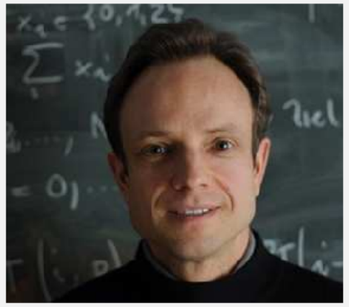
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
[Lecture → Chapter 0]: Introduction

Prof. R. Hiptmair, SAM, ETH Zurich

Date: August 17, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Duration: 16 minutes

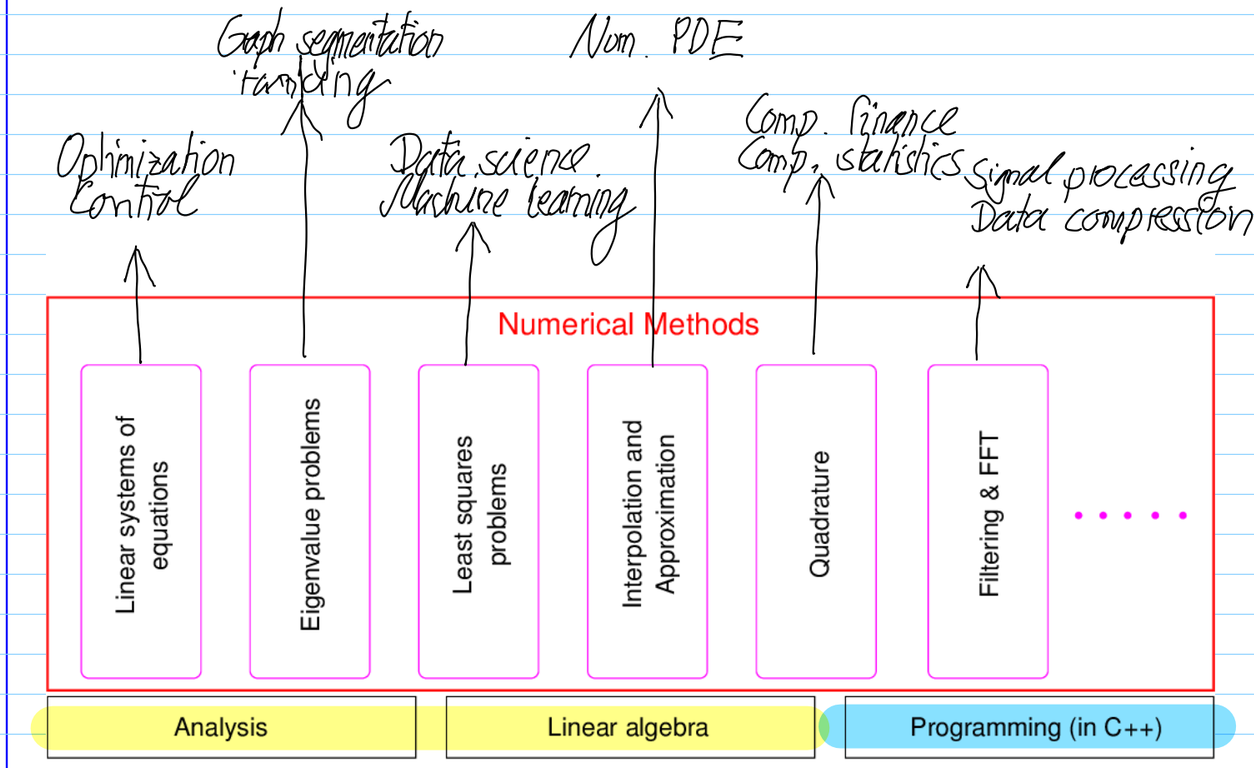
Related to Section 0.1. of Lecture Document

Course Topic

Numerical Methods (≠ numerical analysis)

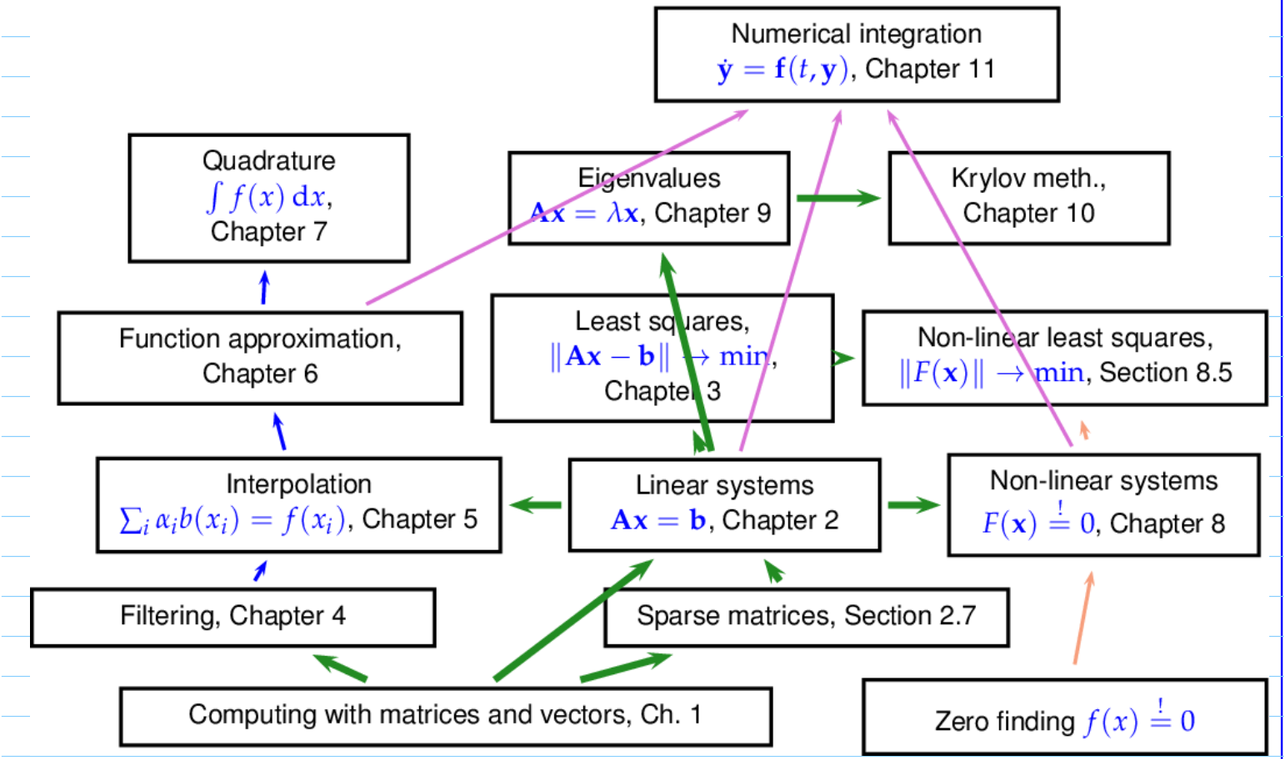
≙ Algorithms for evaluating functions and solving equations in $\mathbb{R}$

↳ focus of this course



NEW: From autumn term 2022 the course for CS students includes a two-week introduction to C++ in the beginning.

Topics : Very diverse



▷ A toolkit course

Hardly anyone will need everything covered in this course, but *most of you will need something*.

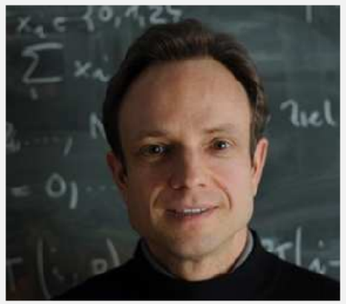
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 1.1.1–Section 1.1.2: Notations and Classes of Matrices

Prof. R. Hiptmair, SAM, ETH Zurich

Date: August 20, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- Concept of "Matrix" and "Vector" from linear algebra.

Dependencies. None

Duration: 10 minutes

I. Computing with Matrices and Vectors

1.1. Fundamentals

1.1.1. Notations

Vector: $\underline{x} \in \mathbb{K}^n$ ($\mathbb{K} = \mathbb{R}, \mathbb{C}$)

$(\underline{x})_i \triangleq$ i-th component of $\underline{x}$, $i = 1, \dots, n$

Matrix: $A \in \mathbb{K}^{n,m}$

$$\mathbf{A} := \begin{bmatrix} a_{11} & \dots & a_{1m} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nm} \end{bmatrix}$$

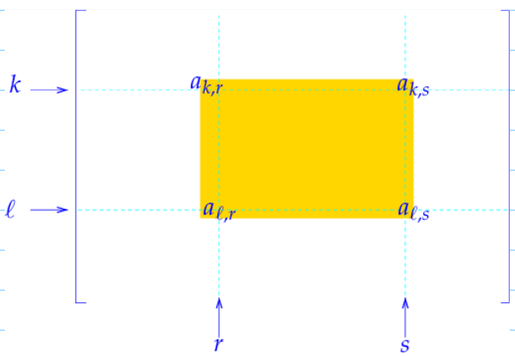
→ matrix entry/matrix element $(\mathbf{A})_{i,j} := a_{ij}$, $1 \le i \le n, 1 \le j \le m$,

→ i-th row, $1 \le i \le n$: $(\mathbf{A})_{i,:} := [a_{i,1}, \dots, a_{i,m}]$,

→ j-th column, $1 \le j \le m$: $(\mathbf{A})_{:,j} := [a_{1,j}, \dots, a_{n,j}]^\top$,

→ matrix block (sub-matrix) $(\mathbf{A})_{k:\ell,r:s} := [a_{ij}]_{\substack{i=k,\dots,\ell \\ j=r,\dots,s}}$, $1 \le k \le \ell \le n$, $1 \le r \le s \le m$.

$\in \mathbb{K}^{\ell-k+1, s-r+1}$



▽ Linear algebra indexing starting from 1

vs. C++/Python indexing starting from 0!

1.1.2. Classes of Matrices

Identity matrix: $\mathbf{I} := \mathbf{I}_n := \begin{bmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{bmatrix} \in \mathbb{K}^{n,n}$,

Zero matrix: $\mathbf{O} := \mathbf{O}_{n,m} := \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix} \in \mathbb{K}^{n,m}$,

Diagonal matrix: $\operatorname{diag}(d_1, \dots, d_n) := \begin{bmatrix} d_1 & & 0 \\ & \ddots & \\ 0 & & d_n \end{bmatrix} \in \mathbb{K}^{n,n}$, $d_j \in \mathbb{K}$, $j = 1, \dots, n$.

▷ Do review questions 1.1.2.9

NumCSE@ETHZ

Review questions 1.2.9

(Try to answer without consulting notes)

A:

Give a compact notation for the row vector containing the diagonal entries of a square matrix $\mathbf{S} \in \mathbb{R}^{n,n}$, $n \in \mathbb{N}$.

B:

How can you write down the $s \times s$-submatrix, $s \in \mathbb{N}$, in the upper right corner of $\mathbf{C} \in \mathbb{R}^{n,m}$, $n, m \geq s$.

C:

We consider two matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n,m}$, both with at most $N \in \mathbb{N}$ non-zero entries. What is the maximal number of non-zero entries of $\mathbf{A} + \mathbf{B}$?

D:

A matrix $\mathbf{A} \in \mathbb{R}^{n,m}$ enjoys the following property (banded matrix):

$$i \in \{1, \dots, n\},\ j \in \{1, \dots, m\},\ i - j \notin \{-B_-, \dots, B_+\} \quad \Rightarrow \quad (\mathbf{A})_{ij} = 0,$$

for given $B_-, B_+ \in \mathbb{N}_0$. What is the maximal number of non-zero entries of $\mathbf{A}$.

E:

A matrix $\mathbf{A}$ with real entries is known to be **skew-symmetric**: $\mathbf{A}^\top = -\mathbf{A}$ What does this tell us about A and its entries?

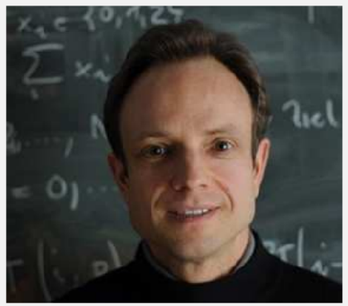
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 1.2.1: EIGEN

Prof. R. Hiptmair, SAM, ETH Zurich

Date: August 24, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- Basics of C++
- Classes and methods in C++
- Templates in C++

Dependencies. None

Duration: minutes

I. Computing with Matrices and Vectors

1.2. Software and Libraries

1.2.1. Eigen

→ Header-only C++ template library for numerical LA

Fundamental data type :

Matrix

- fixed size (size known at compile time)
- dynamic (size known only at run time)
- sparse matrix

(2)

C++ code 1.2.1.2: Vector type and their use in EIGEN → GITLAB

```
1  #include <Eigen/Dense>
2
3  template<typename Scalar>
4  void eigenTypeDemo(unsigned int dim)
5  {
6    // General dynamic (variable size) matrices
7    using dynMat_t = Eigen::Matrix<Scalar,Eigen::Dynamic,Eigen::Dynamic>;
8    // Dynamic (variable size) column vectors
9    using dynColVec_t = Eigen::Matrix<Scalar,Eigen::Dynamic,1>;
10   // Dynamic (variable size) row vectors
11   using dynRowVec_t = Eigen::Matrix<Scalar,1,Eigen::Dynamic>;
12   using index_t = typename dynMat_t::Index;
13   using entry_t = typename dynMat_t::Scalar;
14
15   // Declare vectors of size 'dim'; not yet initialized
16   dynColVec_t colvec(dim);
17   dynRowVec_t rowvec(dim);
18   // Initialisation through component access
19   for(index_t i=0; i<colvec.size(); ++i) colvec[i] = (Scalar)i;
20   for(index_t i=0; i<rowvec.size(); ++i) rowvec[i] = (Scalar)1/(i+1);
21   colvec[0] = (Scalar)3.14; rowvec[dim-1] = (Scalar)2.718;
22   // Form tensor product, a matrix, see Section 1.3.1
23   dynMat_t vecprod = colvec*rowvec;
24   const int nrows = vecprod.rows();
25   const int ncols = vecprod.cols();
26 }
```

→ column vector

→ row vector

accessing a vector comp.

→ matrix product

Line 23 : $\begin{bmatrix} & \\ & \end{bmatrix} * \begin{bmatrix} \quad \end{bmatrix} = \begin{bmatrix} & \\ & \end{bmatrix}$

tensor product of vectors

§ 1.2.1.5 Accessing sub-matrix

`M.block(i,j,p,q)` method for matrix types

→ $(M)_{i+1:i+p,\,j+1:j+q}$

▽ Eigen uses indexing from 0

C++ code 1.2.1.6: Demonstration code for access to matrix blocks in EIGEN → GITLAB

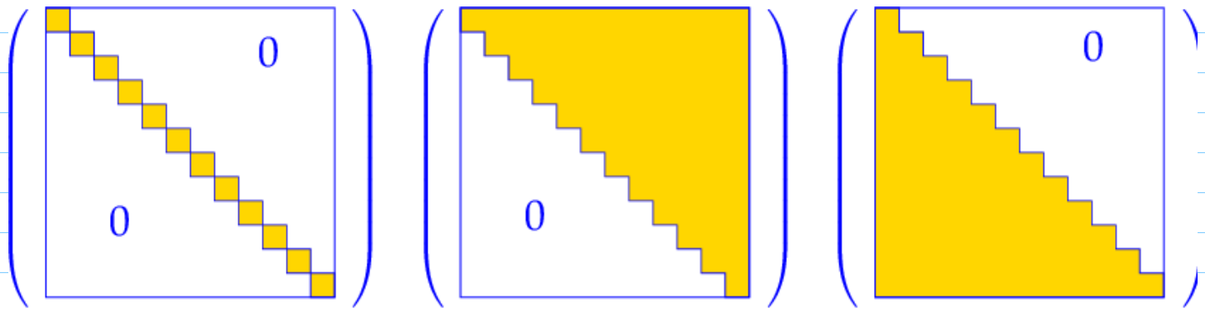
```
2  template<typename MatType>
3  void blockAccess(Eigen::MatrixBase<MatType> &M)
4  {
5    using index_t = typename Eigen::MatrixBase<MatType>::Index;
6    using entry_t = typename Eigen::MatrixBase<MatType>::Scalar;
7    const index_t nrows(M.rows()); // No. of rows
8    const index_t ncols(M.cols()); // No. of columns
9
10   cout << "Matrix M = " << endl << M << endl; // Print matrix
11   // Block size half the size of the matrix
12   index_t p = nrows/2, q = ncols/2;
13   // Output submatrix with left upper entry at position (i,i)
14   for(index_t i=0; i < min(p,q); i++)
15     cout << "Block (" << i << ',' << i << ',' << p << ',' << q
16          << ") = " << M.block(i,i,p,q) << endl;
17   // l-value access: modify sub-matrix by adding a constant
18   M.block(1,1,p,q) += Eigen::MatrixBase<MatType>::Constant(p,q,1.0);
19   cout << "M = " << endl << M << endl;
20   // r-value access: extract sub-matrix
21   MatrixXd B = M.block(1,1,p,q);
22   cout << "Isolated modified block = " << endl << B << endl;
23   // Special sub-matrices
24   cout << p << " top rows of m = " << M.topRows(p) << endl;
25   cout << p << " bottom rows of m = " << M.bottomRows(p) << endl;
26   cout << q << " left cols of m = " << M.leftCols(q) << endl;
27   cout << q << " right cols of m = " << M.rightCols(p) << endl;
28   // r-value access to upper triangular part
29   const MatrixXd T = M.template triangularView<Upper>(); //
30   cout << "Upper triangular part = " << endl << T << endl;
31   // l-value access to upper triangular part
32   M.template triangularView<Lower>() *= -1.5; //
33   cout << "Matrix M = " << endl << M << endl;
34 }
```

→ block read access

→ write access

} special sub-matrix access operations

③



→ Review questions 1.2.1.14

Review questions 1.2.1.14

[tackle using only your memory and the Eigen documentation]

A:

Outline a C++ function

```
template <typename Matrix>
  void replaceWithId(Eigen::DenseBase<Matrix> &M);
```

that checks whether the matrix is an $n \times n$-matrix with even $n \in \mathbb{N}$ and then replaces its upper right $n/2 \times n/2$-block with an identity matrix. Do not use any C++ loops.

B:

Given an **Eigen::VectorXd** object v ($\leftrightarrow \mathbf{v} \in \mathbb{R}^n$), sketch a C++ code snippet that replaces it with a vector $\widetilde{\mathbf{v}}$ defined by

$$(\widetilde{\mathbf{v}})_i := \begin{cases} (\mathbf{v})_n & \text{for } i = 1\,, \\ (\mathbf{v})_{i-1} & \text{for } i = 2,\ldots,n\,. \end{cases}$$

Do not use C++ loops. Can you see a problem?

(4)

C :

Given a matrix $\mathbf{M} \in \mathbb{R}^{m,n}$ stored in an **Eigen::MatrixXd** object `M` write down a C++ code snippet that initializes another variable `Mext` of type **Eigen::MatrixXd** corresponding to

$$\widetilde{\mathbf{M}} := \begin{bmatrix} \mathbf{M} & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix} \in \mathbb{R}^{m+1,n+1}$$

using EIGEN's << matrix construction operator EIGEN documentation.

D :

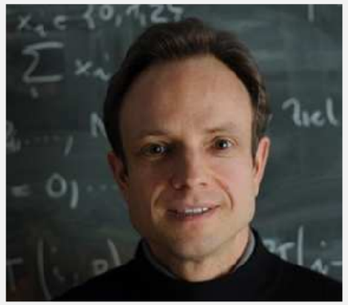
Learn about the methods `head()` and `tail()` from the EIGEN documentation and express them by means of the `block()` method applied to the same variable.

(You may find more review questions in the lecture document)

ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 1.2.3: (Dense) Matrix Storage Formats

Prof. R. Hiptmair, SAM, ETH Zurich



Date: August 24, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich

Prerequisites.
- Matrices in EIGEN

Dependencies.
- Section 1.2.1

Duration: 18 minutes

I. Computing with Matrices and Vectors

1.2. Software and Libraries

1.2.3. (Dense) Matrix Storage Formats

Always: The entries of a (generic, dense) $A \in \mathbb{K}^{m,n}$ are stored in a contiguous linear array of size $m \cdot n$

Exception: structured / sparse

↳ diagonal / banded / triangular ...

