MAT 460 Numerical Differential Equations

Spring 2016

Michael Karow, Andrea Dziubek

Lecture 8

Topics: Horner's method, Polynomial Interpolation

Horner's method I

Consider the polynomial $p(x) = a_4\,x^4 + a_3\,x^3 + a_2\,x^2 + a_1\,x + a_0, \qquad a_k, x \in \mathbb{C}.$

Problem: Compute $p(x_0)$ for a given x_0.

First solution: Compute the powers of x_0, multiply them with the coefficients a_k, and add these results. This method requires 3+4=7 multiplications.

Better solution: Write the polynomial in the form

$$p(x_0) = (((a_4\,x_0 + a_3)\,x_0 + a_2)\,x_0 + a_1)\,x_0 + a_0$$

and evaluate from inside to outside: first compute $a_4\,x_0 + a_3$, then multiply the result by x_0, add a_2, and so on. This method requires only 4 multiplications.

Algorithm: Set

$$\begin{aligned} q_3 &= a_4 \\ q_2 &= q_3\,x_0 + a_3 \\ q_1 &= q_2\,x_0 + a_2 \\ q_0 &= q_1\,x_0 + a_1 \\ p_0 &= q_0\,x_0 + a_0 \end{aligned}$$

We get $p_0 = p(x_0)$. The table with the intermediate results is called **(Horner-scheme)**:

a_4	a_3	a_2	a_1	a_0
0	$+q_3\,x_0$	$+q_2\,x_0$	$+q_1\,x_0$	$+q_0\,x_0$
$q_3 = a_4$	q_2	q_1	q_0	$p_0 = p(x_0)$

Horner's method II

The intermediate results $q_3, \ldots, q_0$ in the Horner-scheme are the coefficients of the polynomial that we get when we divide the polynomial $p(x)$ by the linear factor $x - x_0$. In particular we have:

$$p(x) = (q_3\,x^3 + q_2\,x^2 + q_1\,x + q_0)\,(x - x_0) + p(x_0). \qquad (*)$$

Proof:

$$\begin{aligned} p(x) &= a_4\,x^4 + a_3\,x^3 + a_2\,x^2 + a_1\,x + a_0 \\ &= a_4\,x^3\,(x - x_0) + (a_4x_0 + a_3)\,x^3 + a_2\,x^2 + a_1\,x + a_0 \\ &= a_4\,x^3\,(x - x_0) + (a_4x_0 + a_3)\,x^2(x - x_0) + ((a_4x_0 + a_3)\,x_0 + a_2)x^2 + a_1\,x + a_0 \\ &= a_4\,x^3\,(x - x_0) + (a_4x_0 + a_3)\,x^2(x - x_0) + ((a_4x_0 + a_3)\,x_0 + a_2)x\,(x - x_0) \\ &\quad (((a_4x_0 + a_3)\,x_0 + a_2)x_0 + a_1)\,x + a_0 \\ &= a_4\,x^3\,(x - x_0) + (a_4x_0 + a_3)\,x^2(x - x_0) + ((a_4x_0 + a_3)\,x_0 + a_2)x\,(x - x_0) \\ &\quad (((a_4x_0 + a_3)\,x_0 + a_2)x_0 + a_1)\,(x - x_0) + (((a_4x_0 + a_3)\,x_0 + a_2)x_0 + a_1)x_0 + a_0 \\ &= (q_3\,x^3 + q_2\,x^2 + q_1\,x + q_0)\,(x - x_0) + p(x_0). \end{aligned}$$

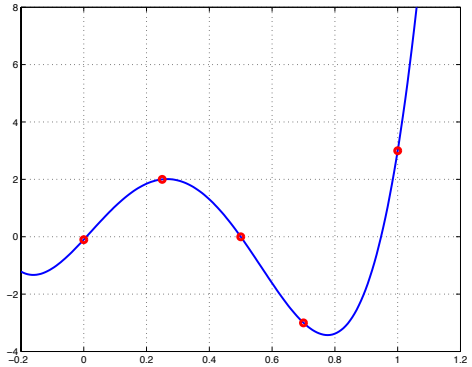
All statements apply similarly to polynomials p of any degree ≥ 1.

