interpolation16.tex

1 Interpolation and approximation

General question: *How to approximate a given function with a class of simpler functions.*

Assume we are given the following table of values:

x_0	x_1	x_2	$\ldots$	x_n
y_0	y_1	y_2	$\ldots$	y_n

Table 1:

The table may consist of some measured data or values y_k of a given function at the given x_k-points

If we know or if there's reason to believe, that the y-values represent values of a smooth [1] function, it may be reasonable to look for a function in a given class of functions that passes through all the data points. This is called *interpolation.*

In case the measurements are inaccurate or there are other reasons, like lots of data, it is often more reasonable to look for the trend of the data instead, and thus give up the requirement of the model function to exactly pass through the data points. Instead of interpolation we then usually look for *a least squares approximation.*

Typically the class of simpler functions is taken to be polynomials but other "basis functions" are also possible. For instance periodic data is better approximated by trigonometric polynomials.

In addition to linear approximations (problem parameters appear linearly), there are natural non-linear models, whose solution requires non-linear optimization techniques.

[1] By *smooth* we mean a function that is at least continuous and has sufficiently many derivatives for the application

1.1 Polynomial interpolation

1.2 Review to polynomial basics

Start by a well-known "high-school theorem":

Theorem 1 *If the polynomial* $p(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$ *has a zero* x_0*, then* $p(x)$ *is divisible by* $(x - x_0)$*:lla.*

Proof:
Now,

$$p(x) = p(x) - p(x_0) = a_1(x - x_0) + a_2(x^2 - x_0^2) + \ldots + a_n(x^n - x_0^n).$$

Each term $(x^k - x_0^k)$ has $(x - x_0)$ as a factor because of the formula:

$$x^k - x_0^k = (x - x_0)(x^{k-1} + x^{k-2}x_0 + \ldots + xx_0^{k-2} + x_0^{k-1}).$$

From this we get very simply a "uniqueness theorem".

Corollary 2 *If two polynomials of degree at most* n *agree at* $n + 1$ *different points, they are identical.*

Proof: Let p and q be polynomials of degree at most n, which agree at the distinct points $x_0, \ldots, x_n$. Then their difference $r(x) = p(x) - q(x)$ is a polynomial of degree $d \leq n$.

Now, r has $n + 1$ and thus certainly $d + 1$ different zeros $x_0, \ldots, x_d$.

Applying d times Theorem 1 gives us the representation

$$r(x) = c(x - x_0).\ldots.(x - x_{d-1}),$$

with some constant c .

As $r(x_d) = 0$ and all factors $x_d - x_i, i = 0 \ldots d - 1$ are different from zero, c must be zero. Thus $r(x) = p(x) - q(x) \equiv 0$.

Note 3 *Let's emphasize that all what is needed for this conclusion is the above elemantary reasoning. So* ***no dependence on*** *the much deeper* ***Fundamental theorem of algebra****.*

1.3 Polynomial interpolation

Interpolation task:

Given $n+1$ x- and y-values (Table 1), find a polynomial p of degree at most n, that satisfies

$$p(x_k) = y_k, k = 0, \ldots, n.$$

Solution by linear system of equations

Let $p(x) = a_0 + a_1 x + \ldots + a_n x^n$.

There are $n+1$ unknown coefficients $a_0, a_1, \ldots a_n$, and the known data points give us $n+1$ equations $p(x_k) = y_k, k = 0, \ldots n$. So it's reasonable to hope for a (unique) solution.

Let's start with an example:

Example 1 *Let* $x_0 = -2, x_1 = -1, x_2 = 3$ *and* $y_0 = 1, y_1 = -2, y_2 = 5$. *We are looking for a* 2^{nd} *degree polynomial*

$p(x) = a_0 + a_1 x + a_2 x^2$, *satisfying* $p(x_k) = y_k, k = 0, 1, 2$.

Thus we get the system of equations

$$\begin{cases} a_0 + a_1(-2) + a_2(-2)^2 = 1 \\ a_0 + a_1(-1) + a_2(-1)^2 = -2 \\ a_0 + a_1 3 + a_2 3^2 = 5 \end{cases}$$

for solving the coefficients a_0, a_1, a_2.

$$A = \begin{bmatrix} 1 & -2 & 4 \\ 1 & -1 & 1 \\ 1 & 3 & 9 \end{bmatrix} \text{ ja } b = \begin{bmatrix} 1 \\ -2 \\ 5 \end{bmatrix}$$

Note the structure of the matrix A: First column: `ones`, second column: xdata, third column: xdata^2.