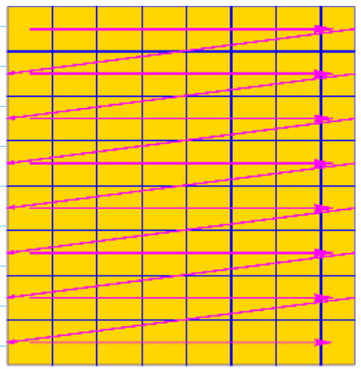
$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Row major (C-arrays, bitmaps, Python):

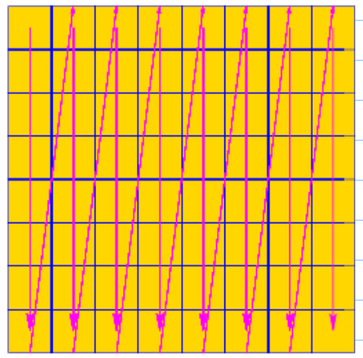
A_arr	1	2	3	4	5	6	7	8	9

Column major (Fortran, MATLAB, EIGEN):

A_arr	1	4	7	2	5	8	3	6	9



row major



column major

Index mapping (row major):

$(i,j) \to i \cdot n + j$

C++-indexing: $i \in \{0,\dots,m-1\}$, $j \in \{0,\dots,n-1\}$

(2)

Eigen: accessing underlying array

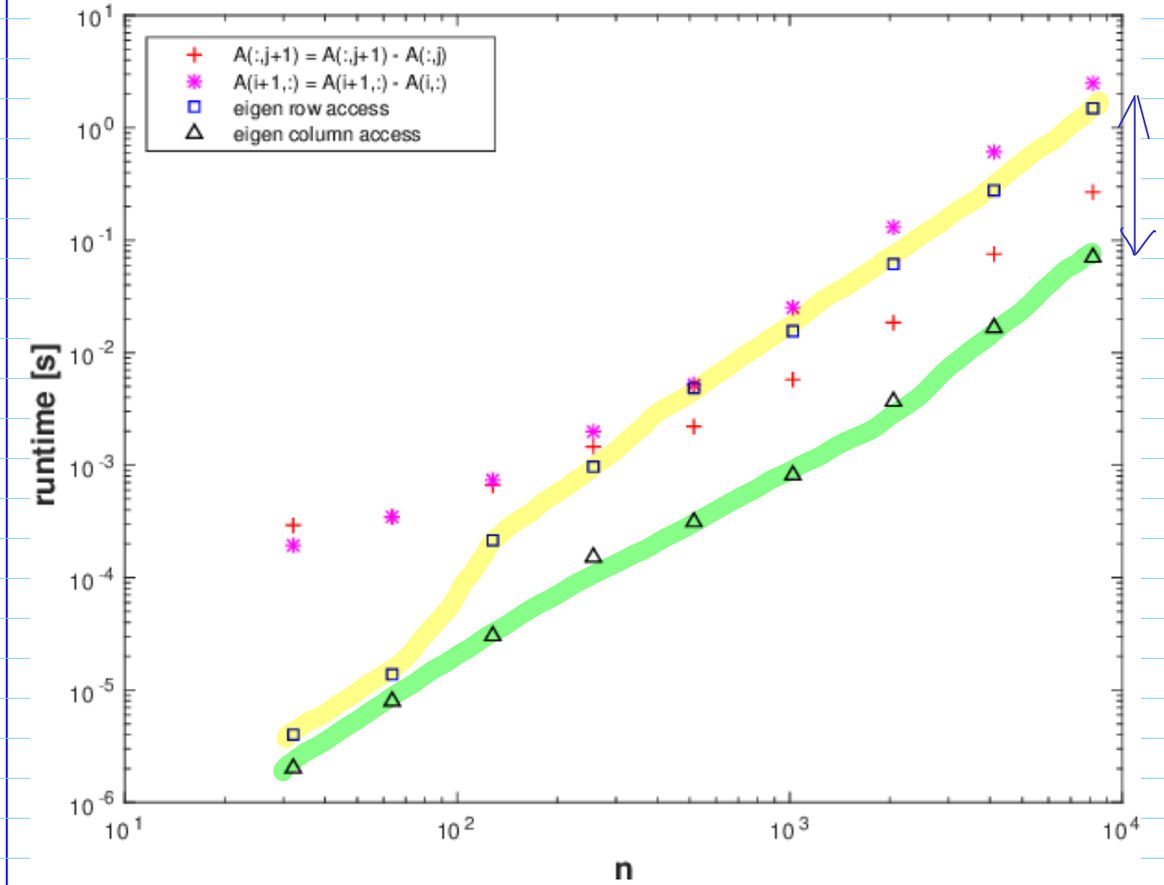
- $A(i) \hat{=}$ reference to the i-th element of the array
- A.data() $\hat{=}$ raw pointer, e.g. double *

Exp. 1.2.3.9. : Data layout impacts efficiency

C++ code 1.2.3.10: Timing for row and column oriented matrix access for EIGEN ⇢ GITLAB

```
void rowcolaccesstiming(void)
{
  const int K = 3; // Number of repetitions
  const int N_min = 5; // Smalles matrix size 32
  const int N_max = 13; // Scan until matrix size of 8192
  unsigned long n = (1L << N_min);
  Eigen::MatrixXd times(N_max-N_min+1,3);

  for(int l=N_min; l<= N_max; l++, n*=2) {
    Eigen::MatrixXd A = Eigen::MatrixXd::Random(n,n);
    double t1 = 1000.0;
    for(int k=0;k<K;k++) {
      auto tic = high_resolution_clock::now();
      for(int j=0; j < n-1; j++) A.row(j+1) -= A.row(j); // row access
      auto toc = high_resolution_clock::now();
      double t = (double)duration_cast<microseconds>(toc-tic).count()/1E6;
      t1 = std::min(t1,t);
    }
    double t2 = 1000.0;
    for(int k=0;k<K;k++) {
      auto tic = high_resolution_clock::now();
      for(int j=0; j < n-1; j++) A.col(j+1) -= A.col(j); //column access
      auto toc = high_resolution_clock::now();
      double t = (double)duration_cast<microseconds>(toc-tic).count()/1E6;
      t2 = std::min(t2,t);
    }
    times(l-N_min,0) = n;    times(l-N_min,1) = t1;    times(l-N_min,2) = t2;
  }
  std::cout << times << std::endl;
}
```

→ declare matrix object

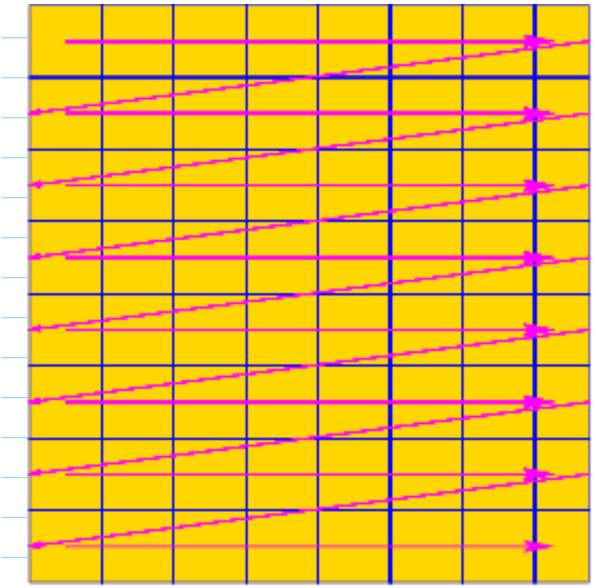


▷ row-wise ops. are much slower than col-wise access

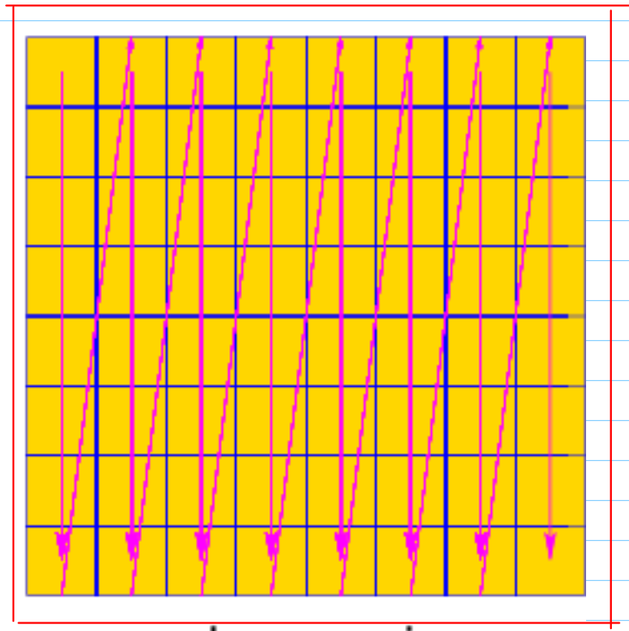
↓

incurr cache missing

NumCSE@ETHZ



row major



column major

↓

elements of all columns are contiguously in memory

▷ Review questions 1.2.3.11

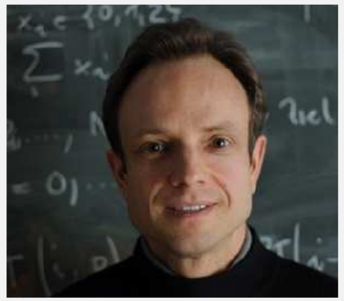
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 1.4: Computational Effort

Prof. R. Hiptmair, SAM, ETH Zurich

Date: August 24, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.
- Familiarity with basic matrix-vector operations

Dependencies. None

Duration: 55 minutes

Note: Possible minor *mismatch of video and tablet notes*!

[Corrections and updates can be incorporated into tablet notes only]

I. Computing with Matrices and Vectors

1.4. Computational Effort

Traditional: no. of elementary operations $+, -, *, /, \sqrt{\ }$

"Computational effort $\not\propto$ runtime"

The computational effort involved in a run of a numerical code is only loosely related to overall execution time on modern computers.

Mainly determined by:
- memory access pattern
- vectorization / pipelining

1.4.1. Asymptotic complexity

Definition 1.4.1.1. (Asymptotic) complexity

The **asymptotic complexity** of an algorithm characterises the worst-case dependence of its computational effort ($\rightarrow$ Def. 1.4.0.1) on one or more problem size parameter(s) when these tend to ∞.

$\text{cost}(n)$

Numerical LA: vector length / matrix dimension

Notation: "Landau-O"

$$\text{cost}(n) = O(n^\alpha), \quad \alpha > 0$$

for $n \to \infty$

$\Leftrightarrow \exists C>0, n_0 \in \mathbb{N}: \quad \text{cost}(n) \leq C n^\alpha \quad \forall n > n_0$

Tacit sharpness assumption:

$$\text{cost}(n) \neq O(n^\beta), \quad \forall \beta < \alpha$$

Meaning: Asymptotic complexity predicts the dependence of runtime on problem size for
- large problems
- a concrete implementation

e.g.: $\text{cost}(n) = O(n^2)$

then $n \longrightarrow 2*n$ leads to cost $\longrightarrow 4*$cost

1.4.2. Computational cost of basic numerical LA operations

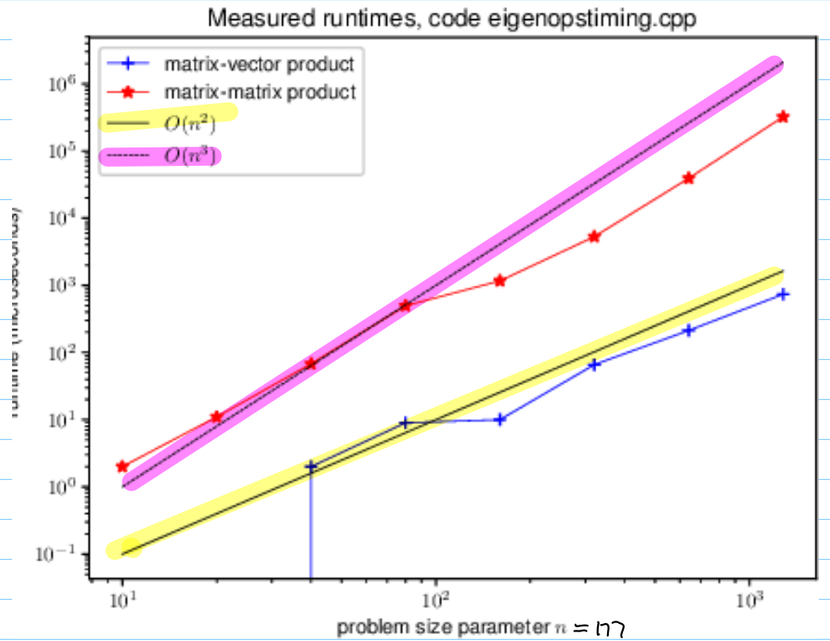
operation	description	#mul/div	#add/sub	asymp. complexity
dot product	$(\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^n) \mapsto \mathbf{x}^H\mathbf{y}$	n	$n-1$	$O(n)$ } optimal
tensor product	$(\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n) \mapsto \mathbf{x}\mathbf{y}^H$	nm	0	$O(mn)$ } optimal
Matrix×vector	$(\mathbf{x} \in \mathbb{R}^n, \mathbf{A} \in \mathbb{R}^{m,n}) \mapsto \mathbf{A}\mathbf{x}$	mn	$(n-1)m$	$O(mn)$ } optimal
matrix product(*)	$(\mathbf{A} \in \mathbb{R}^{m,n}, \mathbf{B} \in \mathbb{R}^{n,k}) \mapsto \mathbf{A}\mathbf{B}$	mnk	$mk(n-1)$	$O(mnk)$

↳ triple-loop implementation
(faster implementations available, cf Rem 1.4.2.2)

NumCSE@ETHZ

Exp. 1.4.2.1: Runtimes of M/V operation

Implementation based on Eigen, $m = n$



→ A doubly logarithmic plot

[If $\text{cost}(n) = O(n^\alpha) \Rightarrow$ data points aligned in a doubly logarithmic plot]

1.4.3. Some Tricks to Improve Complexity

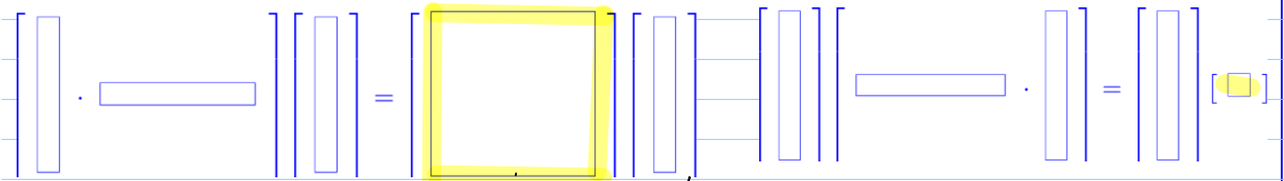
Ex 1.4.3.1 : Exploit associativity

$a, b, x \in \mathbb{R}^n$ (column vectors)

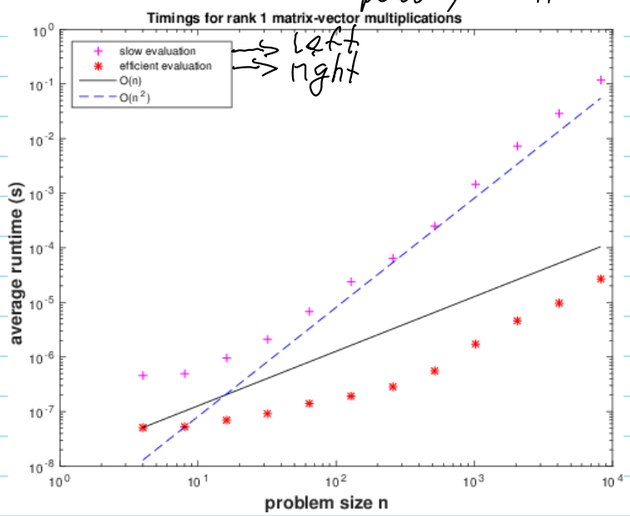
$$\mathbf{y} = \left(\mathbf{a}\mathbf{b}^\top\right)\mathbf{x}\,. \qquad (1.4.3.2) \qquad\qquad \mathbf{y} = \mathbf{a}\left(\mathbf{b}^\top\mathbf{x}\right)\,. \qquad (1.4.3.3)$$

```
T = (a*b.transpose())*x;   -> O(n^2)        t = a*b.dot(x);   => O(n)
```

Asymptotic cost = ?



temporary $n \times n$ matrix



Ex. 1.4.3.5 (Hidden summation)

$A, B \in \mathbb{R}^{n,p}$, $p \ll n$, $x \in \mathbb{R}^n$

```
Eigen::MatrixXd AB = A*B.transpose();   -> O(n^2 p)
y = AB.triangularView<Eigen::Upper>()*x;   -> O(n^2)      } -> O(n^2 p)
```

$y = \operatorname{triu}(AB^T)x$

↳ upper triangular part

We can do better?

- Case $p = 1$: $A, B \longleftrightarrow a, b \in \mathbb{R}^n$

$$(ab^T)_{i,j} = a_i b_j \,, \quad 1 \le i, j \le n$$

Trick: Matrix factorization

$$\mathbf{y} = \operatorname{triu}(\mathbf{a}\mathbf{b}^T)\mathbf{x} = \begin{bmatrix} a_1b_1 & a_1b_2 & \dots & \dots & a_1b_n \\ 0 & a_2b_2 & a_2b_3 & \dots & a_2b_n \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & a_nb_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ \\ \vdots \\ x_n \end{bmatrix}$$

scaling matrix · reverse cumulative summation · scaling matrix : $M \times V = O(n)$

$$= \begin{bmatrix} a_1 & & \\ & \ddots & \\ & & a_n \end{bmatrix} \left(\underbrace{\begin{bmatrix} 1 & 1 & \dots & \dots & 1 \\ 0 & 1 & 1 & \dots & 1 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ & & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & 1 \end{bmatrix}}_{T} \left(\begin{bmatrix} b_1 & & \\ & \ddots & \\ & & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ \\ \vdots \\ x_n \end{bmatrix} \right) \right) .$$

Efficient algorithm for $\begin{bmatrix} 1 & & \\ & \ddots & \\ 0 & & 1 \end{bmatrix} x$

$$\text{Cost}(n) = O(n)$$

▷ Total cost $= O(n)$

- Case $p > 1$:

Trick : $$AB^T = \sum_{\ell=1}^{p} \underbrace{(A)_{:,\ell}(B)_{:,\ell}^T}_{\in \mathbb{R}^{n,n}}$$

trim-operation is linear : $\text{trim}(AB^T) = \sum_{\ell=1}^{p} \text{trim}((A)_{:,\ell}(B)_{:,\ell}^T)$

▷ $\text{trim}(AB^T)x = \sum_{\ell=1}^{p} \text{trim}((A)_{:,\ell}(B)_{:,\ell}^T)x$

total cost $O(np)$

NumCSE@ETHZ

Ex. 1.4.3.8 (Reuse of intermediate results)

Definition 1.4.3.7. Kronecker product

The Kronecker product $\mathbf{A} \otimes \mathbf{B}$ of two matrices $\mathbf{A} \in \mathbb{K}^{m,n}$ and $\mathbf{B} \in \mathbb{K}^{l,k}$, $m,n,l,k \in \mathbb{N}$, is the $(ml) \times (nk)$-matrix

$$\mathbf{A} \otimes \mathbf{B} := \begin{bmatrix} (\mathbf{A})_{11}\mathbf{B} & (\mathbf{A})_{12}\mathbf{B} & \dots & \dots & (\mathbf{A})_{1,n}\mathbf{B} \\ (\mathbf{A})_{21}\mathbf{B} & (\mathbf{A})_{22}\mathbf{B} & & & \vdots \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ (\mathbf{A})_{m,1}\mathbf{B} & (\mathbf{A})_{m,2}\mathbf{B} & \dots & \dots & (\mathbf{A})_{m,n}\mathbf{B} \end{bmatrix} \in \mathbb{K}^{ml,nk}.$$

↑ matrix-block notation

Task : Given $A, B \in \mathbb{R}^{n,n}$, $\underline{x} \in \mathbb{R}^{n^2}$ we want to compute $\underline{y} = \underbrace{(A \otimes B)}_{\in \mathbb{R}^{n^2,n^2}} \underline{x}$

[Naive implementation : cost $= O(n^4)$]

$$(\mathbf{A} \otimes \mathbf{B})\mathbf{x} = \begin{bmatrix} (\mathbf{A})_{1,1}\mathbf{B}\mathbf{x}^1 + (\mathbf{A})_{1,2}\mathbf{B}\mathbf{x}^2 + \cdots + (\mathbf{A})_{1,n}\mathbf{B}\mathbf{x}^n \\ (\mathbf{A})_{2,1}\mathbf{B}\mathbf{x}^1 + (\mathbf{A})_{2,2}\mathbf{B}\mathbf{x}^2 + \cdots + (\mathbf{A})_{2,n}\mathbf{B}\mathbf{x}^n \\ \vdots \\ \vdots \\ (\mathbf{A})_{m,1}\mathbf{B}\mathbf{x}^1 + (\mathbf{A})_{m,2}\mathbf{B}\mathbf{x}^2 + \cdots + (\mathbf{A})_{m,n}\mathbf{B}\mathbf{x}^n \end{bmatrix}.$$

(5)

with partitioned vector $\underline{X} = \begin{bmatrix} \underline{x}^1 \\ \underline{x}^2 \\ \vdots \\ \underline{x}^n \end{bmatrix}$, $\underline{x}^\ell \in \mathbb{R}^n$

Idea: precompute $B\underline{x}^\ell$, $\ell = 1, \dots, n$

$$Z = B \cdot \underbrace{[\underline{x}^1, \dots, \underline{x}^n]}_{\in \mathbb{R}^{n,n}} = [B\underline{x}^1, \dots, B\underline{x}^n]$$

↓ requires $O(n^3)$ operations

$$(A)_{1,1} B\underline{x}^1 + (A)_{1,2} B\underline{x}^2 + \dots + (A)_{1,n} B\underline{x}^n = Z\,(A)_{1,:}^T$$

→ Form: $Y = ZA^T$ → cost $O(n^3)$

Result vector $y = \begin{bmatrix} (Y)_{:,1} \\ \vdots \\ (Y)_{:,n} \end{bmatrix} \in \mathbb{R}^{n^2}$

Total cost: $O(n^3)$

Remark: Reshaping operations at no cost!
Just a reinterpretation of arrays

C++ code 1.4.3.9: Efficient multiplication of Kronecker product with vector in EIGEN
→ GITLAB

```
2  template <class Matrix, class Vector>
3  Vector kronmultv(const Matrix &A, const Matrix &B, const Vector &x){
4      unsigned int m = A.rows(); unsigned int n = A.cols();
5      unsigned int l = B.rows(); unsigned int k = B.cols();
6      // 1st matrix mult.  computes the products Bx^j
7      // 2nd matrix mult.  combines them linearly with the coefficients of
           A
8      Matrix t = B * Matrix::Map(x.data(),k,n) * A.transpose(); //
9      return Matrix::Map(t.data(), m*l, 1);
10 }
```

$O(n^3)$ $O(n^3)$

Review questions 1.4.3.11

NumCSE@ETHZ

A:

Explain why the classical concept of "computational effort" (= computational cost) is only loosely related to the runtime of a concrete implementation of an algorithm.