Lagrange-interpolation problem.

Consider the $n+1$ interpolation points $(x_j, f_j) \in \mathbb{C}$, $j = 1, \ldots, n+1$, where all points x_j are distinct.

Problem: Find a polynomial of degree $\leq n$, such that

$$p(x_j) = f_j \qquad j = 1, \ldots, n+1.$$



The Lagrange-interpolation problem always has a unique solution.

See next slides.

Three different methods to compute interpolation polynomials:

1. Vandermonde-method (this is not the official name):
   Solve a linear equation system with a Vandermonde-matrix.

2. Lagrange-method:
   Represent the interpolation polynomial in terms of Lagrange-basis polynomials.
   Conclusion: barycentric formula.

3. Newton-method:
   Represent the interpolation polynomial in terms of divided differences.

We need polynomial interpolation for

1. numerical differentiation (finite difference formulas),

2. numerical integration,

3. numerical solution of differential equations.

Vandermonde-method

Let $p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0$.

Find the coefficients a_k, such that $p(x_j) = f_j$, $j = 1, \ldots, n+1$.

This gives the linear equation system

$$\begin{aligned} a_0 + a_1 x_1 + a_2 x_1^2 + \ldots + a_{n-1} x_1^{n-1} + a_n x_1^n &= f_1 \\ a_0 + a_1 x_2 + a_2 x_2^2 + \ldots + a_{n-1} x_2^{n-1} + a_n x_2^n &= f_2 \\ &\vdots \\ a_0 + a_1 x_{n+1} + a_2 x_{n+1}^2 + \ldots + a_{n-1} x_{n+1}^{n-1} + a_n x_{n+1}^n &= f_{n+1}. \end{aligned}$$

In matrix-vector-notation:

$$\underbrace{\begin{bmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^n \\ 1 & x_2 & x_2^2 & \ldots & x_2^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n+1} & x_{n+1}^2 & \ldots & x_{n+1}^n \end{bmatrix}}_{V} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n+1} \end{bmatrix}$$

The matrix V is called **Vandermonde-matrix**.

For large n this matrix is ill-conditioned.

For $n = 20$ and $x_j = j/n$ Matlab computes $cond_2(V) \approx 10^{16}$.

Vandermonde-method II

Matlab and Python compute the coefficients of an interpolation polynomial with the Vandermonde-method. The command is

```
a=polyfit(x,f,n)
```

where `n` is the degree of the polynomial. If `n` is smaller than the number of points - 1, the interpolation problem does not have a solution. Then a polynomial is computed which approximates the interpolation points in a least square sense (polynomial curve fitting).

Warning. In Matlab the coefficients of a polynomial are numbered oppositely compared to on this slides: The coefficient before x^n is in Matlab $a(1)$ etc. Similarly, the Matlab comand `vander(x)` does not create the Vandermonde-matrix from the last slide but gives instead

$$\begin{bmatrix} x_1^n & \ldots & x_1^2 & x_1 & 1 \\ x_2^n & \ldots & x_2^2 & x_2 & 1 \\ \vdots & & \vdots & \vdots & \vdots \\ x_{n+1}^n & \ldots & x_{n+1}^2 & x_{n+1} & 1 \end{bmatrix}$$

To evaluate a polynomial at the points x_j use the Matlab or Python function `polyval`.

Lagrange-method

Lets write out an interpolation polynomial without computing anything.