In this simple case we could build A just row-wise and proceed as follows:

```
>> A=[1 -2 4;1 -1 1;1 3 9]

A =
     1    -2     4
     1    -1     1
     1     3     9

>> b=[1;-2;5]

b =
     1
    -2
     5

>> a=A\b

a =
   -3.1000
   -0.1500
    0.9500
```

A generalizable way of forming the matrix `A` would be:

```
>> xdata=[-2;-1;3];
>> A=[ones(3,1),xdata,xdata.^2]
```

This kind of matrix has the name *Vandermonde* matrix.

Note 4 MATLAB *has the command* vander. *Try* `vander(xdata)` *and* `fliplr(vander(xdata))` *. See also* `help vander`*,* `help polyval` *,*
`http://se.mathworks.com/help/matlab/ref/polyval.html`

Plot the xy-data

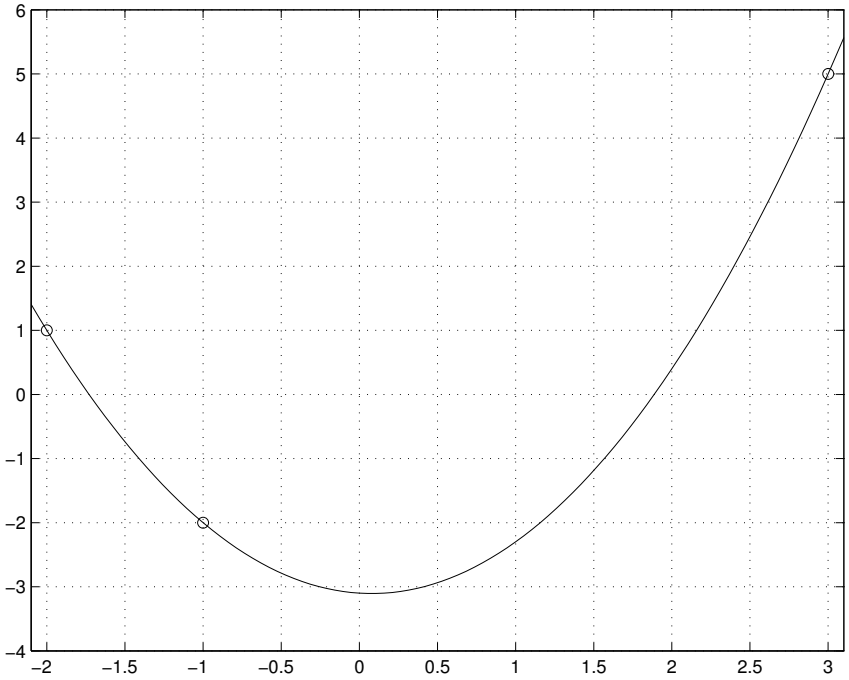
```
>> xdata=[-2 -1 3];
>> ydata=[1 -2 5];
>> plot(xdata,ydata,'o')
>> hold on
```

Add interpolation polynomial to the picture

```
>> x=linspace(-2.1,3.1,100); % 100 evalution points.
>> p=a(1)+a(2)*x+a(3)*x.^2;  % Note .^
>> plot(x,p)
>> xlim([-2.1 3.1])          % Adjust x-view.
>> grid on
```

Note 5 MATLAB *has the command* ***polyval*** *for evaluating polynomials. It is highly recommended in general. Its form is* `polyval(coeff,x)` ***but*** *the coefficient vector is given in the "****most significant first****" order. Thus in this case one would do*
`p=polyval(flipud(a),x);`

(Note here that a is a column vector, thus *flipud* instead of *fliplr*.)



The obvious first check is to see if the polynomial passes through all data points, like in this figure.

In the general case we are looking for a polynomial

$$p(x) = a_0 + a_1 x + \ldots + a_n x^n$$

and we are led to the system of equations with matrix A and rhs y

$$A = \begin{bmatrix} 1 & x_0 & \ldots & x_0^n \\ 1 & x_1 & \ldots & x_1^n \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & \ldots & x_n^n \end{bmatrix}, y = \begin{bmatrix} y_0 \\ \vdots \\ y_n \end{bmatrix}.$$

There is a unique solution provided $\det(A) \neq 0$.

As noted earlier, A is a *Vandermonde matrix*. It is possible to derive a nice formula for the determinant as a product of differences of the x_k-points, thus showing that it's different from zero as long as the x_k-points are distinct.

However, we don't need to do (or believe in) this exercise, as the existence will be proved by a clever direct construction (invented by *Lagrange*) and uniqueness was already shown in Corollary 2.

