gives another $n-2$ conditions and ensures continuity of the first derivative.

   (iv) At the interior data points, the second derivatives of two splines are matched: $f_i''(x_{i+1}) = f_{i+1}''(x_{i+1})$. This gives another $n-2$ conditions and ensures continuity of the second derivative.

   (v) At the edge, different boundary conditions can be chosen to determine the remaining parameters. *Natural* or *simple* boundary conditions are often used and defined by $f_1''(x_0) = f_n''(x_n) = 0$. Alternatively, *clamped* boundary conditions are sometimes used ($f_1'(x_0) = f_n'(x_n) = 0$). MATLAB uses *not-a-knot* boundary conditions defined by matching the third derivative at the two points x_2 and x_{n-1}, i.e. $f_1'''(x_2) = f_1'''(x_2)$ and $f_{n-2}'''(x_{n-2}) = f_{n-1}'''(x_{n-2})$. This translates into $a_3(1) = a_3(2)$ and $a_3(n-2) = a_3(n-1)$.

(4) Consider the points $(x_0, y_0) = (0, 0)$, $(x_1, y_1) = (1, 4)$, $(x_2, y_2) = (2, 3)$. Construct a cubic spline interpolation by calculating the parameters a_i, b_i, c_i, d_i $(i = 1, 2)$ for the two pieces f_1 and f_2 of the piecewise defined function

$$\mathcal{S}(x) = \begin{cases} f_1(x) & x_0 \le x \le x_1 \\ f_2(x) & x_1 \le x \le x_2 \end{cases}$$

using natural boundary conditions. Set up a system of 8 coupled linear equations from the conditions (i) - (v) and solve the system with MATLAB.

(5) MATLAB provides a variety of ways to calculate splines. One way is to call the command `spline`. Run `x = 0:10; y = sin(x); xx = 0:.25:10; yy = spline(x,y,xx); plot(x,y,'o',xx,yy);` Plot the sin function with higher reolution and compare with the spline fit. Also plot the difference of the spline fit and the function.