Example: The polynomial p satisfying the conditions

$$p(x_1) = f_1, \quad p(x_2) = f_2, \quad p(x_3) = f_3,$$

is

$$p(x) = f_1 \underbrace{\frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)}}_{L_1(x)} + f_2 \underbrace{\frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)}}_{L_2(x)} + f_3 \underbrace{\frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}}_{L_3(x)}.$$

The polynomials L_k are called **Lagrangian basis polynomials**. For $n+1$ points x_k they have the general form

$$L_k(x) = \frac{(x-x_1)\ldots(x-x_{k-1})(x-x_{k+1})\ldots(x-x_{n+1})}{(x_k-x_1)\ldots(x_k-x_{k-1})(x_k-x_{k+1})\ldots(x_k-x_{n+1})}.$$

And we have

$$L_k(x_j) = \begin{cases} 1 & \text{for } j = k \\ 0 & \text{sonst.} \end{cases}$$

The interpolation polynomial corresponding to the data (x_j, f_j) is $p(x) = \sum_{k=1}^{n+1} f_k L_k(x)$.

Disadvantage of the Lagrange-basis polynomials:

The computational effort and numerical error to evaluate the polynomials are too large. Adding one interpolation point requires re-computation of all polynomials.

Illustration: Some Lagrangian-basis polynomials

The pictures show some Lagrangian-basis polynomials (blue curves) to 10 equidistant points (red marks).

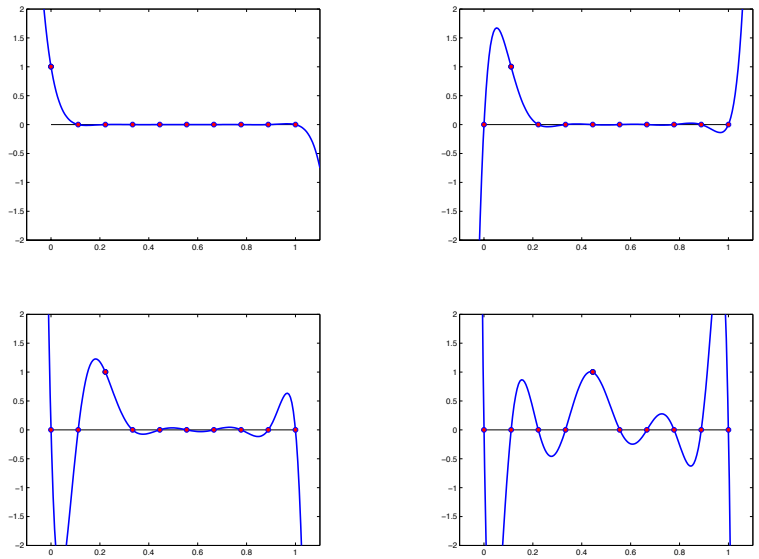
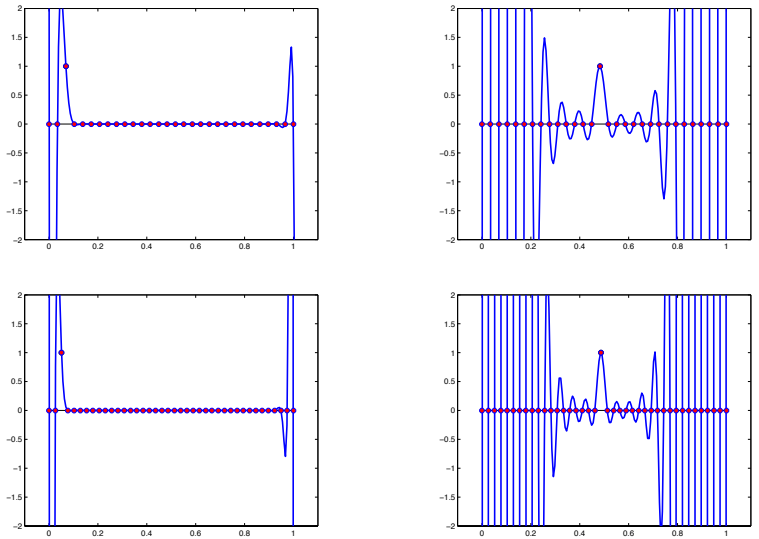


Illustration: Some Lagrangian-basis polynomials

The pictures show some Lagrangian-basis polynomials (blue curves) to 30 (top) and 40 (bottom) equidistant points (red marks), resp.