Note 6 *A Vandermonden matriisi (with distinct $x_0, \ldots, x_n$) is non-singular, but it gets more and more* ***ill-conditioned*** [2] *as n increases.*

1.4 Lagrangen interpolation method

The interpolation task has two beautiful constructive solutions. One is due to *Lagrange* and the other, guess who, *Newton*.

Let's proceed with the former.

Thus we are given distinct points $x_0, \ldots, x_n$ and corresponding arbitrary (not necessarily distinct) $y-$points like in Table 1.

Our task is to construct a polynomial p of degree at most n that "interpolates the given data", ie. satisfies: $p(x_k) = y_k, k = 0 \ldots n$.

[2] Briefly: small errors in data cause large errors in results

Lagrange's idea is a kind of orthogonal representation of the polynomial in terms of simpler polynomials in the form

$$p(x) = y_0 L_0(x) + y_1 L_1(x) + \ldots + y_n L_n(x).$$

Let's see if we can choose the polynomials L_j of degree n so that they only depend on the xdata and satisfy the "orthogonality type" relations:

$$L_k(x_j) = \delta_{kj} = \begin{cases} 1, & \text{if } k = j \\ 0, & \text{if } k \neq j \end{cases}$$

If such poynomials can be found, our problem is immedisately solved, because

$$p(x_k) = \underbrace{y_0 L_0(x_k)}_{=0} + \ldots + \underbrace{y_k L_k(x_k)}_{=y_k} + \underbrace{y_{k+1} L_{k+1}(x_k)}_{=0} + \ldots + \underbrace{y_n L_n(x_k)}_{=0} = y_k$$

How to find the "basispolynomials" L_k

To avoid notational complications, let's look at L_0. It must have zeros $x_1, x_2, \ldots, x_n$. Such a polynomial can be written as

$$L_0(x) = c_0(x - x_1)(x - x_2)\ldots(x - x_n)$$

with some constant c_0, which is determined by the condition $L_0(x_0) = 1$. Thus

$$c_0 = \frac{1}{(x_0 - x_1)(x_0 - x_2)\ldots(x_0 - x_n)}.$$

Hence,

$$L_0(x) = \frac{(x - x_1)(x - x_2)\ldots(x - x_n)}{(x_0 - x_1)(x_0 - x_2)\ldots(x_0 - x_n)}$$

Similarly for any L_k, so that the formula for L_k can be written as.

$$L_k(x) = \prod_{j=0, j\neq k}^{n} \frac{x - x_j}{x_k - x_j}.$$

In words:
Numerator: Product of terms $(x - x_j), j \neq k$
Denominator: Same product evaluated at $x = x_k$ (the term $(x_k - x_k)$ is "luckily" absent in the denominator.)

This formula can be directly turned into a Matlab function:

```
function  y  = Lag(k,xdata,x)
% Lagrange basis function for xdata evaluated at vector x.
% k refers to k^th data point.

n=length(xdata);
y=ones(size(x));
for j=[1:k-1 k+1:n]
    y=y.*((x-xdata(j))./(xdata(k)-xdata(j)));
end
```

There's a neat little trick here if you think of the for-loop when k=1. What is `1:0` ? Let's see:

```
>> 1:0
ans =
   Empty matrix: 1-by-0
```

Thus the loop indices are just right for $k = 1$ as well.

The first test would be:

```
M=zeros(N,N);
for k=1:N
    M(k,:)=Lag(k,xdata,xdata);
end
```

Resulting as expected:

```
M
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
```

Some graphic illustartions could be done along these lines:

```
xdata=[0 2 5 8 7]
N=length(xdata);
x=linspace(-.5 ,8.5); % Evaluation points
plot(x,Lag(2,xdata,x),xdata,Lag(2,xdata,xdata),'x')
grid on
```

VIIMEISTELE TEKSTIA !!!

Problem 1 *Write a function* ***polinterp****:*

```
 y=polinterp(xdata,ydata,x)
 %
```

Use the function Lag.

```
%%% Lagscript: Demonstrate Lagrange polynomials
%
%% Function Lag:
type Lag
xdata=[0 2 5 8 7]
%%
M=zeros(N,N);
for k=1:N
    M(k,:)=Lag(k,xdata,xdata);
end
M
%%
close all
N=length(xdata);
x=linspace(-.5 ,8.5); % Evaluation points
%subplot(2,1,1)
  plot(x,Lag(2,xdata,x))
  hold on
  plot(x,Lag(4,xdata,x))
  plot(xdata,0*xdata,'o')
%plot(xdata,Lag(2,xdata,xdata),'or')
```

```
  title('Lagrange multiplier functions L_2 , L_5')
  legend('L_2','L_5','Location','NorthWest')
  grid on
%%
figure
%subplot(2,1,2)
hold on
plot(xdata,0*xdata,'o')
plot(xdata,ones(1,N))
grid on
for k=1:N
  plot(x,Lag(k,xdata,x))
  plot(xdata,Lag(k,xdata,xdata),'xk')
end
title('Whole set of Lagrange multipliers for xdata')
```

