Humboldt Universität zu Berlin

Numerical Introductory Course

INTERPOLATION AND NUMERICAL APPROXIMATION

Student:
Urška Bele

Supervisor:
Prof. Dr. Brenda López Cabrera

Berlin, July 2018

Contents

Abstract

In the paper I have looked at the interpolation and approximation of a set of data points using polynomials. Interpolation polynomial passes through all the data points. I described Lagrange, piecewise linear, cubic spline, B-spline and Hermite interpolation. Approximating polynomial may not pass through any of the data points but in some sense is the best fit. We measure the goodness of fit with a norm. I described Taylor approximation, which is based on one point only. Least maximum approximation minimizes the L_∞ norm and least square approximation minimizes the L_2 norm. I compared the advantages and disadvantages of the methods, found some examples of applications and the improvements with adaptive algorithms. At the end I ran a simulation to find out how the error depends on the number and position of given points. I also used the methods to derive the interpolated yield curve based on the data about yield rates of U.S. Treasury Securities.

1 Introduction

Regardless one's age or investing goals, it's a good idea to have at least a small percentage of portfolio in bonds. They represent a steady, safe and guaranteed source of income. Investments that are considered the safest in the world are Treasury securities.

U.S. Treasury securities (i.e. T-Bills, T-Notes and T-Bonds) are debt obligations of the U.S. government. When an investor buys one, he is lending money to the federal government for a specified period of time. When they are first issued, Treasury securities can be purchased directly from the Federal Reserve (e.g. online at auction). After that, they can be traded with at the secondary bond market. T-Bills are sold with maturities of 1, 3, 6 and 12 months, T-Notes with maturities of 2, 3, 5, 7 and 10 years and T-Bonds with maturities of 20 and 30 years. [24]

Before making an investment, investors are interested in yield curve of the bonds. A yield curve is a curve on a graph in which the yield of fixed-interest securities is plotted against the length of the time they have to run to maturity. It is used as a benchmark for other debt in the market, such as mortgage rates or bank lending rates, and it is also used to predict changes in economic output and growth. [2]

Non-Treasury (corporate) bonds are generally evaluated based on the difference between their yield and the yield on a Treasury bond of comparable maturity (yield spread). The problem arises because we can only find information about yield rate of Treasury bonds for certain maturities. For example, we are able to find the yield for a 1-year security, but not a 1.5-year security. [5]

A possible solution to this is numerical interpolation. It is a method used to approximate unknown values based on available information. Since the Treasuries only have specific maturities, the yield of those that are located between the treasuries needs to be interpolated. Using interpolation we derive the interpolated yield curve (I-curve). Based on that we can estimate the yield spread for non-treasury bonds with maturities that are different from maturities of Treasury securities. The described problem is shown on figure 17 in Appendix. [22]

2 Interpolation Methods

We will look at the approximation of a set of data points using a polynomial. The first possibility is determining a polynomial that passes through all the data points, this is called interpolation. The second posibility is determining a polynomial that may not pass through any of the data points but in some sense is the best fit. [17]

2.1 Polynomial Interpolation

Suppose we are given values of function f in $n+1$ different points $x_0, x_1, ..., x_n$ on the interval $[a, b]$. In general, we are looking for a simpler interpolation function g which satisfies the condition:

$$f(x_i) = g(x_i), \quad i = 0, ...n.$$

Interpolation is useful when we are only given a tabulated function f, but looking for a value at point $x \in (x_0, x_n)$ which is not given in a table. In this case we can use the approximation $f(x) \approx g(x)$.

The main idea of polynomial interpolation is the following: we are given $n+1$ points $(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)$, which we refer to as knots. We refer to $x_0, x_1, ..., x_n \in [a, b]$ as nodes and assume that $x_0 < x_1 < ... < x_k < x_{k+1} < ... < x_n$. No restrictions are imposed on y_k-s. Our goal is to find a polynomial p of degree n or lower that interpolates the knots: $p(x_i) = y_i$ for all $i = 0, ..., n$. We use polynomials because of Weierstrass theorem, which says that any function over a finite range can be approximated arbitrarily well by a polynomial. They are also easy to evaluate, differentiate and integrate. It can be proven that there exists a unique polynomial p of degree n that passes through the given set of $n+1$ knots. It's form is:

$$p(x) = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n.$$

[17]

The simplest case of polynomial interpolation is linear interpolation, that is the case when $n = 1$. Only two knots $P_0(x_0, y_0)$ and $P_1(x_1, y_1)$ are given. We can interpolate these two knots by a linear function with the equation:

$$p(x) = y_0 + \frac{y_1 - y_0}{x_1 - x_0}(x - x_0).$$

Two important forms of expressing the interpolation polynomials of higher degrees are Newton's and Larange's form. We will take a closer look at the latter. First we define Lagrange basis polynomials:

$$l_{i,n}(x) := \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} = \frac{(x - x_0) \cdot ... \cdot (x - x_{i-1}) \cdot (x - x_{i+1}) \cdot ... \cdot (x - x_n)}{(x_i - x_0) \cdot ... \cdot (x_i - x_{i-1}) \cdot (x_i - x_{i+1}) \cdot ... \cdot (x_i - x_n)}, \quad i = 0, ..., n.$$

[8]

The **interpolation polynomial in the Lagrange form** is a linear combination of Lagrange basis polynomials:

$$L_n(x) = \sum_{i=0}^{n} y_i \cdot l_{i,n}(x).$$

A simple example of Lagrange interpolation on 6 nodes is shown on figure 18 in Appendix. [8]

When interpolating a given function f by a polynomial of degree n at the nodes $x_0, x_1, ..., x_n$ we get the remainder $R_n(x)$ which can be expressed as:

$$R_n(x) = f(x) - L_n(x) = \omega(x) \cdot \frac{f^{(n+1)}(\xi_x)}{(n+1)!},$$

where $\omega(x) = (x - x_0) \cdot (x - x_1) \cdot ... \cdot (x - x_n)$ and $\xi_x \in (x_0, x_n)$. [8]

Algorithm for construction of Lagrange interpolation polynomial:

```
INPUT: $x_i, y_i, \quad i = 0, ..., n$
Set $s$ to be the all ones vector of length $n + 1$.
for $i = 0$ to $n$:
    for $j = 0$ to $n$:
        If $i \neq j$ then $s[i] = s[i] \cdot \frac{x - x_j}{x_i - x_j}$
    $L_n(x) = L_n(x) + y_i \cdot s[i]$
OUTPUT: $L_n(x)$
```

Polynomial interpolation in Lagrange form has some weaknesses. If a new knot is added, all the Lagrange basis polynomials have to be changed. Numerical problems can occur when the nodes are getting closer together and it can not be used for interpolation of derivatives (the solution to that is Hermite interpolation).

One might think that increasing the number of knots and, consequently, increasing the degree of interpolating polynomial, lead to a better approximation of a function f. Unfortunately there is no guarantee of this. Actually, when interpolating by using high degree polynomials, oscillation between points can occur and the error can increase. Besides that, the computational cost of constructing and evaluating high-degree polynomial approximations is not inconsiderable. A known example of oscillation is **Runge's phenomenon**. The approximation gets worse with increasing of the degree of polynomial (Figure 1). Computational complexity of polynomial interpolation with n points is $O(n^2)$. [10]

2.2 Interpolation with Splines

In order to avoid the instabilities and high computational cost of interpolation with high degree polynomial, we can combine the polynomials piecewisely and form good approximating functions called **splines**. Splines are particularly useful because they are stable and smooth. The most commonly used cubic spline also has continuous first and second derivatives. Basically, we divide the interval $[x_0, x_n]$ into subintervals and approximate the function f using low degree polynomials on each subinterval.