Conclusion from Lagrange-basis polynomials: barycentric formula

A cool trick to evaluate an interpolation polynomial numerically stable is to represent it as the quotient of partial fractions. This representation is called barycentric formula.

Example with 3 points: Let p be the polynomial with $p(x_1) = f_1, p(x_2) = f_2, p(x_3) = f_3$. Then

$$\begin{aligned} p(x) &= f_1 \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} + f_2 \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} + f_3 \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)} \\ &= (x-x_1)(x-x_2)(x-x_3) * \\ &\quad \left(f_1 \frac{\frac{1}{x-x_1}}{(x_1-x_2)(x_1-x_3)} + f_2 \frac{\frac{1}{x-x_2}}{(x_2-x_1)(x_2-x_3)} + f_3 \frac{\frac{1}{x-x_3}}{(x_3-x_1)(x_3-x_2)} \right) \\ &= (x-x_1)(x-x_2)(x-x_3) \left(f_1 \frac{w_1}{x-x_1} + f_2 \frac{w_2}{x-x_2} + f_3 \frac{w_3}{x-x_3} \right), \qquad (*) \end{aligned}$$

where

$$w_1 = \frac{1}{(x_1-x_2)(x_1-x_3)}, \quad w_2 = \frac{1}{(x_2-x_1)(x_2-x_3)}, \quad w_3 = \frac{1}{(x_3-x_1)(x_3-x_2)}.$$

In particular the constant polynomial with value 1 has the representation

$$1 = (x-x_1)(x-x_2)(x-x_3) \left(\frac{w_1}{x-x_1} + \frac{w_2}{x-x_2} + \frac{w_3}{x-x_3} \right). \qquad (**)$$

Division of $(*)$ by $(**)$ gives

$$p(x) = \frac{f_1 \frac{w_1}{x-x_1} + f_2 \frac{w_2}{x-x_2} + f_3 \frac{w_3}{x-x_3}}{\frac{w_1}{x-x_1} + \frac{w_2}{x-x_2} + \frac{w_3}{x-x_3}}$$

This is the **barycentric formula** for p.

Newton-method I

Goal: Fast computation of the coefficients c_k in the interpolation polynomial p to interpolation points (x_j, f_j) with respect to Newton-basis polynomials.

Representation of the interpolation polynomial w.r.t. **monomial-basis**:

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

Representation of the interpolation polynomial w.r.t. **Newton-basis polynomials**:

$$p(x) = c_0 + c_1\,(x-x_1) + c_2\,(x-x_1)(x-x_2) + \ldots + c_n\,(x-x_1)(x-x_2)\ldots(x-x_n).$$

We introduce the following notation for the coefficients of the highest powers in the interpolation polynomial. (Explained on the next slides.)

Definition:

$$f[x_1,\, x_2\, \ldots, x_j] := c_j.$$

This coefficient is called the n-**th divided difference**.

Note: The order of the numbering of the points does not matter, i. e.

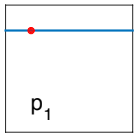
$$f[x_1,\, x_2,\, x_3] = f[x_1,\, x_2,\, x_3] = f[x_3,\, x_1,\, x_2].$$

Newton-method II

Consider the interpolation polynomial for $1-3$ points:

Interpolation polynomial for 1 point (x_1, f_1):

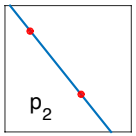
$$p_1(x) = f_1$$



The coefficient of the highest power is f_1, i. e. $f[x_1] = f_1$.