B:

We are given two dense matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n,p}$, $n, p \in \mathbb{N}$, $p < n$ fixed. What is the asmptotic complexity of each of the following two lines of code in terms of $n, p \to \infty$?

```
Eigen::MatrixXd AB = A*B.transpose();
y = AB.triangularView<Eigen::Upper>()*x;
```

C:

Given a vector $\mathbf{u} \in \mathbb{R}^n$, $n \in \mathbb{N}$, we consider the matrix

$$\mathbf{A} \in \mathbb{R}^{n,n}: \quad (\mathbf{A})_{i,j} = u_i + u_j + u_i u_j \,, \quad i,j \in \{1,\ldots,n\} \,.$$

Outline an efficient algorithm for computing the matrix-vector product $\mathbf{Ax}$, $\mathbf{x} \in \mathbb{R}^n$.

▷ Additional review questions can be found in the lecture document

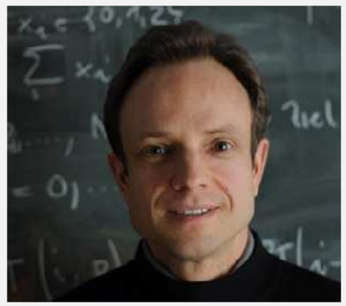
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 1.5: Machine Arithmetic and Consequences

Prof. R. Hiptmair, SAM, ETH Zurich

Date: August 24, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- Gram-Schmidt orthonormalization from linear algebra
- Concept of a relative error

Dependencies.

Duration: 30 minutes

I. Computing with Matrices and Vectors
1.5. Machine Arithmetic and Consequences
1.5.1. Experiment: Loss of Orthogonality

Recall: Gram-Schmidt orthonormalization

Input: $\{\mathbf{a}^1,\ldots,\mathbf{a}^k\} \subset \mathbb{K}^n$ $\mathbb{R}^n$

1: $\mathbf{q}^1 := \frac{\mathbf{a}^1}{\|\mathbf{a}^1\|_2}$ % 1st output vector
2: **for** $j = 2,\ldots,k$ **do**
{ % Orthogonal projection
3: $\mathbf{q}^j := \mathbf{a}^j$
4: **for** $\ell = 1,2,\ldots,j-1$ **do**
5: $\{\ \mathbf{q}^j \leftarrow \mathbf{q}^j - \langle \mathbf{a}^j, \mathbf{q}^\ell \rangle \mathbf{q}^\ell\ \}$
6: **if** $(\ \mathbf{q}^j = \mathbf{0}\ )$ **then** STOP
7: **else** $\{\ \mathbf{q}^j \leftarrow \frac{\mathbf{q}^j}{\|\mathbf{q}^j\|_2}\ \}$
8: }

Output: $\{\mathbf{q}^1,\ldots,\mathbf{q}^j\}$

$Q := [q^1,\ldots,q^j]$

If no STOP ($j=k$) then $Q^TQ = I$

$\Leftrightarrow \{q^j\}_{j=1}^k$ is orthonormal

(GS)

- $\mathrm{Span}\{q^1,\ldots,q^\ell\} = \mathrm{Span}\{a^1,\ldots,a^\ell\}$

(2)

C++ code 1.5.1.3: Gram-Schmidt orthogonalisation in EIGEN → GITLAB

```
2  template <class Matrix> Matrix gramschmidt(const Matrix &A) {
3    Matrix Q = A;
4    // First vector just gets normalized, Line 1 of (GS)
5    Q.col(0).normalize();
6    for (unsigned int j = 1; j < A.cols(); ++j) {
7      // Replace inner loop over each previous vector in Q with fast
8      // matrix-vector multiplication (Lines 4, 5 of (GS))
9      Q.col(j) -= Q.leftCols(j) *
10                 (Q.leftCols(j).adjoint() * A.col(j)); //
11     // Normalize vector, if possible.
12     // Otherwise colums of A must have been linearly dependent
13     if (Q.col(j).norm() <= 10e-9 * A.col(j).norm()) { //
14       std::cerr << "Gram-Schmidt failed: A has lin. dep columns." << std::endl;
15       break;
16     } else {
17       Q.col(j).normalize();
18     } // Line 7 of (GS)
19   }
20   return Q;
21 }
```

Run for $A = \left[\frac{1}{i+j-1}\right]_{i,j=1}^{n} \in \mathbb{R}^{n,n}$, $n = 10$

$\Rightarrow Q^T Q =$

```
I =
    1.0000    0.0000   -0.0000    0.0000   -0.0000    0.0000   -0.0000   -0.0000   -0.0000   -0.0000
    0.0000    1.0000   -0.0000    0.0000   -0.0000    0.0000   -0.0000   -0.0000   -0.0000   -0.0000
   -0.0000   -0.0000    1.0000    0.0000   -0.0000    0.0000   -0.0000   -0.0000   -0.0000   -0.0000
    0.0000    0.0000    0.0000    1.0000   -0.0000    0.0000   -0.0000   -0.0000   -0.0000   -0.0000
   -0.0000   -0.0000   -0.0000   -0.0000    1.0000    0.0000   -0.0008   -0.0007   -0.0007   -0.0006
    0.0000    0.0000    0.0000    0.0000    0.0000    1.0000   -0.0540   -0.0430   -0.0360   -0.0289
   -0.0000   -0.0000   -0.0000   -0.0000   -0.0008   -0.0540    1.0000    0.9999    0.9998    0.9996
   -0.0000   -0.0000   -0.0000   -0.0000   -0.0007   -0.0430    0.9999    1.0000    1.0000    0.9999
   -0.0000   -0.0000   -0.0000   -0.0000   -0.0007   -0.0360    0.9998    1.0000    1.0000    1.0000
   -0.0000   -0.0000   -0.0000   -0.0000   -0.0006   -0.0289    0.9996    0.9999    1.0000    1.0000
```

Complete loss of orthonormality

1.5.2. Machine numbers

Computers are "finite" $\Rightarrow$ they cannot compute in $\mathbb{R}$

$\Rightarrow$ Numerical LA $\neq$ LA !

Computers compute in $\mathbb{M}$

- finite → overflow/underflow
- discrete in $\mathbb{R}$

1.5.3. Roundoff Errors

op $\hat{=}$ elementary arithmetic operation $\in \{+, -, *, /\}$

$\text{op} : \mathbb{M} \times \mathbb{M} \not\to \mathbb{M}$

Hardware implementations replace op with $\widetilde{\text{op}}$

$\widetilde{\text{op}} := \text{rd} \circ \text{op}$

Definition 1.5.3.6. Correct rounding

Correct rounding ("rounding up") is given by the function

$$\text{rd} : \begin{cases} \mathbb{R} & \to & \mathbb{M} \\ x & \mapsto & \max \operatorname{argmin}_{\tilde{x} \in \mathbb{M}} |x - \tilde{x}| \,. \end{cases}$$

③

Good news: Relative error of $\widetilde{op}$ can be controlled

rel. err. $\dfrac{|\widetilde{op}(x,y) - op(x,y)|}{|op(x,y)|} = \dfrac{|(rd-Id)\,op(x,y)|}{|op(x,y)|} \approx$ relative error of rd

Guaranteed: machine precision (of $\mathbb{M}$)

$\max_{x \in \mathbb{M}} \dfrac{|rd(x) - x|}{|x|} =: EPS \approx 2.2 \cdot 10^{-16}$ for double

$\mathbb{M} = [\min\{|x|, x \in \mathbb{M}\}, \max\{|x| \; x \in \mathbb{M}\}] \subset \mathbb{R}$

$\Leftrightarrow$ $rd(x) = x(1+\varepsilon)$, with $|\varepsilon| \le EPS$

Assumption 1.5.3.11. "Axiom" of roundoff analysis

There is a small positive number EPS, the machine precision, such that for the elementary arithmetic operations $\star \in \{+,-,\cdot,/\}$ and "hard-wired" functions* $f \in \{\exp, \sin, \cos, \log, \ldots\}$ holds

$$x \tilde{\star} y = (x \star y)(1+\delta) \quad , \quad \tilde{f}(x) = f(x)(1+\delta) \quad \forall x, y \in \mathbb{M} \, ,$$

with $|\delta| < EPS$.

Remark: Impact of roundoff can be vastly different for different implementations

C++ code 1.5.1.7: Wrong result from Gram-Schmidt orthogonalisation EIGEN ⇢ GITLAB

```
2  void gsroundoff(MatrixXd& A){
3    // Gram-Schmidt orthogonalization of columns of A, see Code 1.5.1.3
4    MatrixXd Q = gramschmidt(A);
5    // Test orthonormality of columns of Q, which should be an
6    // orthogonal matrix according to theory
7    cout << setprecision(4) << fixed << "I = "
8         << endl << Q.transpose()*Q << endl;
9    // EIGEN's stable internal Gram-Schmidt orthogonalization by
10   // QR-decomposition, see Rem. 1.5.1.9 below
11   HouseholderQR<MatrixXd> qr(A.rows(),A.cols()); //
12   qr.compute(A); MatrixXd Q1 = qr.householderQ(); //
13   // Test orthonormality
14   cout << "I1 = " << endl << Q1.transpose()*Q1 << endl;
15   // Check orthonormality and span property (1.5.1.2)
16   MatrixXd R1 = qr.matrixQR().triangularView<Upper>();
17   cout << scientific << "A-Q1*R1 = " << endl << A-Q1*R1 << endl;
18 }
```

$Q^T Q =$

```
I1 =
 1.0000 -0.0000  0.0000 -0.0000 -0.0000 -0.0000 -0.0000 -0.0000 -0.0000  0.0000
-0.0000  1.0000 -0.0000  0.0000  0.0000 -0.0000 -0.0000 -0.0000  0.0000 -0.0000
 0.0000 -0.0000  1.0000 -0.0000 -0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
-0.0000  0.0000 -0.0000  1.0000  0.0000 -0.0000 -0.0000  0.0000  0.0000  0.0000
-0.0000  0.0000 -0.0000  0.0000  1.0000 -0.0000  0.0000 -0.0000 -0.0000  0.0000
-0.0000 -0.0000  0.0000 -0.0000 -0.0000  1.0000 -0.0000 -0.0000  0.0000 -0.0000
-0.0000 -0.0000  0.0000 -0.0000  0.0000 -0.0000  1.0000  0.0000  0.0000 -0.0000
-0.0000 -0.0000  0.0000  0.0000 -0.0000 -0.0000  0.0000  1.0000 -0.0000  0.0000
-0.0000  0.0000  0.0000  0.0000 -0.0000  0.0000  0.0000 -0.0000  1.0000 -0.0000
 0.0000 -0.0000  0.0000  0.0000  0.0000 -0.0000 -0.0000  0.0000 -0.0000  1.0000
```

(4)

Rem 1.5.3.15 : (Testing == 0.0)

Input: $\{\mathbf{a}^1,\ldots,\mathbf{a}^k\} \subset \mathbb{K}^n$

1: $\mathbf{q}^1 := \frac{\mathbf{a}^1}{\|\mathbf{a}^1\|_2}$ *% 1st output vector*
2: **for** $j = 2,\ldots,k$ **do**
{ *% Orthogonal projection*
3: $\mathbf{q}^j := \mathbf{a}^j$
4: **for** $\ell = 1,2,\ldots,j-1$ **do** (GS)
5: { $\mathbf{q}^j \leftarrow \mathbf{q}^j - \langle \mathbf{a}^j, \mathbf{q}^\ell \rangle \mathbf{q}^\ell$ }
6: **if** ($\mathbf{q}^j = \mathbf{0}$) **then** STOP
7: **else** { $\mathbf{q}^j \leftarrow \frac{\mathbf{q}^j}{\|\mathbf{q}^j\|_2}$ }
8: }

Output: $\{\mathbf{q}^1,\ldots,\mathbf{q}^j\}$

C++ code 1.5.1.3: Gram-Schmidt orthogonalisation in EIGEN → GITLAB

```
2  template <class Matrix> Matrix gramschmidt(const Matrix &A) {
3    Matrix Q = A;
4    // First vector just gets normalized, Line 1 of (GS)
5    Q.col(0).normalize();
6    for (unsigned int j = 1; j < A.cols(); ++j) {
7      // Replace inner loop over each previous vector in Q with fast
8      // matrix-vector multiplication (Lines 4, 5 of (GS))
9      Q.col(j) -= Q.leftCols(j) *
10                 (Q.leftCols(j).adjoint() * A.col(j)); //
11     // Normalize vector, if possible.
12     // Otherwise colums of A must have been linearly dependent
13     if (Q.col(j).norm() <= 10e-9 * A.col(j).norm()) { //
14       std::cerr << "Gram-Schmidt failed: A has lin. dep columns." << std::endl;
15       break;
16     } else {
17       Q.col(j).normalize();
18     } // Line 7 of (GS)
19   }
20   return Q;
21 }
```

▽ Replace == 0.0 with a test for relative smallness.

▷ Review questions 1.5.3.18

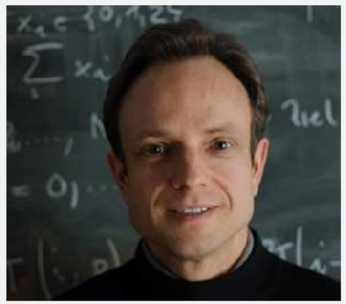
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 1.5.4: Cancellation

Prof. R. Hiptmair, SAM, ETH Zurich

Date: August 24, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.
- Taylor formula
- Elementary trigonometry

Dependencies. Section 1.5.3, Section 1.5.1

Duration: 45 minutes

I. Computing with Matrices and Vectors

1.5. Machine Arithmetic and Consequences

1.5.4. Cancellation

Why worry about EPS-sized errors?

▷ Because of error amplification

Ex.: 1.5.4.1. : Zeros of a quadratic polynomial

C++ code 1.5.4.3: Discriminant formula for the real roots of $p(\xi) = \xi^2 + \alpha\xi + \beta$ ↦ GITLAB

```
2  //!  C++ function computing the zeros of a quadratic polynomial
3  //!  ξ → ξ^2 + αξ + β by means of the familiar discriminant
4  //!  formula ξ_{1,2} = ½(−α ± √(α^2 − 4β)).  However
5  //!  this implementation is vulnerable to round-off!  The zeros are
6  //!  returned in a column vector
7  Vector2d zerosquadpol(double alpha, double beta) {
8    Vector2d z;
9    double D = std::pow(alpha, 2) - 4 * beta; // discriminant
10   if (D < 0)
11     throw "no real zeros";
12   else {
13     // The famous discriminant formula
14     double wD = std::sqrt(D);
15     z << (-alpha - wD) / 2, (-alpha + wD) / 2; //
16   }
17   return z;
18 }
```

$\gamma \gg 1 \Rightarrow \begin{matrix}\alpha \gg 1\\ \beta = 1\end{matrix}$

$\Rightarrow \sqrt{D} \approx \alpha$

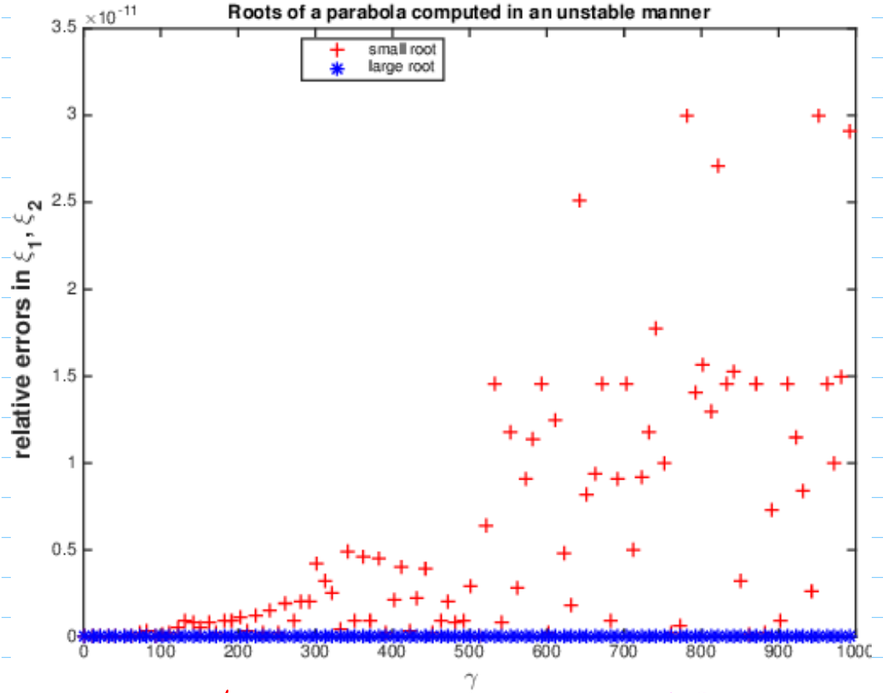
↑ large root

↑ cancellation here

Apply: $p(\xi) = \xi^2 - (\gamma + 1/\gamma)\xi + 1 = (\xi - \gamma)(\xi - 1/\gamma)$

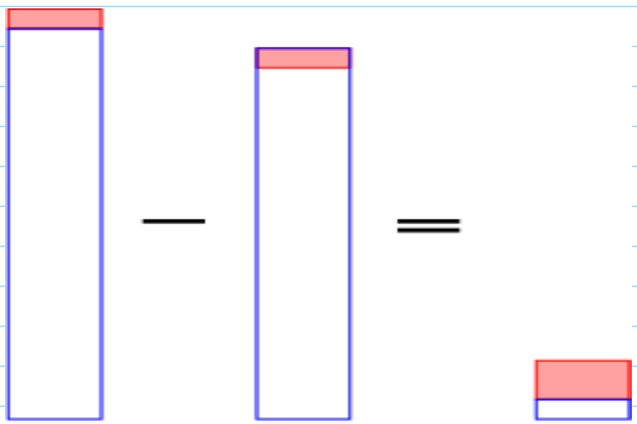
$= \alpha$ $\quad = :\beta$

(2)



⚠ Huge relative error for small root if $\gamma \gg 1$

Cancellation: Extreme amplification of relative errors during the subtraction of numbers of equal size



← huge relative error

$$\begin{aligned} x &= 0.123467* && \leftarrow \text{7th digit perturbed} \\ y &= 0.123456* && \leftarrow \text{7th digit perturbed} \\ \hline x-y &= 0.000011* = 0.11*000 \cdot 10^{-4} && \leftarrow \text{3rd digit perturbed} \end{aligned}$$

padded zeroes

Ex. 1.5.4.7: Cancellation in difference quotients

$f: \mathbb{R} \to \mathbb{R}$ smooth: $f'(x) \approx \dfrac{f(x+h)-f(x)}{h}$ for $h \ll 1$

$\Rightarrow$ Cancellation in numerator for $h \approx 0$

C++ code 1.5.4.10: Difference quotient approximation of the derivative of exp → GITLAB

```
2  //!  Difference quotient approximation
3  //!  of the derivative of exp
4  void diffq(){
5    double h = 0.1, x = 0.0;
6    for(int i = 1; i <= 16; ++i){
7      double df = (exp(x+h)-exp(x))/h;
8      cout << setprecision(14) << fixed;
9      cout  << setw(5) << -i
10       << setw(20) << df-1 << endl;
11     h /= 10;
12   }
13 }
```

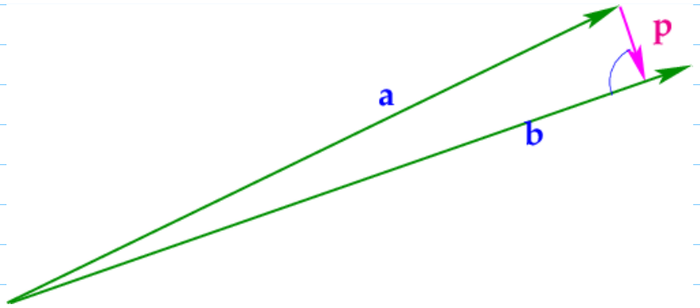
$f(x) = e^x$

$\log_{10}(h)$	relative error	
-1	0.05170918075648	Approaching the limit
-2	0.00501670841679	
-3	0.00050016670838	
-4	0.00005000166714	
-5	0.00000500000696	
-6	0.00000049996218	
-7	0.00000004943368	
-8	-0.00000000607747	Impact of cancellation
-9	0.00000008274037	
-10	0.00000008274037	
-11	0.00000008274037	
-12	0.00008890058234	
-13	-0.00079927783736	
-14	-0.00079927783736	
-15	0.11022302462516	
-16	-1.00000000000000	

Measured relative errors ▷

Minimal relative error for $h \approx \sqrt{EPS}$

Rem 1.5.4.12 (Cancellation & Orthogonalization)



◁ orth. projection

If a, b are nearly parallel

$$p = a - \frac{a^T b}{b^T b} \cdot b \approx 0$$

cancellation here

▷ The reason for failure of naive G.-S. implementation.