The simplest form of spline function is **piecewise linear interpolation**. On each interval $[x_i, x_{i+1}]$, for $i = 0, ..., n - 1$, the interpolating function is defined with the equation for linear interpolation:

$$p_i(x) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i), \quad x \in (x_i, x_{i+1}).$$

The resulting curve is not smooth and derivatives are not continuous. A simple example of piecewise linear interpolation on 6 nodes is shown on figure 19 in Appendix. [17]

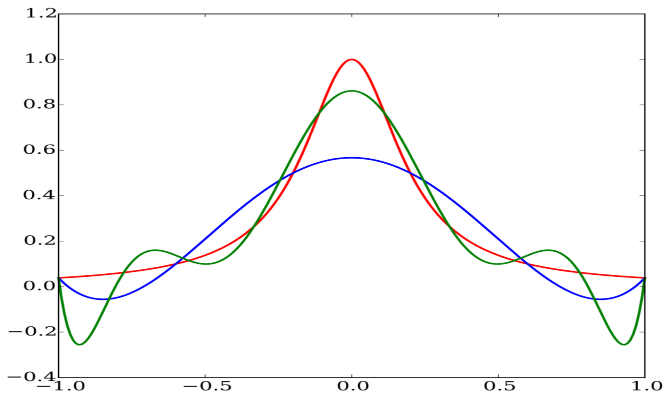

Figure 1: Runge's phenomenon: the red curve is Runge function, the blue curve is 5-th order interpolating polynomial and the green curve is 9-th order interpolating polynomial. The approximation in the middle is very good but terrible at the ends. Source: `https://en.wikipedia.org/wiki/Runge\%27s_phenomenon`.

The most popular are **cubic splines**. A cubic spline S for a function f on interval $[x_0, x_n]$ is a cubic polynomial of a form

$$S_i(x) = y_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

on each subinterval $[x_i, x_{i+1}]$ for $i = 0, ..., n-1$. It matches function values in all nodes ($S(x_i) = f(x_i)$ for $i = 0, ..., n$). It has to satisfy boundary conditions: $S''(x_0) = S''(x_n) = 0$ or $S'(x_0) = f'(x_0)$ and $S'(x_n) = f'(x_n)$. To consturuct the cubic spline interpolant we can use the following algorithm (source: `http://facstaff.cbu.edu/wschrein/media/M329%20Notes/M329L67.pdf`):
INPUT: $x_i, y_i, \quad i = 0, ..., n$

1. Set $h_i = x_{i+1} - x_i$ for all $i = 0, ..., n-1$.
2. Set $\alpha_i = \frac{3}{h_i}(y_{i+1} - y_i) - \frac{3}{h_{i-1}}(y_i - y_{i-1})$ for all $i = 0, ..., n-1$.

   Step 3, 4, 5 and part of Step 6 solve the tridiagonal linear system.
3. Set $l_0 = 1, \quad \mu_0 = 0, \quad z_0 = 0$.
4. For $i = 1, 2, ..., n-1$ set
   $l_i = 2(x_{i+1} - x_{i-1}) - h_{i-1}\mu_{i-1}$,
   $\mu_i = \frac{h_i}{l_i}$,
   $z_i = \frac{\alpha_i - h_{i-1}z_{i-1}}{l_i}$.
5. Set $l_n = 1, \quad z_n = 0, \quad c_n = 0$.
6. For $i = n-1, n-2, ..., 0$ set
   $c_i = z_i - \mu_i c_{i+1}$,
   $b_i = \frac{y_{i+1} - y_i}{h_i} - h_i \frac{c_{i+1} + 2c_i}{3}$,
   $d_i = \frac{c_{i+1} - c_i}{3h_i}$.

OUTPUT: $y_i, b_i, c_i, d_i, \quad i = 0, ..., n-1$.

The spline S as well as its first and second derivatives S' and S'' are continuous. Computational complexity of cubic spline interpolation with n points is $O(n + log(n))$. [10] A simple example of cubic spline interpolation on 6 nodes is shown on figure 20 in Appendix.

Another method of curve fitting is interpolation with B-splines (basis splines). A **B-spline curve** of degree k is defined as a linear combination of given control points $p_0, ..., p_n$ and B-spline basis functions $N_{i,k}(x)$:

$$B(x) = \sum_{i=0}^{n} N_{i,k}(x) p_i.$$

We are also given a so called knot vector in non-descending order, $\mathbf{x} = (x_0, x_1, ..., x_m)$, where $m = k + n + 1$. The 0-th basis function is defined as:

$$N_{i,0}(x) = \begin{cases} 1 & \text{if } x_i \leq x \leq x_{i+1}, \\ 0 & \text{otherwise.} \end{cases}$$

We define other basis functions $N_{i,k}(x)$ using the Cox-de Boor recursion formula:

$$N_{i,k}(x) = \frac{x - x_i}{x_{i+k} - x_i} N_{i,k-1}(x) + \frac{x_{i+k+1} - x}{x_{i+k+1} - x_{i+1}} N_{i+1,k-1}(x).$$

The control points provide the shape of the curve. [3]

To perform a B-spline interpolation or approximation we first have to select parameters $x_0, x_1, ..., x_n$ and generate the knot vector. To do that, many methods are available (e. g. the uniformly spaced method, the chord length method,...). After that we derive the B-spline curve that passes all the given data points in the given order (interpolation). An example of B-spline curve is shown on figure 21 in Appendix.

The advantage of B-splines is that they are very flexible and allow one to create a wide variety of curves. The main disadvantage is the complexity in their definition. [3]

2.3 Hermite Interpolation

Interpolation with Lagrange basis polynomials fails if we want to match the interpolating polynomial not only in function values, but also in the values of first derivative. We can use **Hermite interpolation** instead.

Let $f \in C^1[a, b]$. We are given $n + 1$ nodes $x_0, x_1, ..., x_n \in [a, b]$, their values $f(x_0), f(x_1), ..., f(x_n)$ and their values of the first derivative $f'(x_0), f'(x_1), ..., f'(x_n)$. Our task is to find an interpolating polynomial $p(x)$, such that $p(x_i) = f(x_i)$ and $p'(x_i) = f'(x_i)$ for all $i = 0, ..., n$. To satisfy the constraints, such polynomial must have degree $2n + 1$. It is called Hermite polynomial, denoted $H_{2n+1}(x)$, and has the form

$$H_{2n+1}(x) = \sum_{i=0}^{n} f(x_i) H_{i,n}(x) + \sum_{i=0}^{n} f'(x_i) \widehat{H}_{i,n}(x)$$

where

$$H_{i,n}(x) = (1 - 2(x - x_i) l'_{i,n}(x_i)) l^2_{i,n}(x) \quad \text{and} \quad \widehat{H}_{i,n}(x) = (x - x_i) l^2_{i,n}(x)$$

and $l_{i,n}$ is i-th Lagrange basis polynomial of degree n. [7]