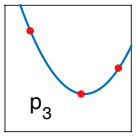
Interpolation polynomial for 2 points (x_1, f_1), (x_2, f_2):

$$p_2(x) = f_1 + \underbrace{\frac{f_2 - f_1}{x_2 - x_1}}_{f[x_1,\, x_2]}(x - x_1).$$



Interpolation polynomial for 3 points (x_1, f_1), (x_2, f_2), (x_3, f_3):

$$p_3(x) = f_1 + \underbrace{\frac{f_2 - f_1}{x_2 - x_1}}_{f[x_1,\, x_2]}(x - x_1) + \underbrace{\frac{\frac{f_3 - f_2}{x_3 - x_2} - \frac{f_2 - f_1}{x_2 - x_1}}{x_3 - x_1}}_{f[x_1,\, x_2,\, x_3]}(x - x_1)(x - x_2)$$



This motivates the term 'divided differences' for the coefficients of the highest powers

$$f[x_1,\, x_2 \ldots, x_n,\, x_{n+1}].$$

Newton-method III: Aitken lemma and recursion-formula

Lemma: Consider the (x_j, f_j), $j = 1, \ldots, n+1$. Let $p_{1,\ldots,n}$ the polynomial of degree $\leq n-1$, which interpolates the interpolation points with indices 1 to n, and let $p_{2,\ldots,n+1}$ the polynomial of degree $\leq n-1$, which interpolates the interpolation points with indices 2 to $n+1$. Then the polynomial $p_{1,\ldots,n+1}$ with

$$p_{1,\ldots,n+1}(x) = \frac{(x - x_1)\, p_{2,\ldots,n+1}(x) - (x - x_{n+1})\, p_{1,\ldots,n}(x)}{x_{n+1} - x_1} \qquad (*)$$

is the interpolation polynomial to all points.

Proof: Check.

Conclusion: By definition the coefficients of the highest powers are:

$$\begin{aligned} &\text{for } p_{1,\ldots,n}: && f[x_1,\, x_2,\, \ldots,\, x_n] \\ &\text{for } p_{2,\ldots,n+1}: && f[x_2,\, \ldots,\, x_n,\, x_{n+1}] \\ &\text{for } p_{1,\ldots,n+1}: && f[x_1,\, x_2,\, \ldots,\, x_n,\, x_{n+1}] \end{aligned}$$

From $(*)$ follows the recursion-formula for the coefficients of the highest powers:

$$\begin{aligned} f[x_1,\, x_2,\, \ldots,\, x_n,\, x_{n+1}] &= \frac{f[x_2,\, x_3,\, \ldots,\, x_n,\, x_{n+1}] - f[x_1,\, x_2,\, \ldots,\, x_n]}{x_{n+1} - x_1} \\ &= \frac{f[x_1,\, x_2,\, \ldots,\, x_n] - f[x_2,\, x_3,\, \ldots,\, x_n,\, x_{n+1}]}{x_1 - x_{n+1}}. \end{aligned}$$

Newton-method IV: Application of the recursion-formula

Recursion-formula:

$$f[x_1,\, x_2,\, \ldots,\, x_n,\, x_{n+1}] = \frac{f[x_2,\, x_3,\, \ldots,\, x_n,\, x_{n+1}] - f[x_1,\, x_2,\, \ldots,\, x_n]}{x_{n+1} - x_1}.$$

Note: Ordering and numbering of the points in square brackets does not change the results of the computations.

Example:

Assume we have

$$f[-1,\, 0.5,\, 7,\, 2] = 5, \qquad f[2,\, 9,\, 0.5,\, -1] = 3.$$

The lists in square brackets differ beside in their order only in the numbers 7 and 9. Set $x_1 = 9$, $x_{n+1} = 7$. Then the divided difference corresponding to the given data is with the formula above given by

$$f[-1,\, 0.5,\, 7,\, 2,\, 9] = \frac{f[2,\, 9,\, 0.5,\, -1] - f[-1,\, 0.5,\, 7,\, 2]}{9 - 7} = \frac{3 - 5}{9 - 7} = -1.$$

Newton-method V: Algorithm to compute divided differences

With the recursion formula we can compute divided differences successively.