§ 1.5.4.16 Avoiding Cancellation

Ex. 1.5.4.17 : Stable discriminant formula

- $p(\xi) = \xi^2 + \alpha\xi + \beta$ w/ roots $\xi_1, \xi_2 \in \mathbb{R}$

Vieta's thm.: $\xi_1 \cdot \xi_2 = \beta$, $|\xi_2| > |\xi_1|$

Idea: · Compute large root first (no cancellation

· $\xi_1 = \beta / \xi_2$

C++ code 1.5.4.18: Stable computation of real root of a quadratic polynomial ➙ GITLAB

```
//!  C++ function computing the zeros of a quadratic polynomial
//!  ξ → ξ^2 + αξ + β by means of the familiar discriminant
//!  formula ξ_{1,2} = ½(−α ± √(α^2 − 4β)).
//!  This is a stable implementation based on Vieta's theorem.
//!  The zeros are returned in a column vector
VectorXd zerosquadpolstab(double alpha, double beta) {
  Vector2d z(2);
  double D = std::pow(alpha, 2) - 4 * beta; // discriminant
  if (D < 0)
    throw "no real zeros";
  else {
    double wD = std::sqrt(D);
    // Use discriminant formula only for zero far away from 0
    // in order to avoid cancellation.  For the other zero
    // use Vieta's formula.
    if (alpha >= 0) {
      double t = 0.5 * (-alpha - wD); //  --> no cancellation, α > 0
      z << t, beta / t;
    } else {
      double t = 0.5 * (-alpha + wD); //  --> no cancelation, α < 0
      z << beta / t, t;
    }
  }
  return z;
}
```

Ex. 1.5.4.19 : Recasting Expression

- $\int_0^x \sin t \, dt = 1 - \cos x = 2\sin^2(x/2)$

cancellation for $x \approx 0$ ↑ ↑ no cancellation

- $y = \sqrt{1+x^2} - \sqrt{1-x^2}$ ← cancellation for $x \approx 0$

$$= \frac{\left(\sqrt{1+x^2} - \sqrt{1-x^2}\right)\left(\sqrt{1+x^2} + \sqrt{1-x^2}\right)}{\left(\sqrt{1+x^2} + \sqrt{1-x^2}\right)} \quad [\, a^2 - b^2 = (a+b)(a-b) \,]$$

$$= \frac{2x^2}{\left(\sqrt{1+x^2} + \sqrt{1-x^2}\right)}$$ → no cancellation

Why is cancellation harmless for $x \approx 1$?

Analytic manipulations offer ample opportunity to rewrite expressions in equivalent form immune to cancellation.

Ex 1.5.4.26 Trading cancellation for approximation

$$I(a) = \int_0^1 e^{at}\,dt = \frac{e^a - 1}{a} \quad \leftarrow \text{cancellation for } a \approx 0$$

Idea: Approximation by Taylor polynomial for $a \approx 0$

$f: \mathbb{R} \to \mathbb{R}$ smooth

$$f(x_0+h) = \sum_{k=0}^{m} \frac{f^{(k)}(x_0)}{k!} h^k + \underbrace{\frac{1}{(m+1)!} f^{(m+1)}(\xi) h^{m+1}}_{\text{remainder term}}, \quad \xi \in]x_0, x_0+h[$$

Apply with $f(x) = e^x$, $x_0 = 0$, $h = a$

$f^{(k)}(x) = e^x$

NumCSE@ETHZ

$$I(a) = \frac{\exp(a)-1}{a} = \underbrace{\sum_{k=0}^{m} \frac{1}{(k+1)!} a^k}_{\widetilde{I}_m(a)} + R_m(a), \quad R_m(a) = \frac{1}{(m+1)!}\exp(\xi)a^m \text{ for some } 0 \le \xi \le a.$$

$\widetilde{I}_m(a)$: an approximation of $I(a)$

How to choose m?

Goal: Relative truncation error $\le$ tol

$$\text{rel. err.} = \frac{|I(a) - \widetilde{I}_m(a)|}{|I(a)|} = \frac{\overbrace{(e^a-1)/a - \sum_{k=0}^{m} \frac{1}{(k+1)!}a^k}^{= R_m(a)}}{(e^a-1)/a}$$

$$\le \frac{1}{(m+1)!}\exp(\xi)a^m \le \frac{1}{(m+1)!}\exp(a)a^m \overset{!}{\le} \text{tol}$$

For $a = 10^{-3}$:

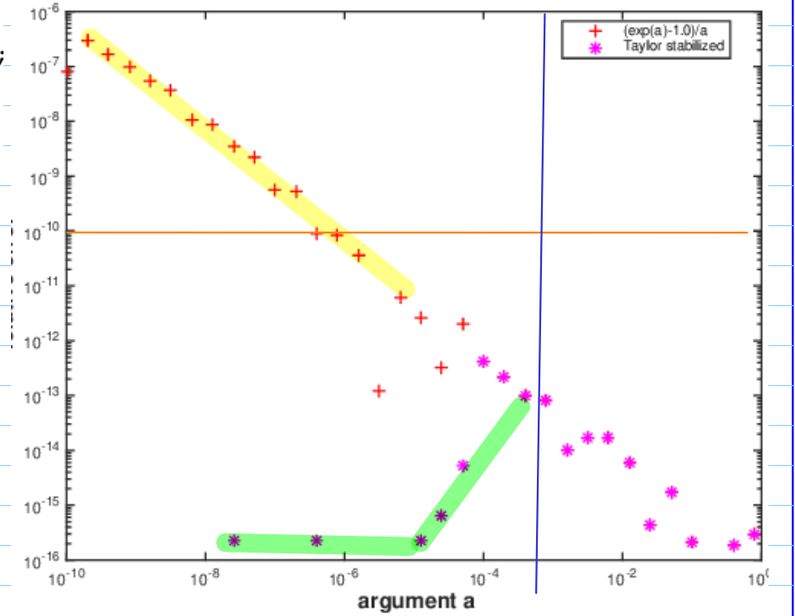
m	1	2	3	4	5
rel. err.	1.0010e-03	5.0050e-07	1.6683e-10	4.1708e-14	8.3417e-18

Note: $\log_{10}$ rel. err $\approx$ no. of decimal digits

(5)

```
if (abs(a) < 1E-3)
  v = 1.0 + (1.0/2 + 1.0/6*a)*a;
else
  v = (exp(a)-1.0)/a;
end
```

rel. err.



Review questions 1.5.4.33

A:

Give an expression for

$$I(a,b) := \int_a^b 1/x \, \mathrm{d}x$$

that allows cancellation-free evaluation for integration bounds $1 \ll a \approx b$.

B:

For integration bounds $a, b > 1$, $a \approx b$, propose a numerically sound way of computing

$$I(a,b) := \int_a^b \frac{1}{1+x^2} \, \mathrm{d}x \, .$$

C:

What is the problem with the C++ expression

```
y = std::log(std: cosh(x));
```

y where `x` is of type **double**? Rewrite this line of code into an algebraically equivalent one so that that problem does no longer occur.

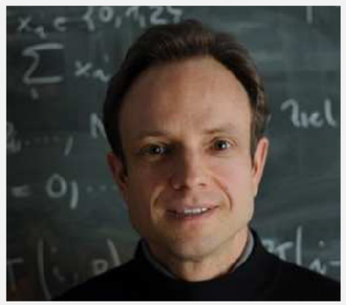
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 1.5.5: Numerical Stability

Prof. R. Hiptmair, SAM, ETH Zurich

Date: August 30, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- Vector and matrix norms

Dependencies.

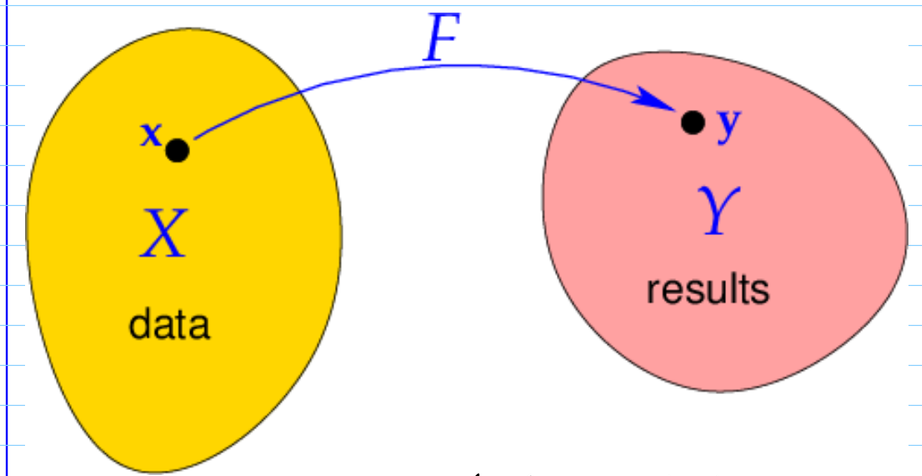
Duration: 35 minutes

I. Computing with Matrices and Vectors

1.5. Machine Arithmetic and Consequences

1.5.5. Numerical Stability (of algorithms)

Problem $\hat{=}$ a function/mapping $F : X \longrightarrow Y$



$X \hat{=}$ data space (input)

$Y \hat{=}$ result space (output)

↓

usually normed vector space

Ex 1.5.5.2: Matrix × Vector multiplication

$F : (A, \underline{x}) \in X \longmapsto A\underline{x} \in Y$

$X = \mathbb{R}^{m,n} \times \mathbb{R}^n \qquad Y = \mathbb{R}^m$

On $\mathbb{R}^n$ we use vector norms, e.g.

$\|\cdot\|_2 \hat{=}$ Euclidean norm, $\|\cdot\|_1$, $\|\cdot\|_\infty$

Matrix norms induced by vector norms :

Definition 1.5.5.10. Matrix norm

Given vector norms $\|\cdot\|_x$ and $\|\cdot\|_y$ on $\mathbb{K}^n$ and $\mathbb{K}^m$, respectively, the associated matrix norm is defined by

$$\mathbf{M} \in \mathbb{R}^{m,n}: \quad \|\mathbf{M}\| := \sup_{\mathbf{x}\in\mathbb{R}^n\setminus\{0\}} \frac{\|\mathbf{M}\mathbf{x}\|_y}{\|\mathbf{x}\|_x} .$$

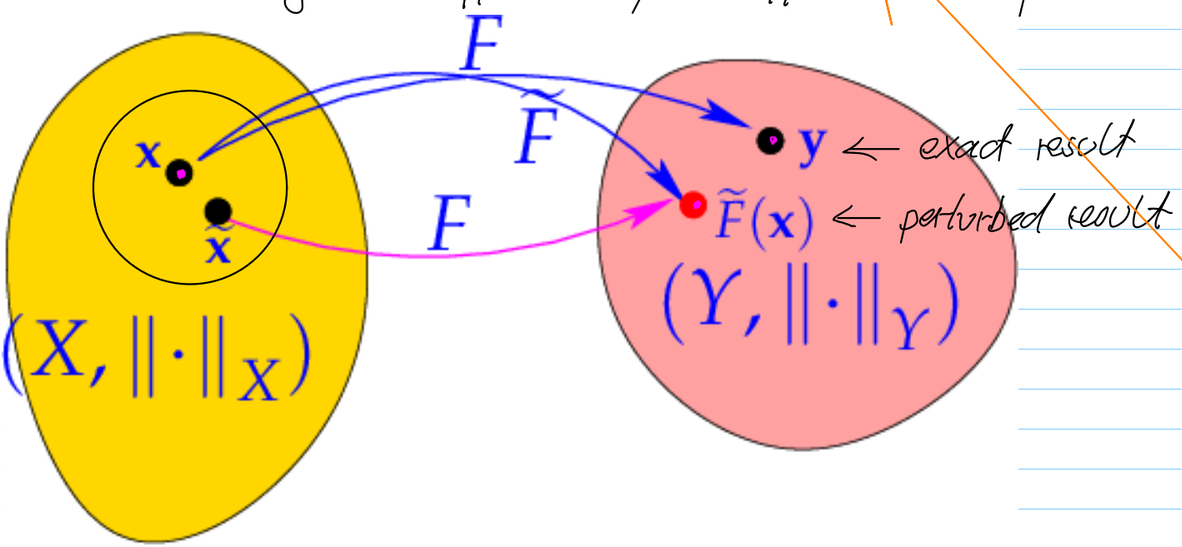
Definition 1.5.5.19. Stable algorithm

An algorithm $\tilde{F}$ for solving a problem $F : X \mapsto Y$ is numerically stable if for all $\mathbf{x} \in X$ its result $\tilde{F}(\mathbf{x})$ (possibly affected by roundoff) is the exact result for "slightly perturbed" data:

$$\exists C \approx 1: \quad \forall \mathbf{x} \in X: \quad \exists \tilde{\mathbf{x}} \in X: \quad \|\mathbf{x} - \tilde{\mathbf{x}}\|_X \le C w(\mathbf{x}) \,\mathrm{EPS}\, \|\mathbf{x}\|_X \;\wedge\; \tilde{F}(\mathbf{x}) = F(\tilde{\mathbf{x}}) .$$

$\tilde{F}$ ≙ algorithm affected by roundoff

#ops.



" Exact data hardly ever available " : input errors

Sloppily speaking, the impact of roundoff (*) on a *stable algorithm* is of the same order of magnitude as the effect of the inevitable perturbations due to rounding the input data.

Ex. 1.5.5.20 :

Problem : $(A, x) \in \mathbb{R}^{m,n} \times \mathbb{R}^n \longmapsto Ax \in \mathbb{R}^m$

$y = \tilde{F}((A,x))$: computed result

[Euclidean vector & matrix norms]

Which perturbations $(\hat{A}, \hat{x})$ of data (A, x) yield

$$y = \hat{A}\hat{x}$$

Possible choice: $\hat{x} = x$

$$\hat{A} = A + zx^T, \quad z := \frac{y - Ax}{\|x\|_2^2} \in \mathbb{R}^m$$

$$\Rightarrow \quad \hat{A}\hat{x} = Ax + \left(\frac{y - Ax}{\|x\|_2^2}\right) x^T x = Ax + y - Ax = y$$

$$F((\hat{A}, \hat{x})) = y$$

$$\|A - \hat{A}\|_2 = \|zx^T\|_2 \le \|x\|_2 \|z\|_2 = \frac{\|y - Ax\|_2}{\|x\|_2}$$

If $\|y - Ax\|_2 \le C mn \cdot \mathrm{EPS}\, \|x\|_2 \|A\|_2$ (*)

$\triangleright \; \|A - \hat{A}\|_2 \le C mn \cdot \mathrm{EPS}\, \|A\|_2$

If (*) holds $\forall$ inputs $(A,x) \Rightarrow \widetilde{F}$ is stable

Rem 1.5.5.22 :

Problem with sensitive dependence on data :

$\|F(x) - F(\hat{x})\|_y$ can be large for tiny $\|x - \hat{x}\|_x$

Example :

Tiny perturbations in initial positions & velocities (=data) will lead to huge difference in positions after a long time (= output)

Random output is a stable algorithm !

▷ Review questions 1.5.5.23

NumCSE@ETHZ

ETH Lecture 401-0663-00L Numerical Methods for CSE

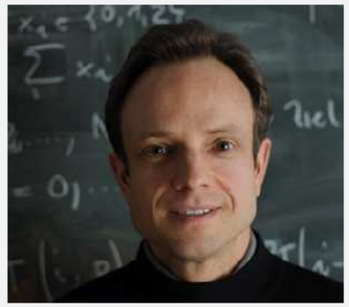
Course Video

Section 2.1–Section 2.2.1: Introduction and Theory: Linear Systems of Equations (LSEs)

Prof. R. Hiptmair, SAM, ETH Zurich

Date: September 18, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- Matric vector calculus
- Notions and facts about linear systems of equations as they were taught in the introductory linear algebra class.

Duration: minutes

II. Direct Methods for Square Linear Systems of Equations

2.1. Introduction: Linear Systems of Equations (LSE)

LSE: $A\underline{x} = \underline{b}$ ← right-hand-side vector $\in \mathbb{K}^n$

$A \in \mathbb{K}^{n,n}$: square system/ coefficient matrix

$\underline{x} \in \mathbb{K}^n$: vector of unknowns

#equation = #unknown = n

↳ $(A)_{i,:}^T \underline{x} = (\underline{b})_i$, $i = 1, \ldots, n$

[A scalar equation for each row of A]

Data/Input: $A, \underline{b}$

Sought/output: $\underline{x}$

NumCSE@ETHZ

2.2. Square LSE : Theory → Linear algebra

2.2.1. Existence and Uniqueness of Solutions

Definition 2.2.1.1. Invertible matrix →

$\mathbf{A} \in \mathbb{K}^{n,n}$ invertible/regular $:\Leftrightarrow \exists_1 \mathbf{B} \in \mathbb{K}^{n,n}: \mathbf{AB} = \mathbf{BA} = \mathbf{I}$.

$\mathbf{B}$ is called the **inverse** of $\mathbf{A}$, (✎ notation $\mathbf{B} = \mathbf{A}^{-1}$)

A invertible : $A\underline{x} = \underline{b} \iff \underline{x} = A^{-1}\underline{b}$ unique

▽ Do not use matrix inversion function to solve LSE with numerical libraries

Criteria:

$A \in \mathbb{K}^{n,n}$ regular $\Leftrightarrow$ $\det A \neq 0$

$\Leftrightarrow$ cols lin. indep.

$\Leftrightarrow$ rows lin. indep.