Furthermore, if $f \in C^{2n+2}[a,b]$, we can express the remainder R_n as

$$R_n = f(x) - H_{2n+1}(x) = \frac{f^{(2n+2)}(\xi_x)}{(2n+2)!}(x-x_0)^2(x-x_1)^2 \dots (x-x_n)^2.$$

The disadvantage of representation of the Hermite polynomial in terms of Lagrange polynomials and their derivatives is the difficulty of differentiating and evaluating these polynomials. Hermite interpolation can also be used to interpolate values of higher order derivatives. [7]

3 Approximation Methods

In mathematics we often come across complicated functions that are very difficult to deal with. A possible solution to the problem is mathematical technique called approximation. We try to find a simpler function that captures some of the essential information of the original function. Even though this is only an approximation to the original problem, it does not matter as long as the approximation is sufficiently good. [17]

To measure the distance between the original function and its approximation we use a norm. A map is a norm if it satisfies the following properties: [19]

1. $f = 0$ iff $||f|| = 0$,
2. $||f+g|| \leq ||f|| + ||g||$,
3. $||\alpha f|| = |\alpha| ||f||$.

We will use an L_p-norm, defined as

$$||f(x) - g(x)||_p = \Big(\int |f(x) - g(x)|^p \omega(x) dx \Big)^{\frac{1}{p}},$$

where $\omega(x)$ is a nonnegative weight function and p usually equals 1,2 or ∞. If $m < p$, then $L_m < L_p$. The L_∞-norm is defined as supremum norm: $||f(x) - g(x)||_\infty = \sup_x |f(x) - g(x)|$.

In a discrete case values of the function are given only at finitely many points: (x_i, y_i) for $i = 0, ..., n$. The L_p-norm is defined as a sum:

$$||f(x) - g(x)||_p = \Big(\sum_{i=0}^{n} |f(x_i) - g(x_i)|^p \Big)^{\frac{1}{p}}.$$

For $p = \infty$ we define $||f(x) - g(x)||_\infty = \sup_{i \in \{0,\dots,n\}} |f(x_i) - g(x_i)|$. [19] We will try to find the best approximating function by minimizing the distance between the original function (or given points) and the approximating function according to these norms.

In addition to L_p-norms we also know matrix norms. A short explanation and important examples can be found in Appendix.

3.1 Taylor Approximation

First, let us take a look at the Taylor approximation. A Taylor polynomial is a simple approximation to a function that is based only on information about the function at a single point a. The **Taylor polynomial of a function f of degree n at point a** is unique and can be written as

$$T_n(f;a)(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2}f''(a) + ... + \frac{(x-a)^n}{n!}f^{(n)}(a).$$

By increasing the degree of a Taylor polynomial the approximation of f improves. An example of Taylor approximation is shown on figure 22 in Appendix. [4]

Since $T_n(f;a)$ is only an approximation, it is usually important to calculate and control the error (remainder) $R_n(f;a)$. Suppose that f is a function whose derivatives up to order $n+1$ exist and are continuous. Then the remainder in the Taylor approximation can be expressed in following forms:

$$R_n(f;a)(x) = f(x) - T_n(f;a)(x) = \frac{1}{n!}\int_a^x f^{(n+1)}(t)(x-t)^n dt = \frac{(x-a)^{n+1}}{(n+1)!}f^{(n+1)}(\xi_x)$$

where $\xi_x \in (a,x)$. [4]

Approximation with Taylor polynomial works well if we know many derivatives of function f at point a. A Taylor polynomial at a point $x = a$ usually provides a very good approximation near a, but as we move away from this point, the error increases. If we want a good approximation to a function f across a whole interval, we have to use information about values of the function in different points of the interval. [4]

3.2 Least Maximum Approximation

We are trying to approximate function f by a polynomial p so that L_∞ norm is minimized: $||f(x) - p(x)||_\infty = \sup_x |f(x) - p(x)|$. The best approximation in the L_∞ norm is obtained by minimizing the maximal distance. In the practice this norm is appropriate to use if the data errors are very small. Since the L_∞ norm is not strictly convex, the best approximations are not necessarily unique. [13]

The following theorem gives the solution to the least maximum (L_∞) approximation problem for polynomials:

Theorem: Let $f(x)$ be continuous on $[a,b]$. Let $p(x)$ be the polynomial of degree n that minimizes $E(f,p) = max_{a \leq x \leq b}|f(x) - p(x)|$, so that $E(f,p) = E_n$. The error is defined as $r(x) := f(x) - p(x)$. Then there are $n+2$ points $a \leq x_0 < x_1 < \cdots < x_{n+1} \leq b$ in which $r(x)$ takes the values E_n or $-E_n$ with alternating signs, $r(x_k) = -r(x_{k-1})$. [14]

For the construction of the polynomial of least maximum approximation we use Remez algorithm. It is an efficient iterative algorithm with the following procedure:

1. For a given function $f(x)$ on an interval $[a,b]$, specify the degree n of interpolating polynomial and a set $E = \{x_0, x_1, ..., x_{n+1}\}$ of $n+2$ nodes (usually Chebyshev nodes).

2. Solve the linear system of equations

$$f(x_i) - (b_0 + b_1 x_i + ... + b_n x_i^n) = (-1)^i m$$

   where $i = 0, 1, ..., n+1$ for the unknowns $b_0, b_1, ..., b_n$ and m.

3. Use the b_i as coefficients to form a polynomial p_n.

4. Find $y \in [a, b]$ such that $|p_n(y) - f(y)| = max_{x \in [a,b]} |p_n(x) - f(x)|$.

5. Find the neighbors of y in E (denoted as y_1 and y_2). Calculate $r(y) = p_n(y) - f(y)$, $r(y_1)$ and $r(y_2)$.

6. If $sign(r(y)) = sign(r(y_1))$, replace y_1 in E with y. Otherwise, replace y_2 in E with y.

7. Repeat until the set of nodes E do not change anymore. Then p_n is the polynomial of least maximum approximation. [16]

An example of least maximum approximation with Remez algorithm is shown on figure 23 in Appendix.

3.3 Least Squares Approximation

Another method of approximation is least squares approximation. We approximate the function with a polynomial p_m of degree m so that the L_2 norm is minimized. The approximating polynomial does not necessarily pass through the data points. [17]

Since the L_2 norm is strictly convex, the best approximation exists, is unique and depends continuously and smoothly on a function which is approximated. It has been shown that the L_2 norm is the most suitable choice for approximating the data if data errors are normally distributed. In this case we minimize the effect of these errors by using the L_2 norm. [13]

We are given a set of $n + 1$ points (x_i, y_i), where $i = 0, ..., n$. The basic idea of least squares approximation is minimizing S which is the sum of the squares of the differences (residuals) r_i-s between given values of the dependent variable y_i-s and the values predicted by the approximation:

$$S(\beta) = \sum_{i=0}^{n} r_i^2 = \sum_{i=0}^{n} (y_i - p_m(x_i, \beta))^2,$$

where p_m is the approximating polynomial of degree m, depending on a vector of parameters β.