Scheme to compute divided differences:

x_1	$f_1 = f[x_1]$	$f[x_1,\, x_2]$	$f[x_1,\, x_2,\, x_3]$	$f[x_1,\, x_2,\, x_3,\, x_4]$
x_2	$f_2 = f[x_2]$	$f[x_2,\, x_3]$	$f[x_2,\, x_3,\, x_4]$	
x_3	$f_3 = f[x_3]$	$f[x_3,\, x_4]$		
x_4	$f_4 = f[x_4]$			

Explanation: The first two columns contain the interpolation data. To compute the 3rd column we evaluate the divided differences of each pair of values from the 2nd column. To compute the 4th column we evaluate the divided differences of each pair of values from the 3rd column, etc. The first row contains the coefficients of the interpolation polynomial p with respect to Newton-basis polynomials, i. e. we write

$$\begin{aligned} p(x) \;=\; & f[x_1] + f[x_1,\, x_2]\,(x - x_1) + f[x_1,\, x_2,\, x_3]\,(x - x_1)(x - x_2) \\ & + f[x_1,\, x_2,\, x_3,\, x_4]\,(x - x_1)(x - x_2)(x - x_3). \end{aligned}$$

To add a new point (x_5, f_5), simply add the corresponding values at the diagonal (see next slide).

Scheme to compute divided differences:

x_1	$f_1 = f[x_1]$	$f[x_1,\ x_2]$	$f[x_1,\ x_2,\ x_3]$	$f[x_1,\ x_2,\ x_3,\ x_4]$	$f[x_1,\ x_2,\ x_3,\ x_4,\ x_5]$
x_2	$f_2 = f[x_2]$	$f[x_2,\ x_3]$	$f[x_2,\ x_3,\ x_4]$	$f[x_2,\ x_3,\ x_4,\ x_5]$	
x_3	$f_3 = f[x_3]$	$f[x_3,\ x_4]$	$f[x_3,\ x_4,\ x_5]$		
x_4	$f_4 = f[x_4]$	$f[x_4,\ x_5]$			
x_5	$f_5 = f[x_5]$				

Explanation: After the blue values are computed, we can compute the red values with the recursion-formula.

Closed formula for divided differences

The interpolation polynomial to the interpolation points (x_j, f_j), $j = 1, \ldots, n+1$ is

with Newton:

$$p(x) \;=\; f[x_1] + f[x_1,\ x_2]\,(x - x_1) + \ldots + f[x_1,\ x_2,\ \ldots,\ x_{n+1}]\,(x - x_1)(x - x_2)\ldots(x - x_n)$$

with Lagrange:

$$p(x) \;=\; \sum_{k=1}^{n+1} f_k\, L_k(x) \;=\; \sum_{k=1}^{n+1} f_k \frac{(x - x_1)\ldots(x - x_{k-1})(x - x_{k+1})\ldots(x - x_{n+1})}{(x_k - x_1)\ldots(x_k - x_{k-1})(x_k - x_{k+1})\ldots(x_k - x_{n+1})}.$$

Comparing the coefficients of the highest powers x^n gives

$$f[x_1,\ x_2,\ \ldots,\ x_{n+1}] = \sum_{k=1}^{n+1} \frac{f_k}{(x_k - x_1)\ldots(x_k - x_{k-1})(x_k - x_{k+1})\ldots(x_k - x_{n+1})}.$$

Example ($n = 2$):

$$f[x_1,\ x_2,\ x_3] \;=\; \frac{f_1}{(x_1 - x_2)(x_1 - x_3)} + \frac{f_2}{(x_2 - x_1)(x_2 - x_3)} + \frac{f_3}{(x_3 - x_1)(x_3 - x_2)}$$