$\Leftrightarrow$ $\mathcal{N}(A) = \{\underline{z} \in \mathbb{K}^n : A\underline{z} = \underline{0}\} = \{\underline{0}\}$

↑ nullspace of A

ETH Lecture 401-0663-00L Numerical Methods for CSE

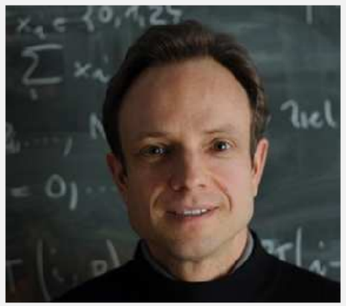
Course Video

Ex. 2.1.0.3: Nodal Analysis of Linear Electric Circuits

Prof. R. Hiptmair, SAM, ETH Zurich

Date: September 18, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- Complex numbers
- Elementary physics: current and voltage

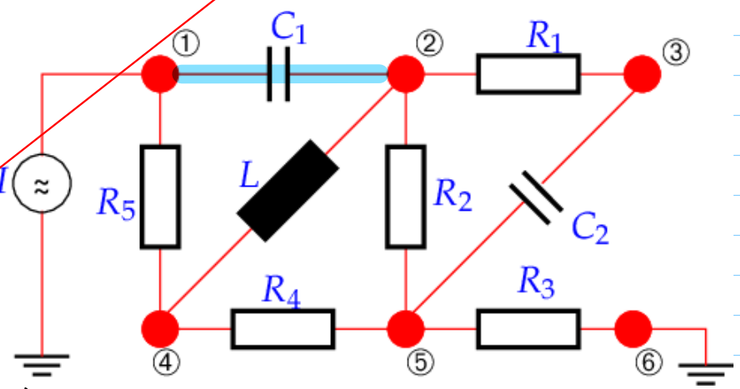
Dependencies.

Duration: 15 minutes

II. Direct Methods for Square Linear Systems of Equations

2.1. Introduction: Linear Systems of Equations (LSE)

Ex 2.1.0.3. Nodal Analysis of Linear Electric Circuits [in frequency domain]



- $\hat{=}$ nodes, numbered $1, \ldots, n$
- $\hat{=}$ branch/edge (directed)

▷ branch currents I_{kj} (directed)

$I_{kj} \in \mathbb{C}$ (complex)

Kirchhoff current law: $\sum_{j=1}^{n} I_{kj} = 0$ (KCL)

Nodal potentials $U_k \in \mathbb{C}$, $k = 1, \ldots, n$

↳ Unknowns

(2)

Circuit element ⟶ branch current-voltage relationship

[branch voltage : $U_k - U_j$]

- Ohmic resistor: $I = \frac{U}{R}$, $[R] = 1\mathrm{VA}^{-1}$
- capacitor: $I = \imath\omega C U$, capacitance $[C] = 1\mathrm{AsV}^{-1}$
- coil/inductor : $I = \frac{U}{\imath\omega L}$, inductance $[L] = 1\mathrm{VsA}^{-1}$

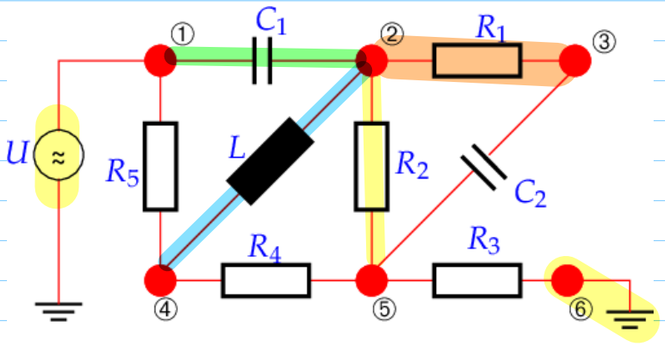
$$\blacktriangleright \quad I_{kj} = \begin{cases} R^{-1}(U_k - U_j)\,, \\ \imath\omega C(U_k - U_j)\,, \\ -\imath\omega^{-1}L^{-1}(U_k - U_j)\,. \end{cases}$$

branch current ↑ $\frac{U}{\imath\omega L}$ ← branch voltage

$$I = \alpha U \quad , \alpha \in \mathbb{C}$$

with KCL : $\sum_{j=1}^{n} \alpha_{kj}(U_k - U_j) = 0 \quad ; k = 1, \dots, n$

$\hat{=}$ LSE



Voltage source

⟶ U_1 known

$U_6 = 0$

⇒ Can drop the KCL equ. for $k = 1, 6$

$$\begin{aligned}
② &: \imath\omega C_1(U_2 - U_1) + R_1^{-1}(U_2 - U_3) - \imath\omega^{-1}L^{-1}(U_2 - U_4) + R_2^{-1}(U_2 - U_5) = 0\,, \\
③ &: R_1^{-1}(U_3 - U_2) + \imath\omega C_2(U_3 - U_5) = 0\,, \\
④ &: R_5^{-1}(U_4 - U_1) - \imath\omega^{-1}L^{-1}(U_4 - U_2) + R_4^{-1}(U_4 - U_5) = 0\,, \\
⑤ &: R_2^{-1}(U_5 - U_2) + \imath\omega C_2(U_5 - U_3) + R_4^{-1}(U_5 - U_4) + R_3^{-1}(U_5 - U_6) = 0\,, \\
&U_1 = U \quad , \quad U_6 = 0\,.
\end{aligned}$$

known ⟶ r.h.s vector

$$\begin{pmatrix} \imath\omega C_1 + \frac{1}{R_1} - \frac{i}{\omega L} + \frac{1}{R_2} & -\frac{1}{R_1} & \frac{i}{\omega L} & -\frac{1}{R_2} \\ -\frac{1}{R_1} & \frac{1}{R_1} + \imath\omega C_2 & 0 & -\imath\omega C_2 \\ \frac{i}{\omega L} & 0 & \frac{1}{R_5} - \frac{i}{\omega L} + \frac{1}{R_4} & -\frac{1}{R_4} \\ -\frac{1}{R_2} & -\imath\omega C_2 & -\frac{1}{R_4} & \frac{1}{R_2} + \imath\omega C_2 + \frac{1}{R_4} + R_3^{-1} \end{pmatrix} \begin{pmatrix} U_2 \\ U_3 \\ U_4 \\ U_5 \end{pmatrix} = \begin{pmatrix} \imath\omega C_1 U \\ 0 \\ \frac{1}{R_5} U \\ 0 \end{pmatrix}$$

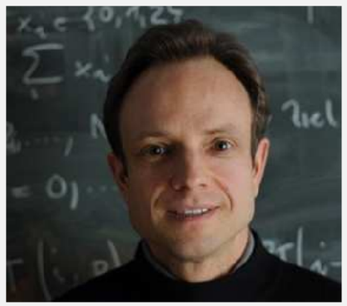
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 2.2.2: Sensitivity of Linear Systems

Prof. R. Hiptmair, SAM, ETH Zurich

Date: September 18, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- Ability to compute with matrix and vector norms

Dependencies. Section 1.5.5

Duration: 30 minutes

II. Direct Methods for Square Linear Systems of Equations

2.2. Theory: Linear Systems of Equations

2.2.2. Sensitivity/Conditioning of LSE

↓

quantifies how small (relative) perturbation of data lead to changes of the output

Needed: vector norms & (induced) matrix norms $\|\cdot\|$

$$\|M\| = \sup_{x \neq 0} \frac{\|Mx\|}{\|x\|} \quad \triangleright \quad \|Mx\| \leq \|M\|\|x\|$$

$$\|x+y\| \leq \|x\| + \|y\|$$

$$\Rightarrow \quad \|x-y\| \geq \|x\| - \|y\|$$

Ex.: 2.2.1 (Sensitivity of linear mappings)

$A \in \mathbb{R}^{n,n}$ fixed & regular: input $x \longmapsto y := Ax$ ≙ output

(2)

$$A\underline{x} = \underline{y} \Leftrightarrow \underline{x} = A^{-1}\underline{y} \Rightarrow \|\underline{x}\| \le \|A^{-1}\|\,\|\underline{y}\|$$

$$A(\underline{x}+\Delta\underline{x}) = \underline{y}+\Delta\underline{y} \Rightarrow \|\Delta\underline{y}\| \le \|A\|\,\|\Delta\underline{x}\|$$

rel. pert. $\dfrac{\|\Delta\underline{y}\|}{\|\underline{y}\|} \le \dfrac{\|A\|\,\|\Delta\underline{x}\|}{\|A^{-1}\|^{-1}\|\underline{x}\|} \le \|A\|\|A^{-1}\| \dfrac{\|\Delta\underline{x}\|}{\|\underline{x}\|}$

$\|A\|\|A^{-1}\|$ ↑ sensitivity

▽ Sensitivity heavily depends on choice of norms

LSE:

$A \in \mathbb{R}^{n,n}$ regular : $A\underline{x} = \underline{b}$

$$\frac{(A+\Delta A)(\underline{x}+\Delta\underline{x}) = \underline{b}+\Delta\underline{b}}{(A+\Delta A)\Delta\underline{x} = \Delta\underline{b} - \Delta A\cdot\underline{x}} \Rightarrow$$

Tool:

Lemma 2.2.2.5. Perturbation lemma →

$\mathbf{B} \in \mathbb{R}^{n,n}$, $\|\mathbf{B}\| < 1 \Rightarrow \mathbf{I}+\mathbf{B}$ *regular* $\wedge$ $\left\|(\mathbf{I}+\mathbf{B})^{-1}\right\| \le \dfrac{1}{1-\|\mathbf{B}\|}$.

Proof: Auxiliary estimate, $x \in \mathbb{R}^n$

$$\|(I+B)\underline{x}\| \ge \|x\| - \|Bx\| \ge \overbrace{(1-\|B\|)}^{>0}\|x\|$$

- $\underline{x} \neq 0 \Rightarrow (I+B)\underline{x} \neq 0 \rightsquigarrow \mathcal{N}(I+B) = \{0\}$

- $\|(I+B)^{-1}\| = \sup_{x\neq 0} \dfrac{\|(I+B)^{-1}x\|}{\|x\|} = \sup_{\substack{y\in\mathbb{R}^n \\ y\neq 0}} \dfrac{\|y\|}{\|(I+B)y\|}$ (y = $(I+B)^{-1}x$)

$$\le \sup_{y\neq 0} \frac{\|y\|}{(1-\|B\|)\|y\|}$$

□

Apply to $A+\Delta A = A(I + A^{-1}\Delta A)$

$$\|(A+\Delta A)^{-1}\| = \|(I+A^{-1}\Delta A)^{-1}A^{-1}\|$$

[Lemma 2.2.2.5] $\le \|A^{-1}\| \cdot \dfrac{1}{1-\|A^{-1}\|\|\Delta A\|}$

$$\|\Delta\underline{x}\| \le \frac{\|A^{-1}\|}{1-\|A^{-1}\|\|\Delta A\|}\cdot(\|\Delta\underline{b}\| + \|\Delta A\|\,\|\underline{x}\|)$$

$$\|\Delta\underline{x}\| \le \frac{\|A^{-1}\|\,\|A\|}{1-\|A^{-1}\|\|\Delta A\|}\left(\frac{\|\Delta\underline{b}\|}{\|A\|\|\underline{x}\|} + \frac{\|\Delta A\|}{\|A\|}\right)\|\underline{x}\|$$

Theorem 2.2.2.4. Conditioning of LSEs →,

If $\mathbf{A}$ *regular,* $\|\Delta\mathbf{A}\| < \|\mathbf{A}^{-1}\|^{-1}$ *and* (2.2.2.3), *then*

(i) $\mathbf{A}+\Delta\mathbf{A}$ *is regular/invertible,*

(ii)

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \le \frac{\|\mathbf{A}^{-1}\|\|\mathbf{A}\|}{1-\|\mathbf{A}^{-1}\|\|\mathbf{A}\|\|\Delta\mathbf{A}\|/\|\mathbf{A}\|}\left(\frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|} + \frac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|}\right).$$

relative error of ~~data~~ result — *relative perturbations*

Definition 2.2.2.7. Condition (number) of a matrix

Condition (number) of a matrix $\mathbf{A} \in \mathbb{R}^{n,n}$: $\quad \operatorname{cond}(\mathbf{A}) := \left\|\mathbf{A}^{-1}\right\| \|\mathbf{A}\|$

$$\begin{array}{c}\mathbf{Ax} = \mathbf{b} \\ (\mathbf{A}+\Delta\mathbf{A})\tilde{\mathbf{x}} = \mathbf{b}+\Delta\mathbf{b}\end{array} \Rightarrow \frac{\|\mathbf{x}-\tilde{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \frac{\operatorname{cond}(\mathbf{A})}{1-\operatorname{cond}(\mathbf{A})\|\Delta\mathbf{A}\|/\|\mathbf{A}\|}\left(\frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|}+\frac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|}\right). \quad (2.2.2.9)$$

$\operatorname{cond}(A) \approx 1$: LSE well-conditioned

$\operatorname{cond}(A) \gg 1$: LSE ill-conditioned

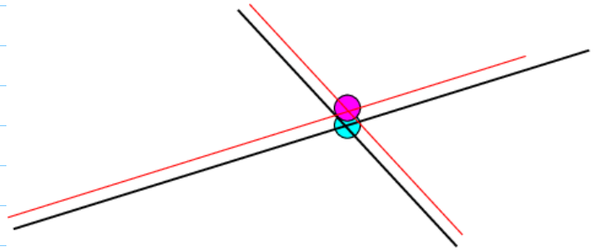
Ex 2.2.2.10 : Visualizing sensitivity : Intersection of lines in 2D

Hesse normal norms

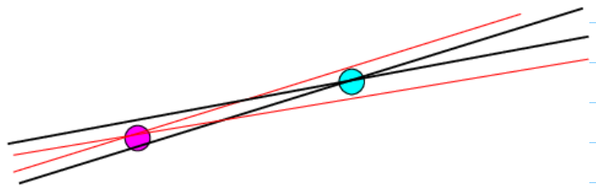
$$L_i = \{\mathbf{x} \in \mathbb{R}^2 : \mathbf{x}^T\mathbf{n}_i = d_i\}, \quad \mathbf{n}_i \in \mathbb{R}^2, d_i \in \mathbb{R}, \quad i = 1,2.$$

LSE for finding intersection: $\underbrace{\begin{bmatrix}\mathbf{n}_1^T \\ \mathbf{n}_2^T\end{bmatrix}}_{=:\mathbf{A}} \mathbf{x} = \underbrace{\begin{bmatrix}d_1 \\ d_2\end{bmatrix}}_{=:\mathbf{b}}$,

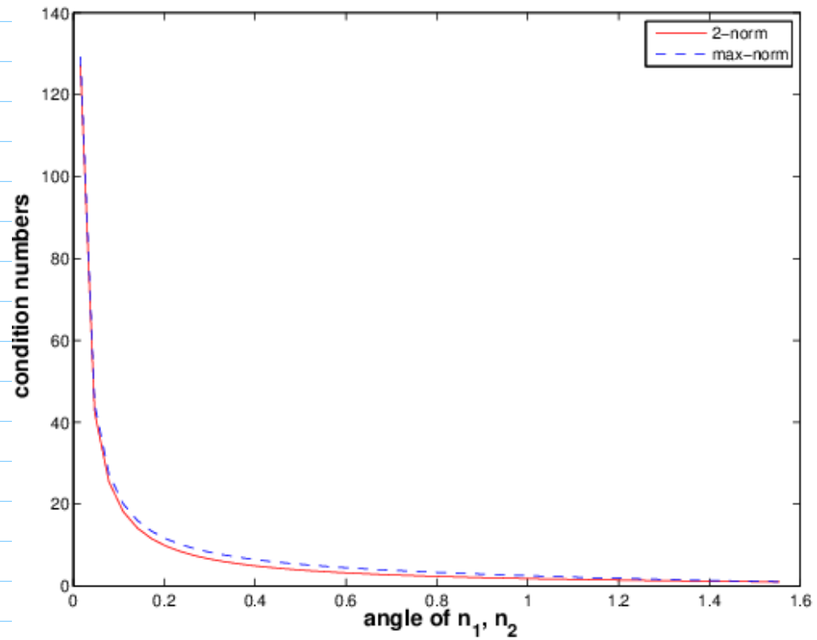
coordinates of $L_1 \cap L_2$



nearly orthogonal intersection: well-conditioned



glancing intersection: ill-conditioned



n_1 "almost linearly dependent" of n_2

$\rightarrow \operatorname{cond}(A) \rightarrow \infty$

Heuristics for predicting large $\operatorname{cond}(\mathbf{A})$

$\operatorname{cond}(\mathbf{A}) \gg 1 \quad \leftrightarrow \quad$ columns/rows of $\mathbf{A}$ "almost linearly dependent"

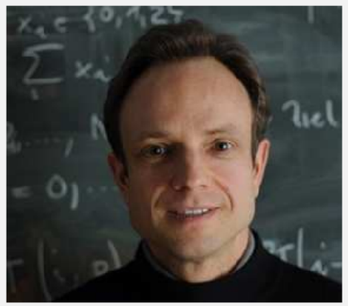
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 2.3–Section 2.5: Gaussian Elimination

Prof. R. Hiptmair, SAM, ETH Zurich

Date: September 18, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- Familiarity with Gaussian elimination as introduced in the linear algebra course.
- Ability to read C++ codes

Dependencies. Section 1.4

Duration: 30 minutes

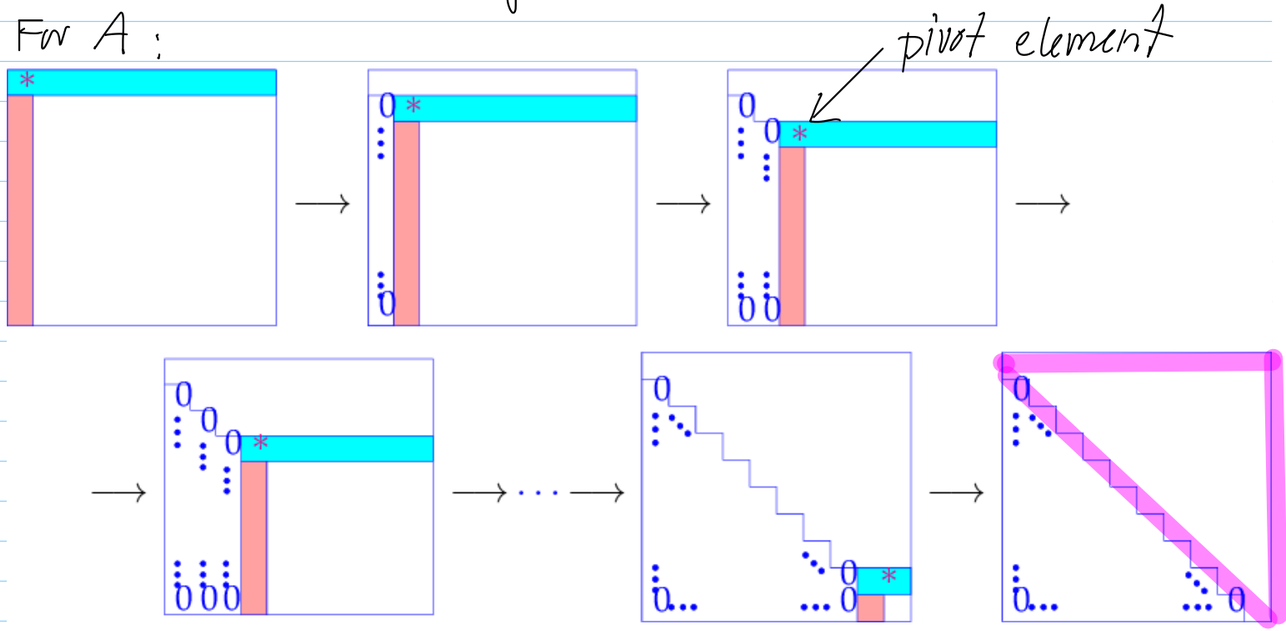
II. Direct Methods for Square Linear Systems of Equations

2.3. Gaussian Elimination (GE)

2.3.1. Basic Algorithm (→ LA)

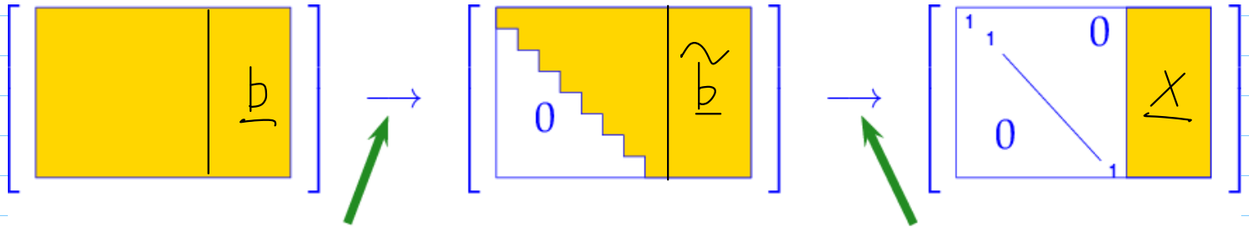
Successive row ops of LSE $A\underline{x} = \underline{b}$ to convert it to triangular form

For A:



(2)

For $A\underline{x} = \underline{b}$, $A \in \mathbb{R}^{n,n}$



forward elimination → $O(n^3)$

back substitution (row mfs.) → $O(n^2)$

§ 2.3.1.15

Rem 2.3.1.14 : Block GE

LSE: $A\underline{x} = \underline{b}$, $A \in \mathbb{R}^{n,n}$, $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ (k rows), $A_{11} \in \mathbb{R}^{k,k}$ (k columns)

Assume: A_{11} is regular

Recall: blockwise $M \times M$:

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{11}\mathbf{B}_{11} + \mathbf{A}_{12}\mathbf{B}_{21} & \mathbf{A}_{11}\mathbf{B}_{12} + \mathbf{A}_{12}\mathbf{B}_{22} \\ \mathbf{A}_{21}\mathbf{B}_{11} + \mathbf{A}_{22}\mathbf{B}_{21} & \mathbf{A}_{21}\mathbf{B}_{12} + \mathbf{A}_{22}\mathbf{B}_{22} \end{bmatrix}. \quad (1.3.1.14)$$

Now: GE like for 2×2 LSE

[except for commutativity]

$$\left[\begin{array}{cc|c} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{b}_1 \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{b}_2 \end{array}\right] \xrightarrow{\text{F.E.}\ \textcircled{1}} \left[\begin{array}{cc|c} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{b}_1 \\ 0 & \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12} & \mathbf{b}_2 - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{b}_1 \end{array}\right] \xrightarrow{\text{B.S.}\ \textcircled{2}} \left[\begin{array}{cc|c} \mathbf{I} & 0 & \mathbf{A}_{11}^{-1}(\mathbf{b}_1 - \mathbf{A}_{12}\mathbf{S}^{-1}\mathbf{b}_S) \\ 0 & \mathbf{I} & \mathbf{S}^{-1}\mathbf{b}_S \end{array}\right];;$$

$$\underline{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^n$$

$S := A_{22} - A_{21}A_{11}^{-1}A_{12}$: Schur complement

Rem 2.3.2.19 : Block LU-dec.