Our task is to determine the optimal set of parameters $\beta = [\beta_0, ..., \beta_m]^T$ for approximating polynomial $p_m(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + ... + \beta_m x^m$. Partial derivatives of S with respect to β must be zero for a local minimum:

$$\frac{\partial S(\beta)}{\partial \beta} = 0.$$

From these first order conditions we derive the optimal parameters $\{\beta_0, ..., \beta_m\}$ and detemine the best fitting polynomial according to the least squares approximation. The equation for optimal parameters can also be written in the following form:

$$\beta = (X^T X)^{-1} X^T \mathbf{y},$$

where $\mathbf{y}$ is a vector of y_i-s and X is a $n \times (m+1)$ matrix:

$$X = \begin{bmatrix} 1 & x_0 & x_0^2 & ... & x_0^m \\ 1 & x_1 & x_1^2 & ... & x_1^m \\ \vdots & \vdots & \vdots & ... & \vdots \\ 1 & x_n & x_n^2 & ... & x_n^m \end{bmatrix}$$

In case of $m = 1$ we are talking about **linear least squares approximation**. [6]

Another way of constructing the least squares approximating polynomial p_m of degree m is using the Gram matrix. Let $\{1, x, ..., x^m\}$ be the basis of the polynomial vector space. The approximating polynomial has the form $p_m(x) = \sum_{i=0}^{m} \beta_i x^i = \beta_0 + \beta_1 x + \beta_2 x^2 + ... + \beta_m x^m$. We define the scalar product $\langle g, h \rangle := \sum_{j=0}^{n} g(x_j)h(x_j)$. If p_m is the least squares approximating polynomial the following equalities must hold:

$$\langle f - p_m, x^i \rangle = \langle f - \sum_{k=0}^{m} \beta_k x^k, x^i \rangle = 0 \quad \text{for all} \quad i = 0, ..., m.$$

These conditions can be represented as linear system of equations

$$G\boldsymbol{\beta} = \mathbf{d}$$

where $\boldsymbol{\beta} = (\beta_i)_{i=0}^{m}$ is the vector of unknown coefficients, $\mathbf{d} = (\langle f, x^i \rangle)_{i=0}^{m}$ and G is the Gram matrix

$$G = \begin{bmatrix} \langle 1, 1 \rangle & \langle x, 1 \rangle & ... & \langle x^m, 1 \rangle \\ \langle 1, x \rangle & \langle x, x \rangle & ... & \langle x^m, x \rangle \\ \vdots & \vdots & ... & \vdots \\ \langle 1, x^m \rangle & \langle x, x^m \rangle & ... & \langle x^m, x^m \rangle \end{bmatrix}$$

Increasing of the degree of approximating polynomial gives better approximation, but there is a limit. When the degree is too high we again face the unstability of the method and the oscillation problem. Anexample of least squares approximation with a 3rd degree polynomial is shown on figure 24 in Appendix. [6]

Using the least square approximation is particularly appropriate when we know that there may be significant errors in the data. Forcing a curve exactly through the data when it is known that the data is inexact clearly does not make much sense. Another advantage is the possibility of keeping the degree of the polynomial low even when the number of data points is high, so the problem of oscillation does not arise. [17]

4 Interpolation and Approximation in Practice

4.1 Applications of the Methods

For choosing the most suitable approximation or interpolation method we should understand the underlying problem well. Linear interpolation is often used to fill the gaps in a table. For example, if we are given a table with the population of some country in 1970, 1980, 1990 and 2000, and want to estimate the population in 1994, linear interpolation is an easy way to do this. Interpolation by linear function is also appropriate for example for describing the relation between the temperature of the liquid and the length of a mercury column on non-digital thermometer. [17]

The next example, we want to know how the volume of gas depends on pressure. We are given some tabulated values and want to determine volumes of the gas at non-tabulated pressure values and estimate the underlying function. In this case, using a polynomial or cubic spline interpolation is more appropriate. [17]

In statistics, interpolation can be used for determining probability density functions from histograms. Numerical interpolation is also used for numerical differentiation and integration: instead of the original function f we differentiate or integrate its interpolating polynomial. Consequently, it is also useful for numerical solving of differential equations. Hermite interpolation is often used in physics. [17]

An important example of least square approximation is linear regression.

4.2 Adaptive Algorithms

An adaptive algorithm is an algorithm that changes its behavior while it is run (e.g. adjusts parameters). Besides the initially defined criterion it also uses run-time acquired available information to provide a better result. [20] New adaptive methods and algorithms are being developed in the field of interpolation and numerical approximation in order to provide more accurate interpolating and approximating functions.

Tao and Watson introduced an adaptive variable-knots algorithm for interpolation with splines. The number of knots and their locations are determined by the algorithm. This selection of knots reduces the error of the solution. [15]

Xiao et al. presented an adaptive interpolation algorithm for image resizing that is based on the Newton polynomial. It improves the limitation of the original algorithm. The adaptive algoritm also has lower complexity and better efficiency in comparison with the traditional one. [18]

Aràndiga et al. developed an adaptive interpolatory procedure based on non-linear techniques for enlarging images. [1]

5 Simulation 1: Interpolation and Approximation of a Function

5.1 Data Description and Simulation Settings

For my first simulation I decided to interpolate and approximate a known underlying function with the methods described above. The underlying function is $f(x) = sin(2x)$. In the first case I chose seven (almost) equidistand nodes on the interval $[-5, 5]$: $nodes1 = [-5, -3, -1, 0.1, 1, 3, 5]$. In the second case I chose 11 equidistant nodes on the same interval: $nodes2 = [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]$. In the third case I chose 11 random nodes with `runif` function on the same interval: $nodes3 = [-3.3433215, -2.9540927, -2.4985372, -1.5352981, -1.4065052,$ $-1.3515494, -1.1475473, -0.6984916, -0.6513513, 1.8622479, 4.0785090]$. In all cases, the y-values of the knots are the corresponding function $sin(2x)$ values of the nodes.

I expect the best fit in the case with 11 equidistant nodes, although the problem of oscillation may occur. All computations and simulations are done using R. For Lagrange interpolation I used the inbuilt function `poly.calc`, for the piecewise linear interpolation `approxfun` and for the cubic spline interpolation `splinefun`.

5.2 Results of Empirical Analysis

Figures 2, 3 and 4 show the approximating polynomials of an underlying function $f(x) = sin(2x)$ for Lagrange interpolation, piecewise linear interpolation, cubic spline interpolation and least squares approximation with polynomials of degree 5. I also derived the Taylor approximating polynomial of degree 5 around point $a = (0, 0)$, but it is not plotted on the figures. The equations of polynomials are shown in Table 1.

Method	Number of nodes	Equation of polynomial
Lagr. int.	7	$L_6(x) = 0.09177904 + 1.081224x - 0.1056479x^2 - 0.1771067x^3 + +0.01427674x^4 + 0.005180224x^5 - 0.0004079069x^6$
	11	$L_{10}(x) = 1.82212x - 1.066242x^3 + 0.1621942x^5 - -0.008936576x^7 + 0.0001612486x^9$
	11 random	$L_{10}(x) = -7.591637e - 06 + 2.000019x + 0.001494562x^2 - 1.338371x^3 + +0.002125043x^4 + 0.2753525x^5 - 0.01047957x^6 - 0.02328872x^7 + +0.001875507x^8 + 0.0007099364x^9 - 8.577362e - 05x^{10}$
Least sq. appr. (degree 5)	7	$p_5(x) = 0.03738 + 1.08487x - 0.00655x^2 - 0.17767x^3 + +2e - 04x^4 + 0.0052x^5$
	11	$p_5(x) = -0.33282x + 0.06335x^3 - 0.00214x^5$
	11 random	$p_5(x) = -0.01807 + 1.08105x - 0.02497x^2 - -0.26256x^3 - 0.00944x^4 + 0.01201x^5$
Taylor poly.	all	$T_5(f, 0)(x) = 2x - 1.33333x^3 + 0.26666x^5$

Table 1: Equations of interpolating and approximating polynomials of a function $sin(2x)$.