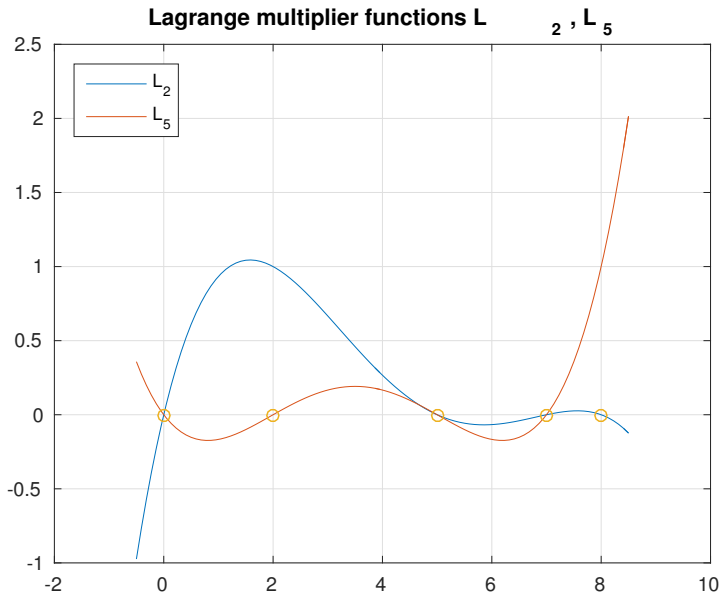


Figure 2: Lagrange multiplier functions L_2, L_5

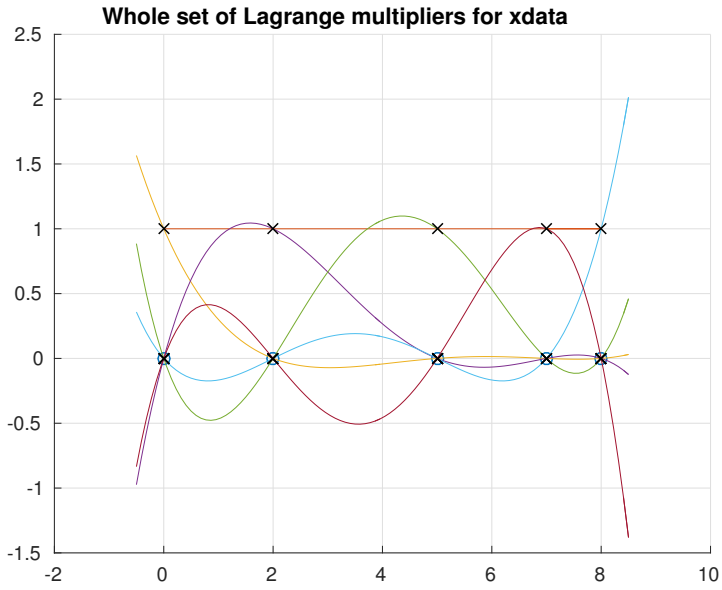


Figure 3: All Lagrange multiplier functions for xdata

Kts, vihko s. 5, ehkä parempi esittää niin ... [3]

Example 2

1.5 Interpolation error

In case the y-values are the values of a known smooth function f, it is possible to derive an error formula.

Theorem 7 ***Error formula for polynomial interpolation*** *Let f have $n+1$ continuos derivatives on the interval where the points $x_0, \ldots, x_n$ lie.*

Let p_n be the interpolation polynomial for this data

Then the error $\epsilon_n(x) = f(x) - p_n(x)$ satisfies

[3] Jossain on tuo maaginen vihko ehkä vuodelta 2003, nyt on 6.4.2014 ja nyt 9.8.2016

$$\epsilon_n(x) = (x - x_0)(x - x_1)\dots(x - x_n)\frac{f^{(n+1)}(\xi)}{(n+1)!},$$

where ξ is a point on the interval including all the x_k-points and x.

Proof: The proof is a straightworward application of *Rolle's* theorem (apply n times). Let's skip it here.

Note 8 *The point ξ is unknown, so all we can do is to use the maximum of the $(n+1)^{st}$ derivative. Computer algebra is needed, one possibility is to use Matlab's symbolic toolbox. The maximum need'n be accurate, figure is enough. There are cases where the estimate is very coarse, but it gives a basis for the analysis of it's behaviour.*