With

$$\underbrace{\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & 0 \\ \mathbf{A}_{21}\mathbf{A}_{11}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ 0 & \mathbf{S} \end{bmatrix}}_{\text{block LU-factorization}},$$

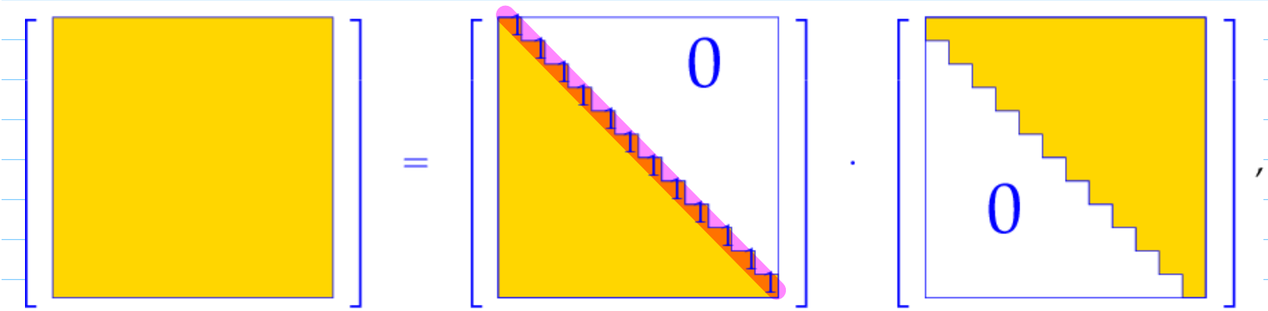
with Schur complement

$$\mathbf{S} := \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}. \quad (2.3.2.20)$$

③

2.3.2. LU-Decomposition

[equivalent to GE ⇒ cost(LU) = $O(n^3)$]



$\mathbf{A}$ = $\mathbf{L}$ · $\mathbf{U}$,

(possibly permuted) = normalized lower triangular · upper triangular

Definition 2.3.2.3. LU-decomposition/LU-factorization

Given a square matrix $\mathbf{A} \in \mathbb{K}^{n,n}$, an *upper triangular matrix* $\mathbf{U} \in \mathbb{K}^{n,n}$ and a *normalized lower triangular matrix* (→ Def. 1.1.2.3) form an LU-decomposition/LU-factorization of $\mathbf{A}$, if $\mathbf{A} = \mathbf{LU}$.

§2.3.2.15 : Solving LSE via LU

LSE: $A\underline{x} = \underline{b} \Leftrightarrow L(U\underline{x}) = \underline{b} \Rightarrow$ $L\underline{z} = \underline{b}$, $U\underline{x} = \underline{z}$

$\mathbf{Ax} = \mathbf{b}$:

① LU-decomposition $\mathbf{A} = \mathbf{LU}$, #elementary operations $= \frac{1}{3}n(n-1)(n+1)$ → $O(n^3)$

② forward substitution, solve $\mathbf{Lz} = \mathbf{b}$, #elementary operations $= \frac{1}{2}n(n-1)$

③ backward substitution, solve $\mathbf{Ux} = \mathbf{z}$, #elementary operations $= \frac{1}{2}n(n+1)$ } → $O(n^2)$

↳ Three-stage stage splitting of GE

setup phase (factorization) Cost: $O(n^3)$	+	**elimination phase** (forward/backward substition) Cost: $O(n^2)$

2.5.0.4. GE/LU-dec. in Eigen

$A \in \mathbb{R}^{n,n}$ ⟷ (square) matrix object

LSE multiple r.h.s.: $AX = B \in \mathbb{R}^{n,\ell}$

↳ contained in columns of B

linear algebra	EIGEN
$\mathbf{X} = \mathbf{A}^{-1}\mathbf{B} = \left[\mathbf{A}^{-1}(\mathbf{B})_{:,1}, \ldots, \mathbf{A}^{-1}(\mathbf{B})_{:,\ell}\right]$	`X = A.lu().solve(B)`
	① ② ③
Cost:	$O(n^3)$ $O(n^2\ell)$

Never contemplate implementing a general solver for linear systems of equations!

If possible, use algorithms from numerical libraries! (→ Exp. 2.3.1.7)

(4) Rem 2.5.0.10 : Many sequential solves of LSE

C++ code 2.5.0.11: Wasteful approach!
➙ GITLAB

```
// Setting:  N ≫ 1,
// large matrix A ∈ K^{n,n}
for(int j = 0; j < N; ++j){
    x = A.lu().solve(b);
    b = some_function(x);
}
```

computational effort $O(Nn^3)$

C++ code 2.5.0.12: Smart approach!
➙ GITLAB

```
// Setting:  N ≫ 1,
// large matrix A ∈ K^{n,n}
auto A_lu_dec = A.lu();
for(int j = 0; j < N; ++j){
    x = A_lu_dec.solve(b);
    b = some_function(x);
}
```

computational effort $O(n^3 + Nn^2)$

setup phase
(factorization)
Cost: $O(n^3)$

+

elimination phase
(forward/backward substition)
Cost: $O(n^2)$

→ A rationale for LU

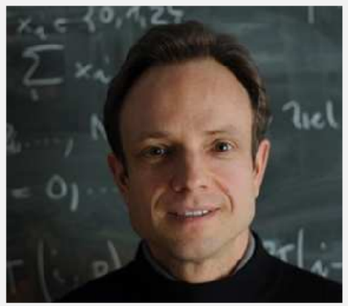
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 2.6: Exploiting Structure when Solving Linear Systems

Prof. R. Hiptmair, SAM, ETH Zurich

Date: September 18, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.
- Proficiency in matrix-vector calculus

Dependencies. Section 2.3, Section 1.4

Duration: 35 minutes

II. Direct Methods for Square Linear Systems of Equations

2.6. Exploiting Structure when Solving Linear Systems

- We have see triangular LSE

Recall: Block GE

regular

$$\begin{bmatrix}\mathbf{A}_{11}\mathbf{A}_{12}\\ \mathbf{A}_{21}\mathbf{A}_{22}\end{bmatrix}\begin{bmatrix}\mathbf{x}_1\\ \mathbf{x}_2\end{bmatrix}=\begin{bmatrix}\mathbf{b}_1\\ \mathbf{b}_2\end{bmatrix},\quad \begin{array}{l}\mathbf{A}_{11}\in\mathbb{K}^{k,k},\mathbf{A}_{12}\in\mathbb{K}^{k,\ell},\mathbf{A}_{21}\in\mathbb{K}^{\ell,k},\mathbf{A}_{22}\in\mathbb{K}^{\ell,\ell},\\ \mathbf{x}_1\in\mathbb{K}^{k},\mathbf{x}_2\in\mathbb{K}^{\ell},\mathbf{b}_1\in\mathbb{K}^{k},\mathbf{b}_2\in\mathbb{K}^{\ell}.\end{array} \tag{2.6.0.3}$$

$$x_1 = A_{11}^{-1}(b_1 - A_{12}x_2) \quad \text{from 1st row}$$

Into 2nd. row $\quad \underbrace{(A_{22} - A_{21}A_{11}^{-1}A_{12})}_{\text{Schur complement matrix} \in \mathbb{R}^{\ell,\ell}} x_2 = b_2 - A_{21}A_{11}^{-1}b_1$

Useful, if $A_{11}x = c$ can be solved "easily"

Rem 2.3.2.19: Block LU-decomposition

With

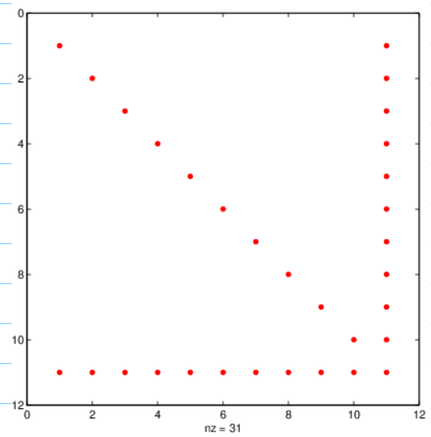
$$\underbrace{\begin{bmatrix}\mathbf{A}_{11} & \mathbf{A}_{12}\\ \mathbf{A}_{21} & \mathbf{A}_{22}\end{bmatrix}}_{\text{block LU-factorization}}=\begin{bmatrix}\mathbf{I} & 0\\ \mathbf{A}_{21}\mathbf{A}_{11}^{-1} & \mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{A}_{11} & \mathbf{A}_{12}\\ 0 & \mathbf{S}\end{bmatrix},$$

with Schur complement $\boxed{\mathbf{S}:=\mathbf{A}_{22}-\mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}}$. (2.3.2.20)

(2)

Ex 2.6.0.5 : LSE with arrow matrix

$$\mathbf{A} = \begin{bmatrix} \mathbf{D} & \mathbf{c} \\ \mathbf{b}^\top & \alpha \end{bmatrix}, \quad \mathbf{D} \in \mathbb{R}^{n,n} \text{ diagonal, regular} \tag{2.6.0.6}$$



C++ code 2.6.0.9: Dense Gaussian elimination applied to arrow system → GITLAB

```
VectorXd arrowsys_slow(const VectorXd &d, const VectorXd &c, const VectorXd &b,
                       const double alpha, const VectorXd &y) {
  int n = d.size();
  MatrixXd A(n + 1, n + 1); // Empty dense matrix
  A.setZero();              // Initialize with all zeros.
  A.diagonal().head(n) = d; // Initializee matrix diagonal from a vector.
  A.col(n).head(n) = c;     // Set rightmost column c.
  A.row(n).head(n) = b;     // Set bottom row b^T.
  A(n, n) = alpha;          // Set bottom-right entry α.
  return A.lu().solve(y);   // Gaussian elimination
}
```

→ Cost = $O(n^3)$

Block GE :

$$\mathbf{Ax} = \begin{bmatrix} \mathbf{D} & \mathbf{c} \\ \mathbf{b}^\top & \alpha \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \xi \end{bmatrix} = \mathbf{y} := \begin{bmatrix} \mathbf{y}_1 \\ \eta \end{bmatrix}, \tag{2.6.0.7}$$

$$\blacktriangleright \quad \xi = \frac{\eta - \mathbf{b}^T \mathbf{D}^{-1} \mathbf{y}_1}{\alpha - \mathbf{b}^T \mathbf{D}^{-1} \mathbf{c}}, \quad \longrightarrow \neq 0 \text{ required!} \tag{2.6.0.8}$$

$$\mathbf{x}_1 = \mathbf{D}^{-1}(\mathbf{y}_1 - \xi \mathbf{c}) .$$

→ $O(n)$ cost

C++ code 2.6.0.10: Solving an arrow system according to (2.6.0.8) → GITLAB

```
VectorXd arrowsys_fast(const VectorXd &d, const VectorXd &c, const VectorXd &b,
                       const double alpha, const VectorXd &y) {
  int n = d.size();
  VectorXd z = c.array() / d.array();          // z = D^{-1}c
  VectorXd w = y.head(n).array() / d.array(); // w = D^{-1}y_1
  const double den = alpha - b.dot(z);         // denominator in (2.6.0.8)
  // Check for (relatively!) small denominator
  if (std::abs(den) <
      std::numeric_limits<double>::epsilon() * (b.norm() + std::abs(alpha))) {
    throw std::runtime_error("Nearly singular system");
  }
  const double xi = (y(n) - b.dot(w)) / den;
  return (VectorXd(n + 1) << w - xi * z, xi).finished();
}
```

→ safe test == 0.0

▽ Fast code more vulnerable to round-off : instability

2.6.0.12 : LSE subject to low-rank modification

Assume : $A\underline{x} = \underline{b}$ is easy to solve, because

(i) A special structure

(ii) LU-dec. available

Sought : $\widetilde{\underline{x}}$: $\widetilde{A}\widetilde{\underline{x}} = \underline{b}$, where $\widetilde{A}$ arises from A by changing a single entry $(A)_{i^*,j^*}$

$\mathbf{A}, \widetilde{\mathbf{A}} \in \mathbb{K}^{n,n}$: $\tilde{a}_{ij} = \begin{cases} a_{ij} & \text{, if } (i,j) \neq (i^*,j^*) \text{,} \\ z + a_{ij} & \text{, if } (i,j) = (i^*,j^*) \text{,} \end{cases}$, $i^*, j^* \in \{1,\dots,n\}$. (2.6.0.13)

$z \in \mathbb{R}$

$\underline{e}_i$ = Cartesian basis vector

$$\widetilde{\mathbf{A}} = \mathbf{A} + z \cdot \mathbf{e}_{i^*}\mathbf{e}_{j^*}^T \quad \text{(2.6.0.14)}$$

$n \times n$ tensor-product matrix, rank = 1

$$\begin{bmatrix} 0 & \cdots & 0 \\ \vdots & z & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \leftarrow i^* \qquad \uparrow\, j^*$$

More general : rank-1 modification

$$\widetilde{A} = A + \underline{u}\underline{v}^T \,, \quad \underline{u}, \underline{v} \in \mathbb{R}^n$$

Trick : Block elimination

$$\begin{bmatrix} A & \underline{u} \\ \underline{v}^T & -1 \end{bmatrix} \begin{bmatrix} \widetilde{\underline{x}} \\ \xi \end{bmatrix} = \begin{bmatrix} \underline{b} \\ 0 \end{bmatrix}$$

block row II $\Rightarrow$ $\xi = \underline{v}^T \widetilde{\underline{x}} \overset{\text{I}}{\Rightarrow} \underbrace{(A + \underline{u}\underline{v}^T)}_{\widetilde{A}} \widetilde{\underline{x}} = \underline{b}$

block row I $\Rightarrow$ $\widetilde{\underline{x}} = A^{-1}(\underline{b} - \xi \underline{u})$

$\overset{\text{II}}{\Rightarrow}$ $\xi = \underline{v}^T A^{-1}(\underline{b} - \xi \underline{u})$

$\Rightarrow$ $\xi = \dfrac{\underline{v}^T A^{-1} \underline{b}}{1 + \underline{v}^T A^{-1} \underline{u}}$

$$\widetilde{\mathbf{x}} = \mathbf{A}^{-1}\mathbf{b} - \frac{\mathbf{A}^{-1}\mathbf{u}(\mathbf{v}^H(\mathbf{A}^{-1}\mathbf{b}))}{1 + \mathbf{v}^H(\mathbf{A}^{-1}\mathbf{u})} \cdot \quad \leftarrow \neq 0 ! \quad \text{(2.6.0.22)}$$

→ have to solve two LSE $Ax = \dots$

C++ code 2.6.0.23: Solving a rank-1 modified LSE ↪ GITLAB

```
2  // Solving rank-1 updated LSE based on (2.6.0.22)
3  template <class LUDec>
4  Eigen::VectorXd smw(const LUDec &lu, const Eigen::VectorXd &u,
5                      const Eigen::VectorXd &v, const Eigen::VectorXd &b) {
6    const Eigen::VectorXd z = lu.solve(b); // z = A^-1 b
7    const Eigen::VectorXd w = lu.solve(u); // w = A^-1 u
8    double alpha = 1.0 + v.dot(w); // Compute denominator of (2.6.0.22)
9    double beta = v.dot(z);        // Factor for numerator of (2.6.0.22)
10   if (std::abs(alpha) < std::numeric_limits<double>::epsilon() * std::abs(beta))
11     throw std::runtime_error("A nearly singular");
12   else
13     return (z - w * beta / alpha); // see (2.6.0.22)
14 }
```

cost $O(n^2)$, if LU-dec. available

$O(n)$ cost

safe check ≈ 0

Generalization of (2.6.0.22):

Lemma 2.6.0.21. Sherman-Morrison-Woodbury formula

For regular $\mathbf{A} \in \mathbb{K}^{n,n}$, *and* $\mathbf{U}, \mathbf{V} \in \mathbb{K}^{n,k}$, $n, k \in \mathbb{N}$, $k \leq n$, *holds*

$$(\mathbf{A} + \mathbf{U}\mathbf{V}^H)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{I} + \mathbf{V}^H\mathbf{A}^{-1}\mathbf{U})^{-1}\mathbf{V}^H\mathbf{A}^{-1} ,$$

if $\mathbf{I} + \mathbf{V}^H\mathbf{A}^{-1}\mathbf{U}$ *is regular.*

General rank k-matrix

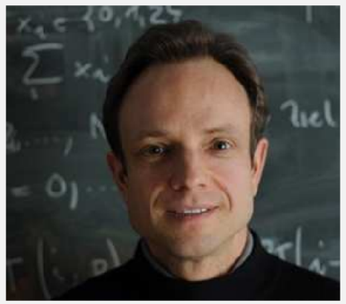
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 2.7.1: Sparse Matrix Storage Formats

Prof. R. Hiptmair, SAM, ETH Zurich

Date: September 18, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 2.1

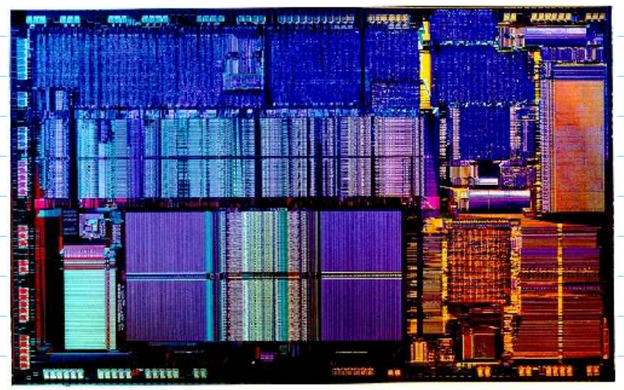
Duration: 20 minutes

II. Direct Methods for Square Linear Systems of Equations

2.7. Sparse Linear Systems

Sparse matrix :
- "almost all" entries = 0
- for which it is worth while exploiting this fact

Ex.: 2.7.0.4 : Sparse LSE in circuit simulation



- every is connected to only a few other nodes

→ LSE $A\underline{x} = \underline{b}$

nnz(A) ~ # nodes

▷ Sparse LSE arise in models of large networks

2.7.1. Sparse Matrix Storage Formats

Purposes :
- Memory ~ nnz(A)
- cost(A×vector) ~ nnz(A)
- provide easily accessible information about location of non-zero entries

§ 2.7.1.1. COO/triplet format

→ Stores A as a list/sequence tuples (i,j,value)

```
struct Triplet {
  size_t i; // row index
  size_t j; // column index
  scalar_t a; // additive contribution to matrix entry
};
using TripletMatrix = std::vector<Triplet>;
```

C++-code 2.7.1.2: Matrix×vector product y = Ax+y in triplet format

```
1 void multTriplMatvec(const TripletMatrix &A,
2                      const vector<scalar_t> &x,
3                      vector<scalar_t> &y)
4 for (size_t l=0; l<A.size(); l++) {
5   y[A[l].i] += A[l].a*x[A[l].j];
6  }
```