On the figures 2 and 3 we can see that polynomials fit better when we used 11 nodes instead of 7. But we have to be careful with adding nodes for Lagrange interpolation. On 11 equidistant nodes we can already notice significant oscillation on both ends of the interval. The random nodes on figure 4 are very concentrated on the left side of the interval. Therefore the fit on that side of the iterval is very good with all methods. On the right side of the intervl we can notice that the fit gets worse. Another important observation is that the derived polynomials in all cases oscillate wildly outside the interval $[-5, 5]$. Because of that these methods are only appropriate for interpolation. For extrapolation different methods should be used.

Figures 5, 6 and 7 show the absolute values of error functions. The error function $r(x)$ is defined as a difference between the underlying function f and the interpolating or approximating polynomial p: $r(x) := f(x) - p(x)$. Again,

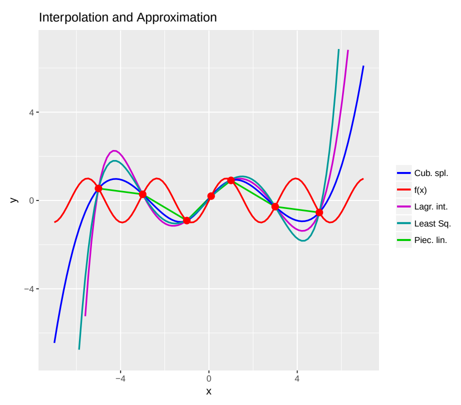

Figure 2: Interpolation and approximation of $sin(2x)$ on 7 nodes

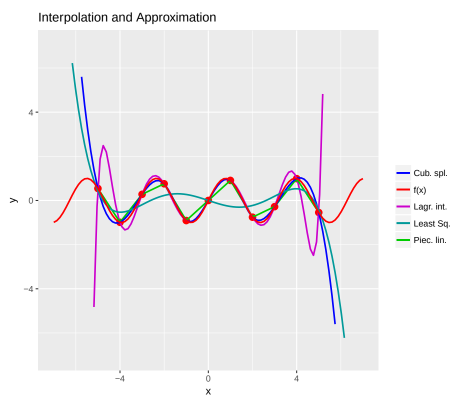

Figure 3: Interpolation and approximation of $sin(2x)$ on 11 nodes

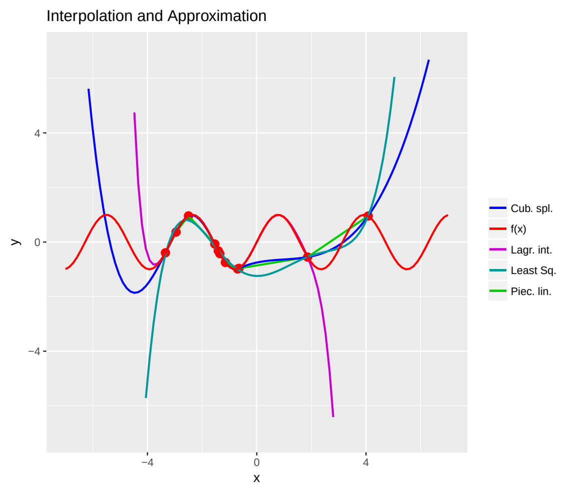

Figure 4: Interpolation and approximation of $sin(2x)$ on 11 random nodes

we can see that errors of interpolating polynomials are closest to zero in the case of 11 equidistant nodes. Surprisingly, in the case of 7 nodes the errors on some parts are the lowest when fitting the piecewisely linear function. The best fit on 11 nodes is the cubic spline interpolation. We can also notice a big increase of the error function (oscillation) of Lagrange interpolation at the ends of the interval as a consequence adding more knots. The error of least square approximation on 11 nodes is quite high compared to the interpolating methods. In the case of 11 random nodes the error of Lagrange interpolation is very small on the left part, but increases on the right part because there are very few knots. If we want a good approximation along the whole interval the equidistant nodes provide better result.

Figures 8 to 11 compare the absolute values of error functions of the same method for different numbers of nodes separately. For all interpolation methods the green line (11 equidistant nodes) is below the other two when considering the whole interval. The increasing the number of nodes improved the result.