Allowed : repeated index pairs → values summed up

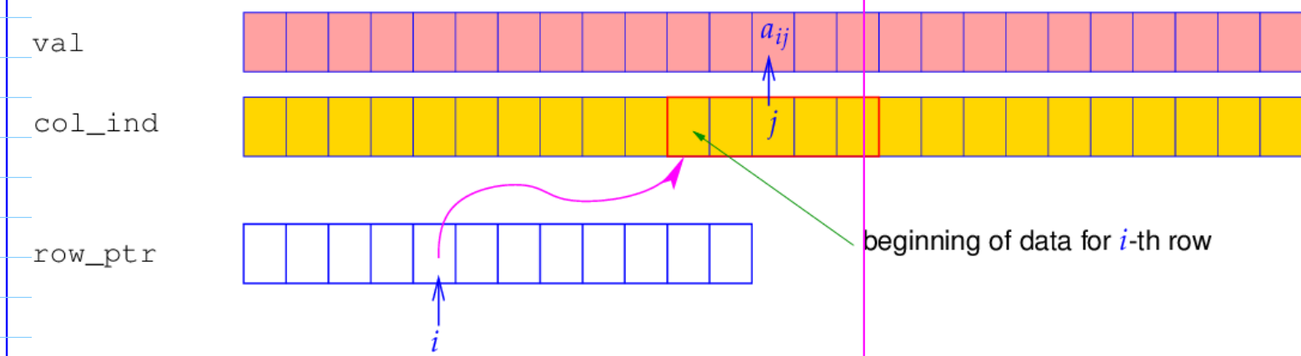
$$(A)_{i,j} = \sum_{k:\, A[k].i=i,\, A[k].j=j} A[k].a$$

§ 2.7.1.4 : Compressed row storage (CRS)

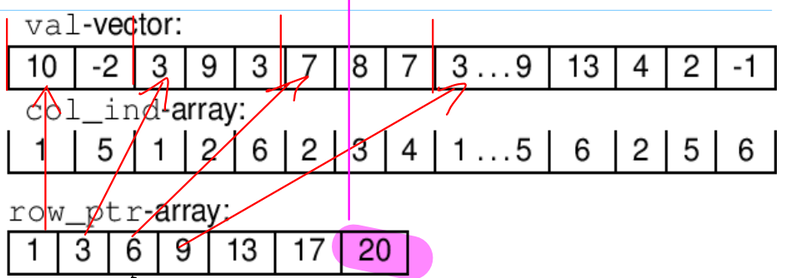
$A \in \mathbb{K}^{n,n}$: stores non-zero entries, row-pointer

`std::vector<scalar_t>`	`val`	size $\mathrm{nnz}(\mathbf{A}) := \#\{(i,j) \in \{1,\dots,n\}^2, a_{ij} \neq 0\}$
`std::vector<size_t>`	`col_ind`	size $\mathrm{nnz}(\mathbf{A})$
`std::vector<size_t>`	`row_ptr`	size $n+1$ & $\texttt{row\_ptr}[n+1] = \mathrm{nnz}(\mathbf{A})+1$ (sentinel value)

$$\texttt{val}[k] = a_{ij} \;\Leftrightarrow\; \begin{cases} \texttt{col\_ind}[k] = j, \\ \texttt{row\_ptr}[i] \le k < \texttt{row\_ptr}[i+1], \end{cases} \quad 1 \le k \le \mathrm{nnz}(\mathbf{A}).$$



$$\mathbf{A} = \begin{bmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{bmatrix}$$



[LA indexing]

CRS : Non-zero entries reside in contiguous memory!

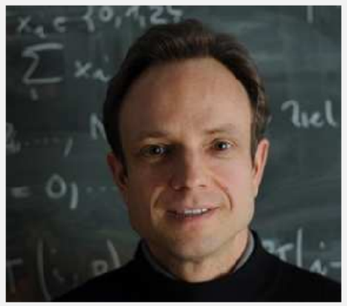
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 2.7.2: Sparse Matrices in EIGEN

Prof. R. Hiptmair, SAM, ETH Zurich

Date: September 18, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 1.2.1, Section 2.7.1

Duration: 12 minutes

II. Direct Methods for Square Linear Systems of Equations

2.7. Sparse Linear Systems

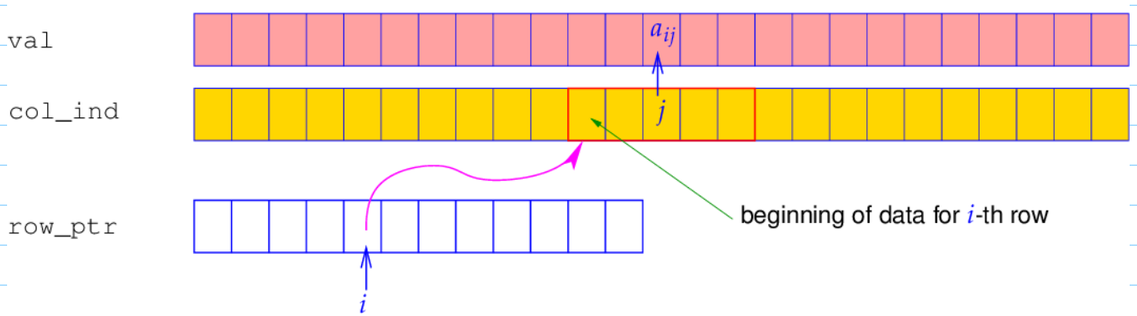
2.7.2. Sparse Matrices in Eigen

Standard: CRS/CCS format

```
#include <Eigen/Sparse>
Eigen::SparseMatrix<int, Eigen::ColMajor> Asp(rows,cols); // CCS format
Eigen::SparseMatrix<double, Eigen::RowMajor> Bsp(rows,cols); // CRS format
```

Challenge: Efficient initialization

$$\texttt{val}[k] = a_{ij} \;\Leftrightarrow\; \begin{cases} \texttt{col\_ind}[k] = j\,, \\ \texttt{row\_ptr}[i] \le k < \texttt{row\_ptr}[i+1]\,, \end{cases} \quad 1 \le k \le \mathrm{nnz}(\mathbf{A})\,.$$



(2)

Set random entry ▷ massive data movement necessary

I. Idea: Use intermediate COO/triplet format

```
std::vector <Eigen::Triplet <double > > triplets;
// .. fill the std::vector triplets ..
Eigen::SparseMatrix<double, Eigen::RowMajor> spMat(rows, cols);
spMat.setFromTriplets(triplets.begin(), triplets.end());
```

O(#triplets) - complexity

II. reserve() & insert(), if nnz/per row,col. known

C++-code 2.7.2.1: Accessing entries of a sparse matrix: potentially inefficient!

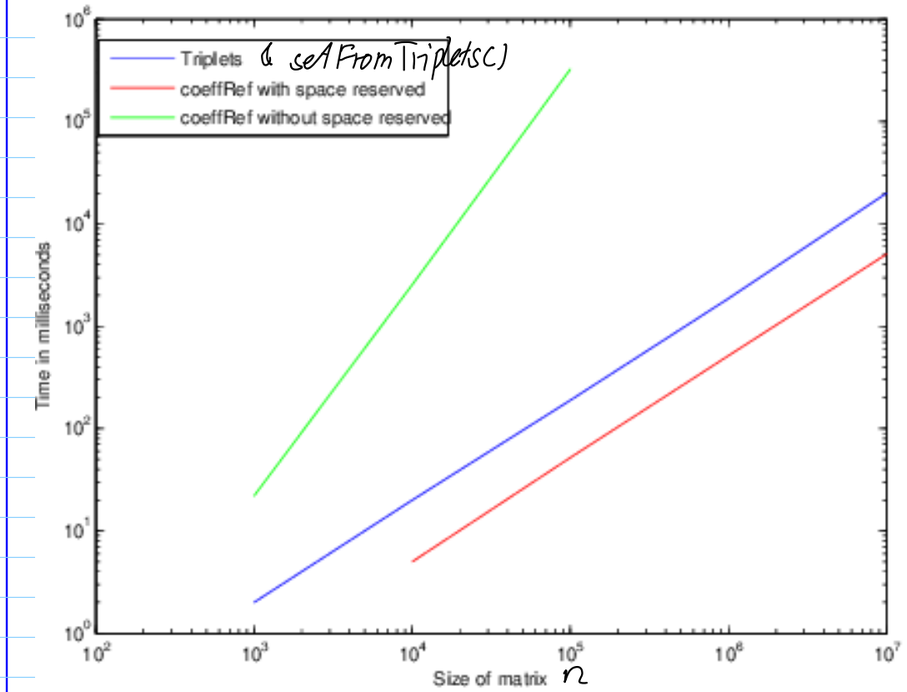
```
1  unsigned int rows,cols,max_no_nnz_per_row;
2  .....
3  SparseMatrix<double, RowMajor> mat(rows,cols);
4  mat.reserve(RowVectorXi::Constant(cols,max_no_nnz_per_row)); --> Allocation of enough space
5  // do many (incremental) initializations
6  for ( ) {
7    mat.insert(i,j) = value_ij;
8    mat.coeffRef(i,j) += increment_ij;
9  }
10 mat.makeCompressed();
```

} O(1), if enough space reserve()'d

↳ squeeze out zeros

Exp 2.7.2.2: Initialization of SparseMatrix<>

$A \in \mathbb{R}^{n,n}$ banded, $(A)_{i,j} \neq 0 \Leftrightarrow |i-j| \leq 2$



} O(n)

ETH Lecture 401-0663-00L Numerical Methods for CSE

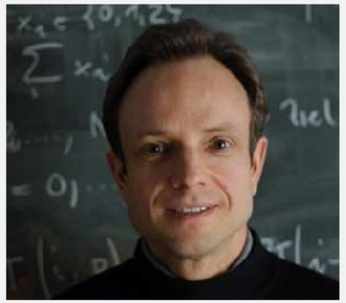
Course Video

Section 2.7.3: Direct Solution of Sparse Linear Systems of Equations

Prof. R. Hiptmair, SAM, ETH Zurich

Date: September 19, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 2.7.1, Section 1.4, Section 2.3

Duration: 18 minutes

II. Direct Methods for Square Linear Systems of Equations

2.7. Sparse Linear Systems

2.7.3. Direct Solution of Sparse Linear Systems of Equations

Assume: System matrix in sparse matrix format

↓

tells location of zero entries

↓

can be exploited by sparse elimination techniques

Eigen:

C++-code 2.7.3.1: Function for solving a sparse LSE with EIGEN ⇢ GITLAB

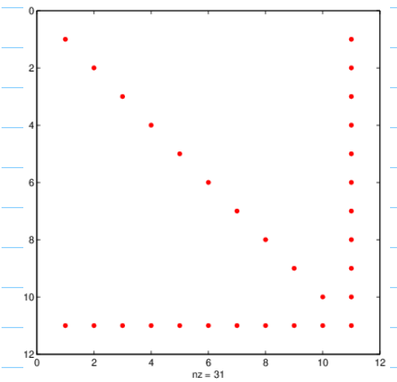
```
using SparseMatrix = Eigen::SparseMatrix<double>;
// Perform sparse elimination
void sparse_solve(const SparseMatrix &A, const VectorXd &b, VectorXd &x) {
  Eigen::SparseLU<SparseMatrix> solver(A);
  if (solver.info() != Eigen::Success) {
    throw "Matrix factorization failed";
  }
  x = solver.solve(b);
}
```

→ the expensive part

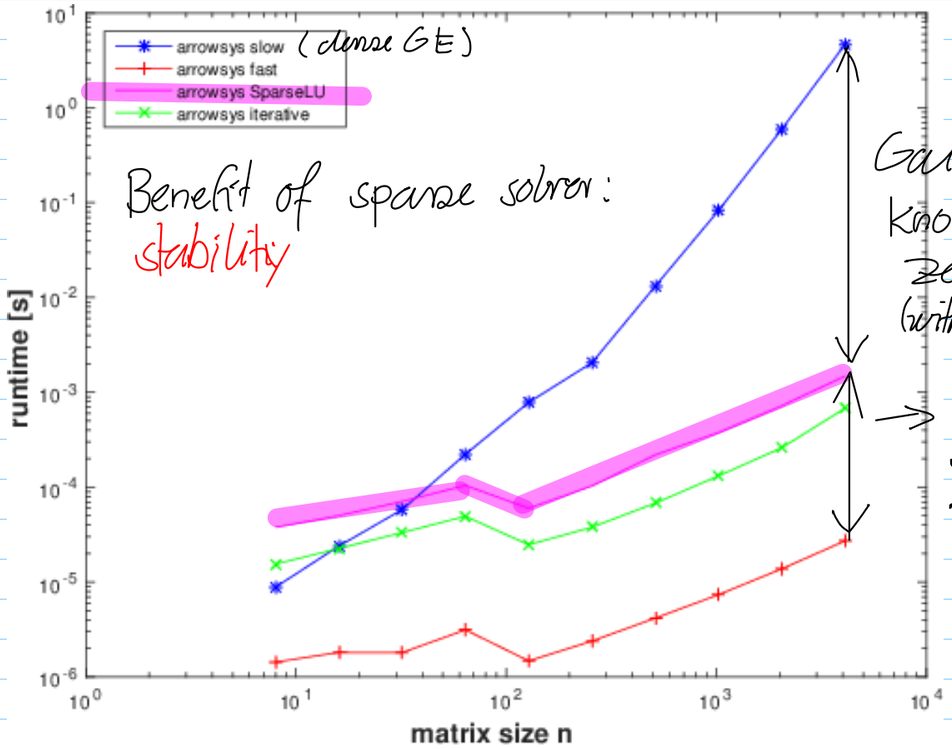
→ Check this

→ triangular solves: cheap

Ex 2.7.3.2: Sparse elimination for arrow matrix



← sparse arrow matrix



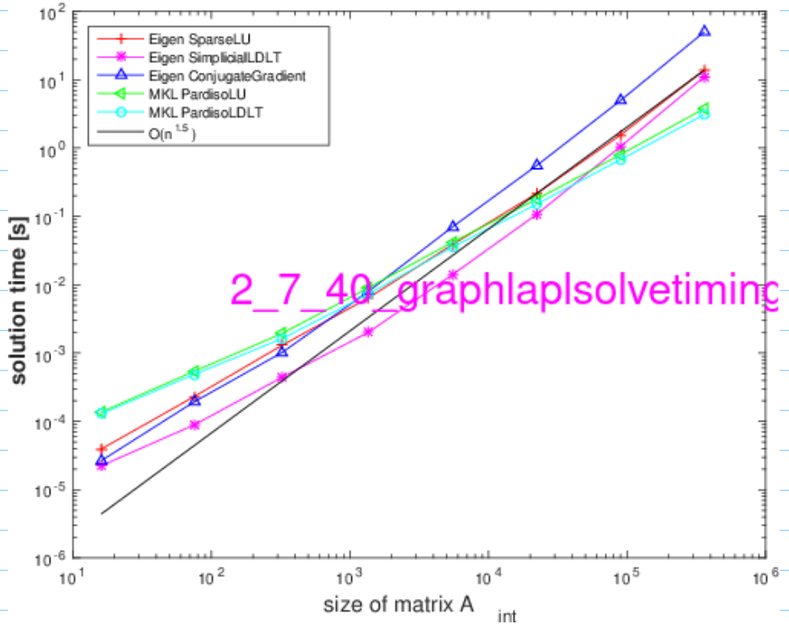
Benefit of sparse solver:
stability

Gain by exploiting knowledge about zero entries (without loss of quality)

→ Cost for
- matrix analysis
- pivoting

Exp. 2.7.3.4: Sparse elimination for combinatorial graph Laplacian

↳ sparse s.p.d., a few non-zero-entries per row/col.



Asymptotic runtime $O(n^{1.5})$

In practice: Cost (Sparse solution of $A\underline{x} = \underline{b}$)
$= O((nnz(A))^{\alpha})$ $\alpha \approx 1.5 - 2.5$

When solving linear systems of equations directly **dedicated sparse elimination solvers** from *numerical libraries* have to be used!

System matrices are passed to these algorithms in sparse storage formats (→ Section 2.7.1) to convey information about zero entries.

ETH Lecture 401-0663-00L Numerical Methods for CSE

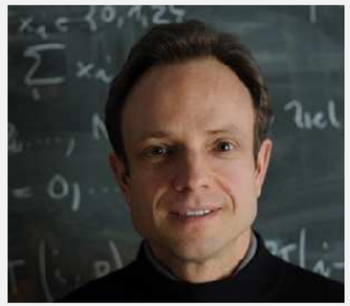
Course Video

Section 3.0.1: Overdetermined Linear Systems of Equations: Examples

Prof. R. Hiptmair, SAM, ETH Zurich

Date: September 21, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- Elementary linear algebra and geometry

Duration: 20 minutes

III. Direct Methods for Linear Least Squares Problems

Overdetermined LSE:

$$\mathbf{x} \in \mathbb{R}^n: \quad \text{“}\mathbf{A}\mathbf{x} = \mathbf{b}\text{”}, \qquad (3.0.0.1)$$

$$\mathbf{b} \in \mathbb{R}^m, \quad \mathbf{A} \in \mathbb{R}^{m,n}, \quad m \geq n.$$

$$\begin{bmatrix} \mathbf{A} \end{bmatrix} \begin{bmatrix} \mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \end{bmatrix}$$

tall system matrices

3.0.1 Overdetermined Linear Systems of Equations

Ex 3.0.1.1: Linear parameter estimation in 1D

Law: $y = \alpha x + \beta$ for some unknown parameters $\alpha, \beta \in \mathbb{R}$

Measured: $(x_i, y_i) \in \mathbb{R}^2$, $i = 1, \dots, m$, $m > 2$

$y_i = \alpha x_i + \beta \quad \forall i \Rightarrow$ LSE

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_m & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \quad \leftrightarrow \quad \mathbf{A}\mathbf{x} = \mathbf{b}, \ \mathbf{A} \in \mathbb{R}^{m,2}, \mathbf{b} \in \mathbb{R}^m, \mathbf{x} \in \mathbb{R}^2. \tag{3.0.1.3}$$

↳ will be perturbed ⇒ no solutions

However: measurement error have occurred

Ex 3.0.1.4: Linear regression: linear parameter estimation

Law: $y = \underline{a}^T \underline{x} + \beta$, parameters $\underline{a} \in \mathbb{R}^n$, $\beta \in \mathbb{R}$, $\underline{x} \in \mathbb{R}^n$

Measurements: $(\underline{x}_i, y_i) \in \mathbb{R}^n \times \mathbb{R}$, $i = 1, \dots, m$

$y_i = \underline{a}^T \underline{x}_i + \beta$

OD, if $m > n+1$

$$\begin{bmatrix} \mathbf{x}_1^\top & 1 \\ \vdots & \vdots \\ \mathbf{x}_m^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \beta \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \quad \leftrightarrow \quad \mathbf{A}\mathbf{x} = \mathbf{b}, \ \mathbf{A} \in \mathbb{R}^{m,n+1}, \mathbf{b} \in \mathbb{R}^m, \mathbf{x} \in \mathbb{R}^{n+1}, \tag{3.0.1.5}$$

↳ may be perturbation

Ex 3.0.1.6: Measuring the angles of a triangle

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} \tilde{\alpha} \\ \tilde{\beta} \\ \tilde{\gamma} \\ \pi \end{bmatrix}. \qquad \tilde{\alpha}, \tilde{\beta}, \tilde{\gamma} \hat{=} \text{measured angles} \tag{3.0.1.7}$$

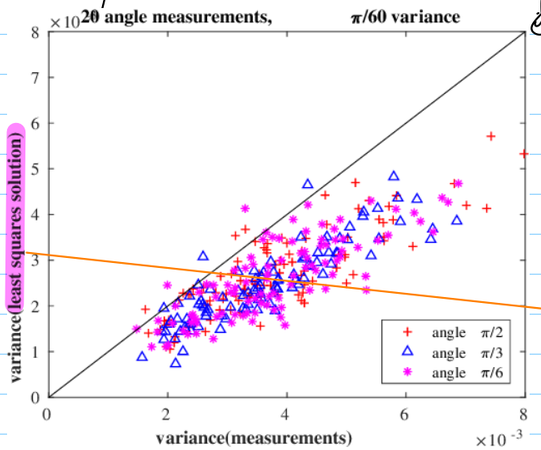
Σ angles $= \pi$

Principle in data science: You cannot afford not to use every piece of information available

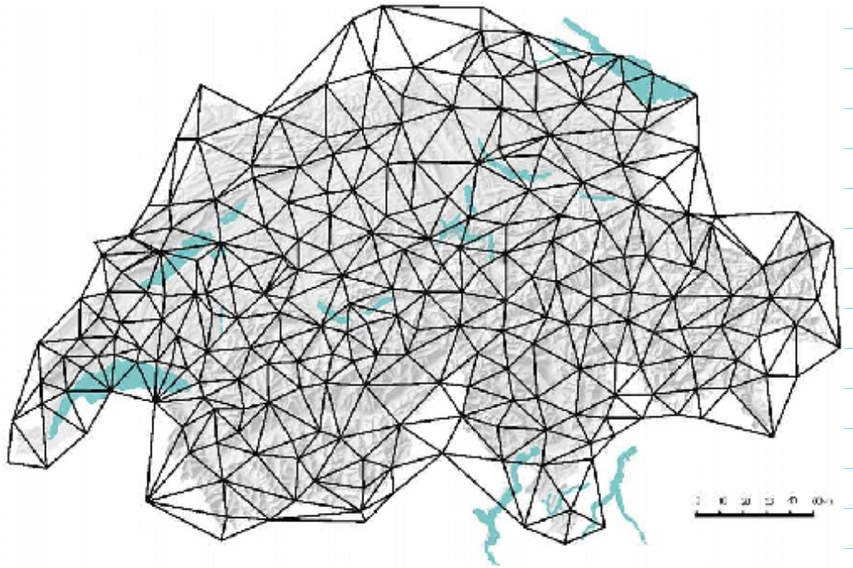
◁ Randomly perturbed angles

▷ Variance reduction

this chapter



Ex 3.0.1.8 : Angles in triangulations



Measured: angles

1. each angle is supposed to be equal to its measured value,
2. the sum of interior angles is π for every triangle,
3. the sum of the angles at an interior node is 2π.