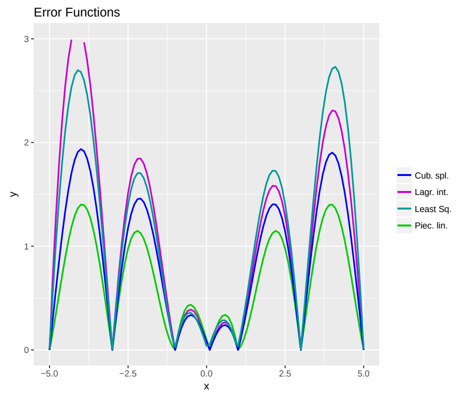

Figure 5: Errors of interpolation of $sin(2x)$ on 7 nodes

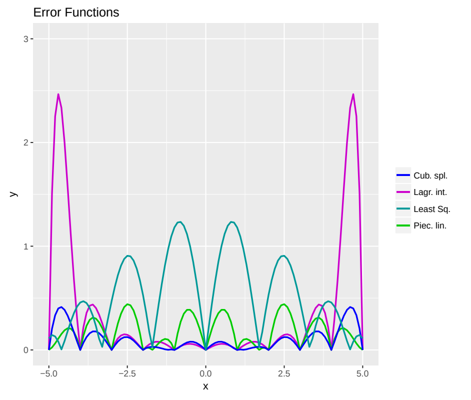

Figure 6: Errors of interpolation of $sin(2x)$ on 11 nodes

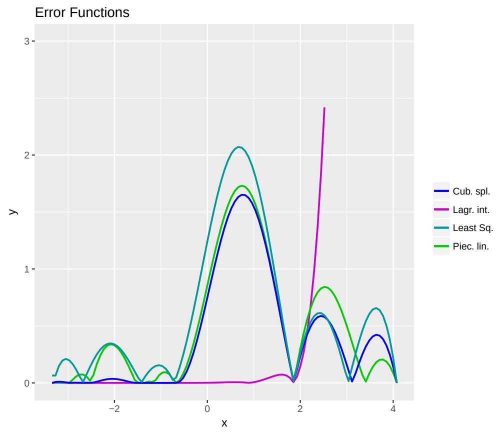

Figure 7: Errors of interpolation of $sin(2x)$ on 11 random nodes

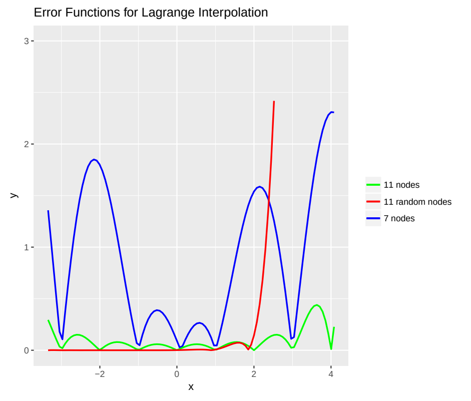

Figure 8: Errors for Lagrange interpolation

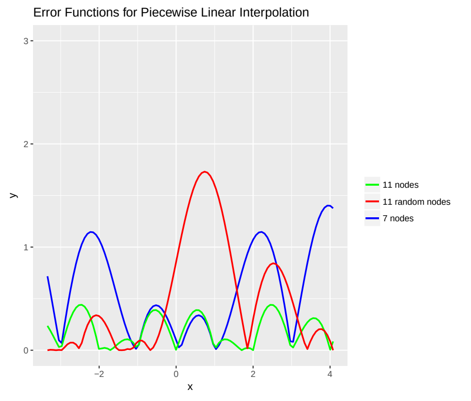

Figure 9: Errors for piecewise linear interpolation

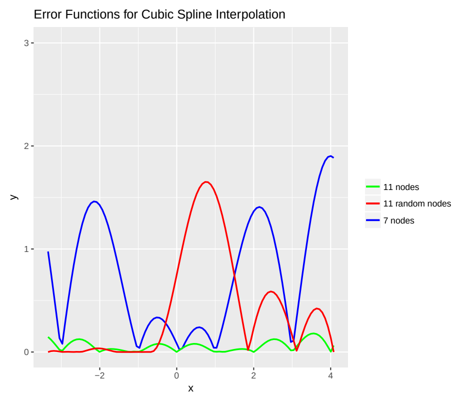

Figure 10: Errors for cubic spline interpolation

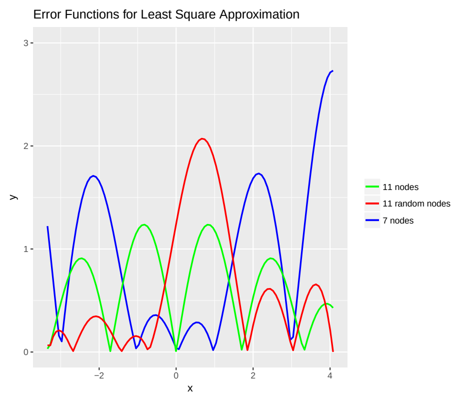

Figure 11: Errors for least square approximation

6 Simulation 2: I-Curve

6.1 Data Description and Simulation Settings

For my second simulation I decided to work on the example described in introduction and derive the I-curve. I used data about daily treasury yield curve rates from U.S. Department of the Treasury. It is available at `https://www.treasury.gov/resource-center/data-chart-center/interest-rates/Pages/TextView.aspx?data=yield`. For maturities of 1, 3, 6 months, 1, 2, 3, 5, 7, 10, 20 and 30 years the table provides annualized yield rates: no matter what maturity we are talking about, that is the rate that on average we would get for a year. The rates are obtained at 3:30 PM each trading day. I used the data from 3rd July 2018. For deriving the interpolated yield curve, the yields of so called on-the-run securities should be used. These are most recently issued Treasury securities. [21], [23]

Given the yield rates of Treasury securities with different maturities I derived the interpolated yield curves with different methods. Maturities are used as nodes and corresponding yield rates as corresponding tabulated function values.

Since the underlying function of the data is not known we can not calculate the errors as in Simulation 1. U.S. Department of the Treasury uses a cubic spline model to estimate the yield curve so I used the cubic spline function as an underlying function for calculating the errors. [21]

I applied the interpolation methods because we know the fitting polynomial has to pass through the given knots in this case. I also derived the least square approximation polynomials of degrees 1, 2 and 3 but using the interpolation polynimials makes more sense. Because of their little importance in the case of I-curve I did not analyse the errors of least square approximation polynomials. All computations and simulations are done using R.

6.2 Results of Empirical Analysis

Figures 12 to 14 separately show the I-Curves (polynomials) that interpolate the data about yields and maturities. Cubic spline interpolation shown on figure 12 is officially used for by U.S. Department of the Treasury for calculating the yield spreads. We can see the piecewise linear inteprolation does not differ a lot from the cubic spline. It is a simpler method that also provides a useful result. The Lagrange interpolation on the other hand does not work well in this case. The interpolation was performed on 11 nodes and the interpolating polynomial is of 10th degree. Its equation is: $L_{10}(x) = 1.896674 + 0.03062676x + 1.753903x^2 - 2.570926x^3 + 1.746458x^4 - 0.6488961x^5 + 0.1378255x^6 - 0.01674792x^7 + 0.001119937x^8 - 3.723101e - 05x^9 + 4.712203e - 07x^10$. Huge oscillation problem appears on the very right side of the interval. Besides the high degree, the reason for oscillation could be the same as in the case of random nodes. On the left side of the interval the nodes are close together while on the right side the gaps are getting bigger (10 years). Figure16 shows the absolute values of errors compared to the cubic spline function. The graphs once again shows very big error in of Lagrange interpolation.

Figure 15 shows linear, quadratic and cubic polynomials that minimize the L_2 norm of the data.

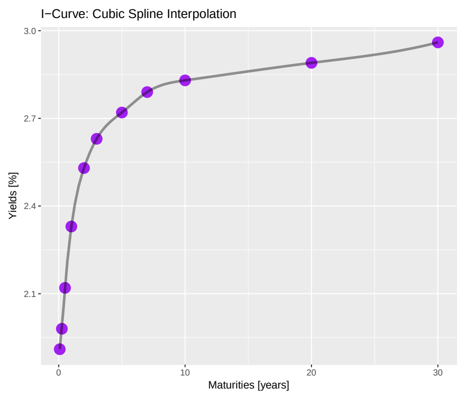

Figure 12: I-Curve: Cubic spline interpolation

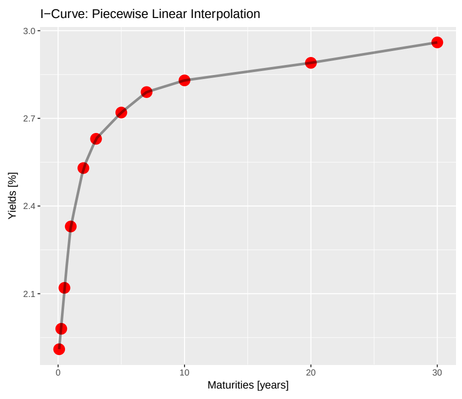

Figure 13: I-Curve: Piecewise linear interpolation

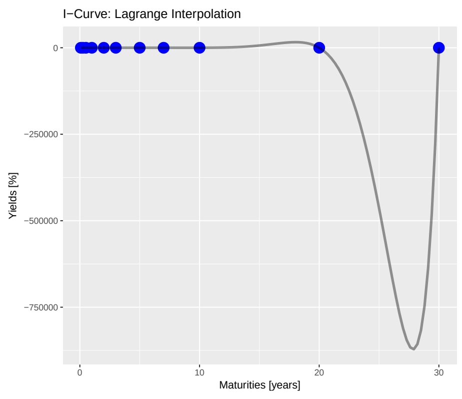

Figure 14: I-Curve: Lagrange interpolation

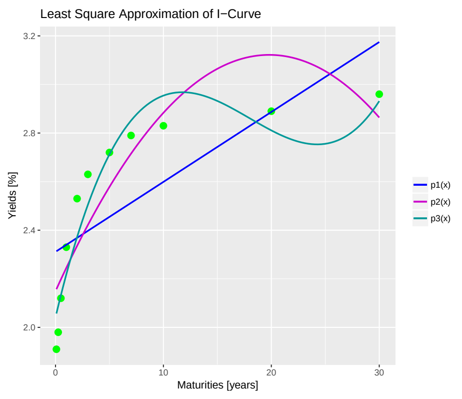

Figure 15: I-Curve: Least square approximation

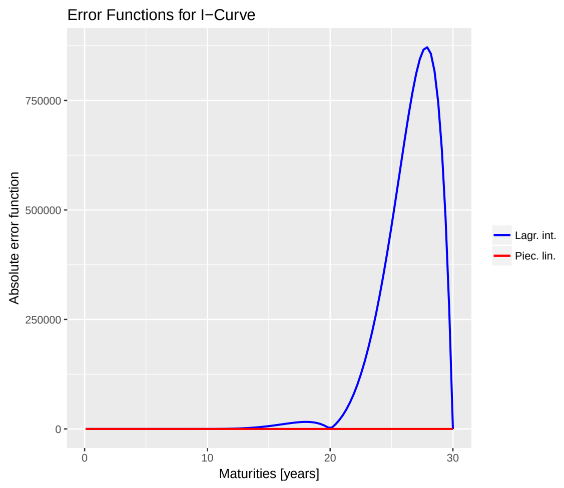

Figure 16: Errors for I-Curve interpolation

7 Conclusion

For my first simulation I used interpolation and approximation methods for the underlying function $sin(2x)$. I distinguished three cases: seven equidistand nodes, 11 equidistant nodes and 11 random nodes on the same interval. The results show that polynomials fit better when we used 11 equidistant nodes instead of 7, but Lagrange interpolation oscillates on both ends of the interval on 11 nodes. In the case of random nodes the fit is very good where the nodes are close together. Where the gaps increase the fit gets worse. All interpolating polynomials oscillate wildly outside the interpolating interval. The best fit on 11 nodes is the cubic spline interpolation. The error of least square approximation on 11 nodes is quite high compared to the interpolating methods.

In the second part I was dealing with the problem about U.S. Treasury securities. These are debt obligations of the U.S. government sold with eleven different maturities between 1 month an 30 years. The yield rates are known for all of these maturities, but not for the maturities inbetween. A possible way to find approximate values of unknown yields is using numerical interpolation. For the simulation I used data about daily treasury yield curve rates from U.S. Department of the Treasury and derived the interpolated yield curves with different methods. U.S. Department of the Treasury uses a cubic spline model to estimate the yield curve so I used the cubic spline function as an underlying function for calculating the errors. The results show that the piecewise linear inteprolation does not differ a lot from the cubic spline. The Lagrange interpolation on the other hand does not work well in this case. Huge oscillation problem appears because of the high degree of interpolating polynomial and because the nodes are very concentrated on one side of the interval.

References

[1] Aràndiga, F. et al. *Fast communication: Adaptive interpolation of images.* Signal Processing 83, Issue 2, (2003) 459 – 464.

[2] Chan, A. *Nonlinear interpolation with Excel to construct U.S. Treasury Bond Yield Curve.* CompAct Electronic Newsletter, Issue 38, Januray 2011.

[3] Finn, D. L. *Course Notes: B-Splines and NURBS.* Available on `https://www.rose-hulman.edu/~finn/CCLI/Notes/day24.pdf`

[4] Globevnik, J., Brojan, M. *Lecture Notes: Analiza 1.* Available on `https://www.fmf.uni-lj.si/~globevnik/skripta.pdf`

[5] Hagan, P. S., West, G. *Methods for Constructing a Yield Curve.* Wilmott magazine, January 2008, 70 – 81.

[6] Krajnc, M. *Lecture notes: Numericna aproksimacija in interpolacija (2013).* Available on `https://ucilnica.fmf.uni-lj.si/pluginfile.php/7435/mod_resource/content/8/NumericnaAnaliza.pdf`

[7] Lambers, J. *Lecture Notes: Hermite Interpolation.* Available on `http://www.math.usm.edu/lambers/mat772/fall10/lecture6.pdf`

[8] Lambers, J. *Lecture Notes: Lagrange Interpolation.* Available on `http://www.math.usm.edu/lambers/mat772/fall10/lecture5.pdf`

[9] Lange, K. *Numerical Analysis for Statisticians.* New York: Springer Verlag (1999).

[10] Liao, S. *Interpolation and American options pricing.* Available on `https://www.csie.ntu.edu.tw/~lyuu/theses/thesis_r91723055.pdf`

[11] Liu, Z., Vandenberghe, L. *Interior-point method for nuclear norm approximation with application to system identification.* SIAM J. Matrix Anal. & Appl., (2010), 31(3), 1235 –1256.

[12] Lyche, T. *Lecture Notes: Matrix Norms.* Available on `https://www.uio.no/studier/emner/matnat/ifi/INF-MAT4350/h08/undervisningsmateriale/chap13slides.pdf`

[13] Marošević, T. *A choice of norm in discrete approximation.* Mathematical Communications 1 (2), (1996), 147–152.

[14] Monahan, J.F. *Numerical Methods of Statistics.* Cambridge: Cambridge University Press (2001).

[15] Tao, T. M., Watson, A. T. *An Adaptive Algorithm for Fitting with Splines.* AIChE Journal, October 1988, Vol. 34, No. 10, 1722 – 1725.

[16] Tasissa, A. *Function Approximation and the Remez Algorithm.* Available on `http://homepages.rpi.edu/~tasisa/remez.pdf`

[17] Woodford, C., Phillips, C. *Numerical Methods with Worked Examples.* London: Chapman & Hall (1997).

[18] Xiao, J. et al. *Adaptive Interpolation Algorithm for Real-time Image Resizing.* International Conference on Innovative Computing, Information and Control, October 2006, 221 – 224.

[19] Zitkovic, G. *Lecture Notes: Lebesgue Spaces and Inequalities.* Available on `https://www.ma.utexas.edu/users/gordanz/notes/lp_inequalities.pdf`

[20] *Adaptive Algorithm*, [retrieved 3. 7. 2018], available on `https://en.wikipedia.org/wiki/Adaptive_algorithm`.

[21] *Daily Treasury Yield Curve Rates*, [retrieved 4. 7. 2018], available on `https://www.treasury.gov/resource-center/data-chart-center/interest-rates/Pages/TextView.aspx?data=yield`.

[22] *Interpolated Yield Curve - I Curve*, [retrieved 4. 6. 2018], available on `https://www.investopedia.com/terms/i/interpolated_yield_curve.asp`.

[23] *On-The-Run Treasuries*, [retrieved 4. 6. 2018], available on `https://www.investopedia.com/terms/o/on-the-runtreasuries.asp`.

[24] *Introduction to Treasury Securities*, [retrieved 4. 6. 2018], available on `https://www.investopedia.com/articles/investing/073113/introduction-treasury-securities.asp`.

[25] *Valuation I: Interpolation*, [retrieved 4. 6. 2018], available on `https://www.financialencyclopedia.net/valuation/i/interpolation.html`.

Appendix

Figures

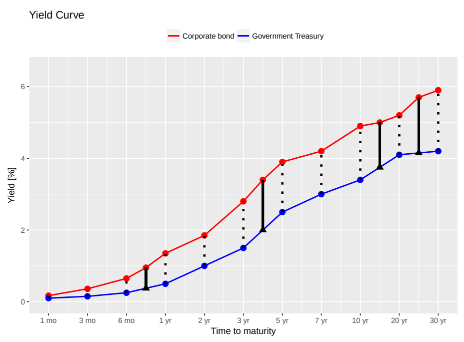

Figure 17: Interpolated yield curves and yield spreads

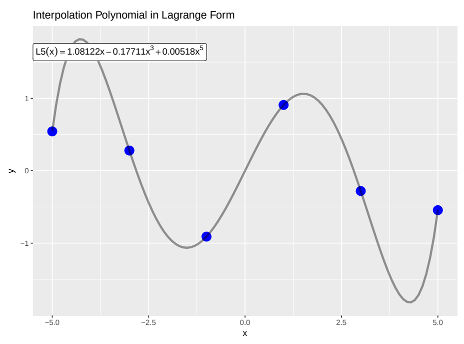

Figure 18: Lagrange interpolation on 6 nodes

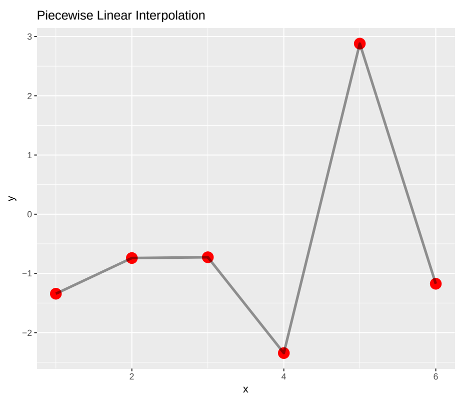

Figure 19: Pieceswise linear interpolation on 6 nodes

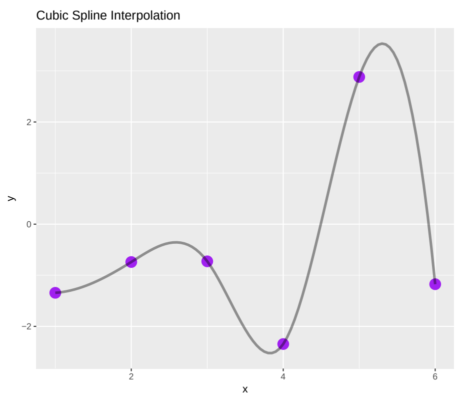

Figure 20: Cubic spline interpolation on 6 nodes

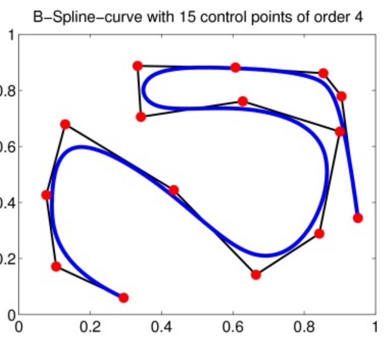

Figure 21: Example of B-spline interpolation. Source: `http://m2matlabdb.ma.tum.de/download.jsp?MC_ID=7&SC_ID=7&MP_ID=485`.

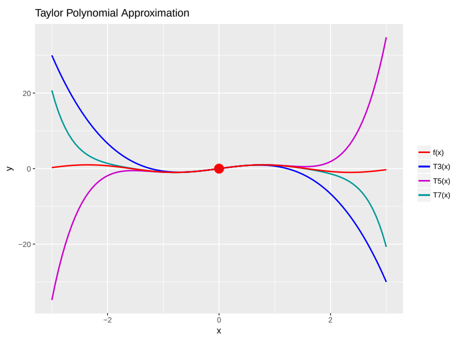

Figure 22: Different degree Taylor polynomials for $sinx$ around point $a = 0$

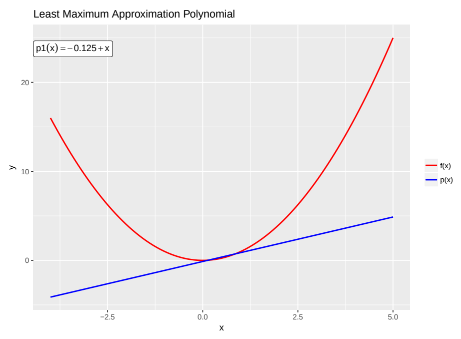

Figure 23: Least maximum approximation of $f(x) = x^2$ with first degree polynomial

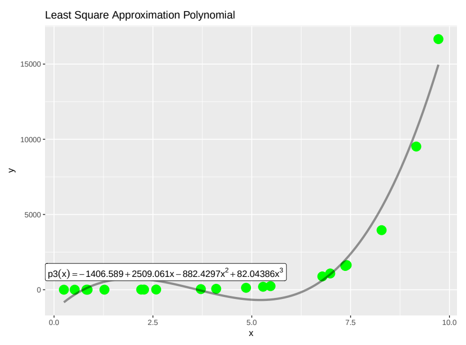

Figure 24: Least square approximation with 3rd degree polynomial

Matrix norms

Besides L_p-norms we also know matrix norms. They capture the size (strength) of a matrix in some sense. A matrix norm has to satisfy the following properties:

1. $||\alpha A|| = |\alpha|||A||$
2. $||A + B|| \leq ||A|| + ||B||$
3. $||A|| \geq 0$
4. $||A|| = 0$ iff $A = 0_{m,n}$

The three most important norms that satisfy all the properties of a matrix norm are Forbenius, operator and nuclear norm. Let A be a $m \times n$ matrix of rank r. Then it has a singular value decomposition $A = UDV^T$, where U is $m \times r$ matrix, D is an $r \times r$ diagonal matrix and V is an $n \times r$ matrix. With d we denote the vector of (always positive) singular values, that is the vector of diagonal elements of D.

Forbenius norm of a matrix A is defined as $||A||_F = \sqrt{Tr(A^TA)} = \sqrt{Tr(D^2)} = \sqrt{\sum_{i=i}^{r} d_i^2}$. Tr is the trace (sum of diagonal elements) of a matrix. We can think of it as the L_2-norm of a vector of singular values $||d||_2$.

Operator (spectral) norm of a matrix A is defined as the largest singular value of $||A||$. We can think of it as the L_∞-norm of a vector of singular values $||d||_\infty$. It is useful in PCA method.

Nuclear (trace) norm of a matrix A is defined as the sum of all singular values of A: $||A||_* = \sum_{i=1}^{r} d_i$. We can think of it as the L_1-norm of a vector of singular values $||d||_1$. [12]

The nuclear norm is especially interesting because of its applications. It is often used in convex heuristics for rank minimization problems in control, signal processing, and statistics. [11]

The nuclear norm approximation problem is

$$min||A(x) - B||_*,$$

where B is a given $p \times q$ matrix and $A(x) = x_1A_1 + x_2A_2 + ... + x_nA_n$ is a linear mapping from $\mathbb{R}^n$ to $\mathbb{R}^{p\times q}$. Nuclear norm can be numerically approximated by so called semidefinite programming (SDP). [11]