↳ + #triag. extra equ.
↳ + #int. nodes extra equ. $\Rightarrow$ OD LSE

NumCSE@ETHZ

Ex. 3.0.1.9. Point locations from distances

1D (on a line): $x_i \in \mathbb{R}$, $i = 1, \dots, n$, $n > 2$

[Assume $x_i < x_{i+1}$]

Measure: $d_{i,j} = |x_i - x_j|$, $i \neq j$

? Use all data $\Rightarrow$ OD LSE

$$x_i - x_j = d_{ij}, \quad 1 \le j < i \le n. \quad \leftrightarrow \quad \mathbf{Ax} = \mathbf{b}$$

$$\begin{bmatrix} -1 & 1 & 0 & \dots & & \dots & 0 \\ -1 & 0 & 1 & 0 & & & \\ \vdots & & \ddots & \ddots & & & \\ \vdots & & & \ddots & \ddots & & \\ -1 & \dots & & & & 0 & 1 \\ 0 & -1 & 1 & & & & 0 \\ \vdots & & & & & & \vdots \\ \vdots & & & & & & \vdots \\ 0 & \dots & & & & -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_{12} \\ d_{13} \\ \vdots \\ \vdots \\ d_{1n} \\ d_{23} \\ \vdots \\ \vdots \\ d_{n-1,n} \end{bmatrix} \tag{3.0.1.10}$$

▷ $\binom{n}{2}$ equ. for n unknowns

Review questions:

Please answer review questions 3.0.1.11 without aids.

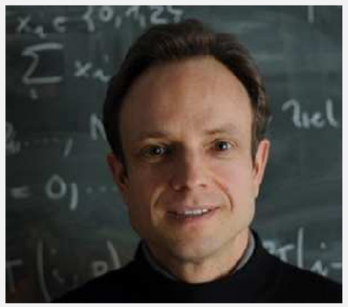
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 3.1.1: Least Squares Solutions

Prof. R. Hiptmair, SAM, ETH Zurich

Date: September 21, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- Euclidean space: norms and orthogonality

Duration: 18 minutes

III. Direct Methods for Linear Least Squares Problems

3.1. Least Squares Solution Concepts

Setting: OD-LSE $A\underline{x} = \underline{b}$, $A \in \mathbb{R}^{m,n}$, $\underline{b} \in \mathbb{R}^m$, $m \geq n$

3.1.1. Least Squares Solutions: Definition

Idea: A LSQ sol. of $A\underline{x} = \underline{b}$ is a vector $x \in \mathbb{R}^n$ that makes the residual $r := \underline{b} - A\underline{x}$ as small as possible (w.r.t. $\|\cdot\|_2$)

Definition 3.1.1.1. Least squares solution

For given $\mathbf{A} \in \mathbb{R}^{m,n}$, $\mathbf{b} \in \mathbb{R}^m$ the vector $\mathbf{x} \in \mathbb{R}^n$ is a least squares solution of the linear system of equations $\mathbf{Ax} = \mathbf{b}$, if

need not be unique

$$\mathbf{x} \in \operatorname*{argmin}_{\mathbf{y}\in\mathbb{R}^n} \|\mathbf{Ay} - \mathbf{b}\|_2^2, =: lsq(A,b)$$

$$\Updownarrow$$

$$\|\mathbf{Ax} - \mathbf{b}\|_2^2 = \min_{\mathbf{y}\in\mathbb{R}^n} \|\mathbf{Ay} - \mathbf{b}\|_2^2 = \min_{y_1,\ldots,y_n\in\mathbb{R}} \sum_{i=1}^{m} \left(\sum_{j=1}^{n} (\mathbf{A})_{i,j} y_j - (\mathbf{b})_i \right)^2 .$$

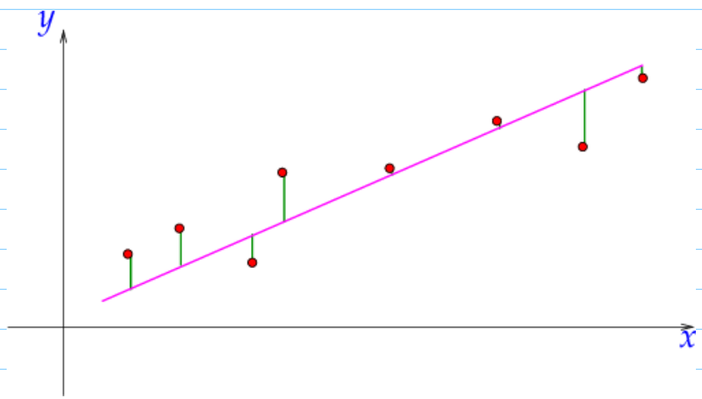
A true generalization:

$A \in \mathbb{R}^{n,n}$ regular $\Rightarrow$ $Lsq(A,b) := \{A^{-1}b\}$

Ex 3.1.1.5: 1D linear regression, Ex 3.0.1.1 cnt'd

OD-LSE: $\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_m & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$ $\leftrightarrow$ $\mathbf{A}\mathbf{x} = \mathbf{b}$, $\mathbf{A} \in \mathbb{R}^{m,2}$, $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{x} \in \mathbb{R}^2$. (3.0.1.3)

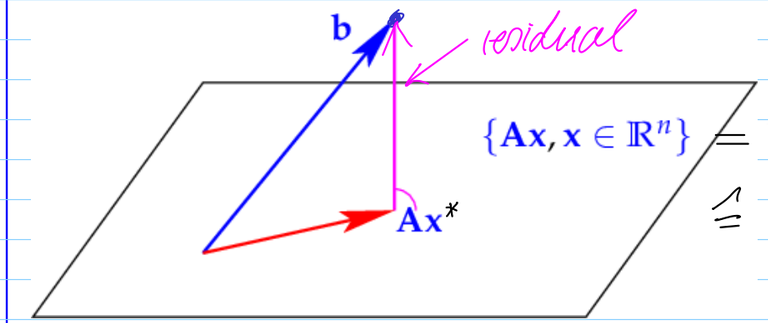
LSQ sol: $\begin{bmatrix} \alpha \\ \beta \end{bmatrix} \in \underset{\tilde{\alpha},\tilde{\beta}}{\operatorname{argmin}} \sum_{i=1}^{m} (y_i - \tilde{\alpha}x_i - \tilde{\beta})^2$



▷ Minimize the sum of squares of vertical distances of the data point from the regression line!

NumCSE@ETHZ

§ 3.1.1.8: Geometric interpretation of least squares solutions of $Ax = b$:



$= \mathcal{R}(A) \subset \mathbb{R}^m$ (range of A)
$\hat{=}$ a (hyper) plane

Ax^* is closest to b in $\mathcal{R}(A)$, $x^* :=$ LSQ sol.

$\Leftrightarrow$ Ax^* is the orthogonal projection of b onto $\mathcal{R}(A)$

$Ax^* = b_0 \hat{=}$ orthogonal projection of b onto $\mathcal{R}(A)$

Geometric intuition: Least squares solutions always exist:

$\forall A \in \mathbb{R}^{m,n}, b \in \mathbb{R}^m$: $Lsq(A,b) \neq \emptyset$

→ Theorem 3.1.1.9

NumCSE@ETHZ

Review questions:

Please tackle Review Questions 3.1.1.14 without consulting the lecture document.

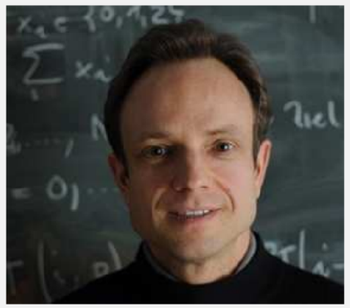
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 3.1.2: Normal Equations

Prof. R. Hiptmair, SAM, ETH Zurich

Date: September 21, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 3.1.1

Duration: 30 minutes

Note: Possible minor *mismatch of video and tablet notes*!

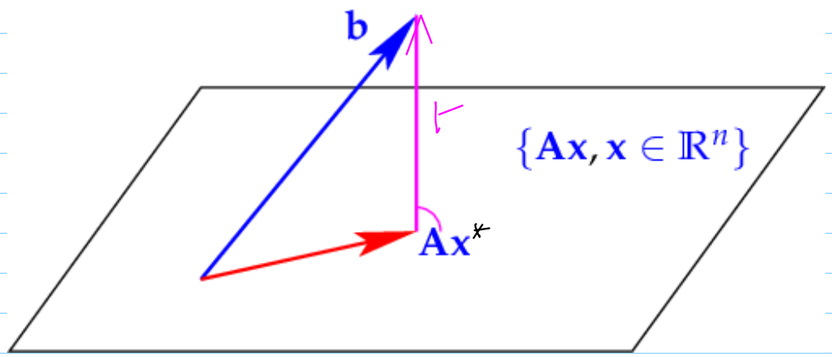
[Corrections and updates can be incorporated into tablet notes only]

III. Direct Methods for Linear Least Squares Problems

3.1. Least Squares Solution Concepts

Setting: OD-LSE $A\underline{x} = \underline{b}$, $A \in \mathbb{R}^{m,n}$, $\underline{b} \in \mathbb{R}^m$, $m \geq n$

3.1.2. Normal Equations



$x^* \in lsq(A, b)$

② Residual $r := b - Ax^* \perp \mathcal{R}(A) = \{Ay : y \in \mathbb{R}^n\}$

$$(Ay)^T(b - Ax^*) = 0 \quad \forall y \in \mathbb{R}^n$$

$$\Rightarrow A^T(b - Ax^*) = 0$$

Theorem 3.1.2.1. Obtaining least squares solutions by solving normal equations

The vector $\mathbf{x} \in \mathbb{R}^n$ is a least squares solution (→ Def. 3.1.1.1) of the linear system of equations $\mathbf{Ax} = \mathbf{b}$, $\mathbf{A} \in \mathbb{R}^{m,n}$, $\mathbf{b} \in \mathbb{R}^m$, if and only if *it solves the* normal equations (NEQ)

$$x \in \mathrm{lsq}(A,b) \iff \boxed{\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}} \quad . \quad n \times n \text{ square LSE} \qquad (3.1.2.2)$$

II NEQ : Size of system matrix

$$\begin{bmatrix} \mathbf{A}^T \end{bmatrix} \begin{bmatrix} \mathbf{A} \end{bmatrix} \begin{bmatrix} \mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{A}^T \end{bmatrix} \begin{bmatrix} \mathbf{b} \end{bmatrix},$$

$$\Leftrightarrow \underbrace{\begin{bmatrix} \mathbf{A}^T\mathbf{A} \end{bmatrix}}_{\in \mathbb{R}^{n,n}} \begin{bmatrix} \mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{A}^T \end{bmatrix} \begin{bmatrix} \mathbf{b} \end{bmatrix} \in \mathbb{R}^n$$

Rem 3.1.2.5 : "Analysis-style derivation"

Define : $J : \mathbb{R}^n \to \mathbb{R}$

$$J(y) := \|b - Ay\|_2^2$$

$$J(\mathbf{y}) = \mathbf{y}^T\mathbf{A}^T\mathbf{A}\mathbf{y} - 2\mathbf{b}^T\mathbf{A}\mathbf{y} + \mathbf{b}^T\mathbf{b}$$

$$= \underbrace{\sum_{i=1}^n \sum_{j=1}^n (\mathbf{A}^T\mathbf{A})_{ij} y_i y_j - 2\sum_{i=1}^m \sum_{j=1}^n b_i (\mathbf{A})_{ij} y_j + \sum_{i=1}^m b_i^2}_{\text{multivariate polynomial in } y : J \text{ smooth}, C^\infty}, \quad \mathbf{y} = [y_1, \ldots, y_n]^T \in \mathbb{R}^n .$$

$$x^* \in \operatorname*{argmin}_{x \in \mathbb{R}^n} J(x) \Rightarrow \underbrace{\operatorname{grad} J}_{\mathbb{R}^n \to \mathbb{R}^n}(x^*) = 0$$

$$\operatorname{grad} J(y) = \left[\frac{\partial J}{\partial y_i}(y)\right]_{i=1}^n = 2A^TAy - 2A^Tb \Big\} \text{(NEQ)}$$

Ex 3.1.2.4 : NEQ for linear regression

$$\begin{bmatrix} \mathbf{x}_1^T & 1 \\ \vdots & \vdots \\ \mathbf{x}_m^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \beta \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \quad \leftrightarrow \quad \mathbf{A}\mathbf{x} = \mathbf{b}, \ \mathbf{A} \in \mathbb{R}^{m,n+1}, \mathbf{b} \in \mathbb{R}^m, \mathbf{x} \in \mathbb{R}^{n+1}, \qquad (3.0.1.5)$$

Tool: Block matrix multiplication

$A = [X^T 1]$, $X = [x_1, x_2, \ldots, x_m] \in \mathbb{R}^{n,m}$

$1 = [1, \ldots, 1]^T \in \mathbb{R}^m$

③

NEQ:

$$A^TA = \begin{bmatrix} XX^T & X\cdot 1 \\ 1^TX^T & m \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} Xy \\ 1^Ty \end{bmatrix}$$

▷ Note: System matrix of NEQ is symmetric

Explicit NEQ for 1D regression

Issue: Uniqueness of LSQ sols.

Theorem 3.1.2.9. Kernel and range of $\mathbf{A}^\top\mathbf{A}$

For $\mathbf{A} \in \mathbb{R}^{m,n}$, $m \geq n$, holds

(I)
$$\mathcal{N}(\mathbf{A}^\top\mathbf{A}) = \mathcal{N}(\mathbf{A}), \tag{3.1.2.10}$$
$$\mathcal{R}(\mathbf{A}^\top\mathbf{A}) = \mathcal{R}(\mathbf{A}^\top). \tag{3.1.2.11}$$

Proof of (I):

$z \in \mathcal{N}(A^TA) \Rightarrow z^TA^TAz = 0 \Leftrightarrow \|Az\|^2 = 0 \Leftrightarrow Az = 0$

$\Leftrightarrow z \in \mathcal{N}(A)$ □

Corollary 3.1.2.13. Uniqueness of least squares solutions

If $m \geq n$ and $\mathcal{N}(\mathbf{A}) = \{\mathbf{0}\}$, then the linear system of equations $\mathbf{Ax} = \mathbf{b}$, $\mathbf{A} \in \mathbb{R}^{m,n}$, $\mathbf{b} \in \mathbb{R}^m$, has a unique least squares solution (→ 3.1.1.1)

$$\mathbf{x} = (\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top\mathbf{b}, \tag{3.1.2.14}$$

that can be obtained by solving the normal equations (3.1.2.2).

$\mathcal{N}(A^TA) = \{0\} \Rightarrow A^TA$ is regular!

Note: $\mathcal{N}(A) = \{0\} \Rightarrow A^TA$ s.p.d.

Criterion: $A \in \mathbb{R}^{m,n}$, $m > n$: $\mathcal{N}(A) = \{0\} \Leftrightarrow \underline{\text{rank}(A) = n}$

Full-rank condition (FRC)

Ex: 3.1.2.17: FRC for 1D linear regression

$$\operatorname{rank}\left(\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_m & 1 \end{bmatrix}\right) = 2 \Leftrightarrow \exists i,j \in \{1,\dots,m\}: x_i \neq x_j,$$

$\Leftrightarrow$ ∃ two linearly indep rows

(4)

▷ No meaningful regression line

Rem.: Failure of FRC points to
- flawed mathematical model
- useless data

Review questions :

Try to answer Review questions 3.1.2.23 without consulting other parts of the lecture document.

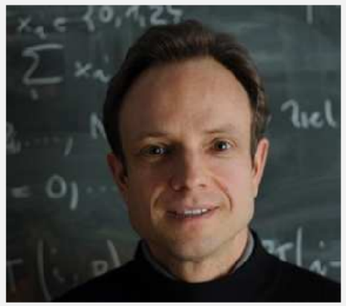
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 3.1.3: Moore-Penrose Pseudoinverse

Prof. R. Hiptmair, SAM, ETH Zurich

Date: September 21, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 3.1.1

Duration: 15 minutes

III. Direct Methods for Linear Least Squares Problems

3.1. Least Squares Solution Concepts

Setting: OD-LSE $A\underline{x} = \underline{b}$, $A \in \mathbb{R}^{m,n}$, $\underline{b} \in \mathbb{R}^m$, $m \geq n$

3.1.3. Moore-Penrose Pseudoinverse

LSQ sol. is not unique:

FRC violated $\Leftrightarrow$ $\operatorname{rank}(A) < n \Leftrightarrow \mathcal{N}(A) \neq \{0\}$

$\rightarrow$ Additional selection criterion

"Minimal norm condition"

Definition 3.1.3.1. Generalized solution of a linear system of equations

The generalized solution $\mathbf{x}^\dagger \in \mathbb{R}^n$ of a linear system of equations $\mathbf{Ax} = \mathbf{b}$, $\mathbf{A} \in \mathbb{R}^{m,n}$, $\mathbf{b} \in \mathbb{R}^m$, is defined as

$$\mathbf{x}^\dagger := \operatorname{argmin}\{\|\mathbf{x}\|_2 : \mathbf{x} \in \operatorname{lsq}(\mathbf{A}, \mathbf{b})\} . \tag{3.1.3.2}$$

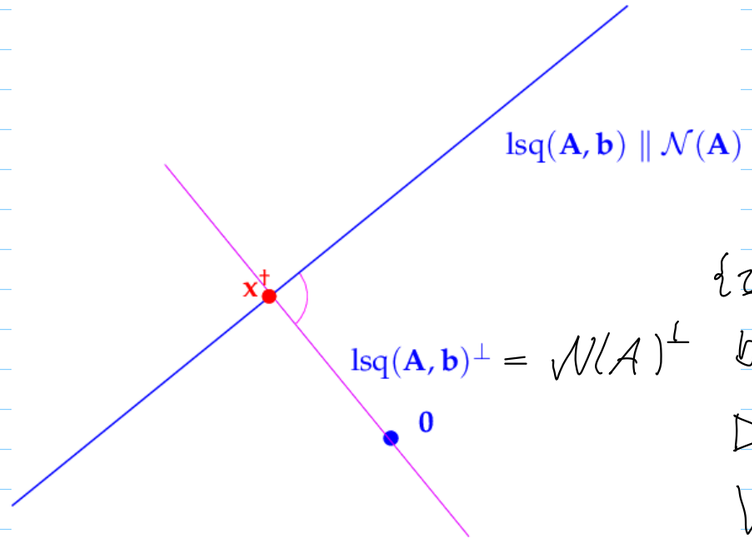
(2)

$$Lsq(A,\underline{b}) = \{x \in \mathbb{R}^n : A^TAx = A^T\underline{b}\}$$

[an affine subspace of $\mathbb{R}^n$]

$$= x_0 + \mathcal{N}(A), \quad x_0 \in Lsq(A,\underline{b})$$

Visualization:



$x^+ \perp \mathcal{N}(A)$

$x^+ \in \mathcal{N}(A)^\perp$

[orth. complement]

$\{v_1, \dots, v_k\}$, $k := \dim \mathcal{N}(A)^\perp$ basis of $\mathcal{N}(A)^\perp$

$\triangleright \; x^+ = Vy, y \in \mathbb{R}^k$

$V = [v_1, \dots, v_k] \in \mathbb{R}^{n,k}$

+ NEQ: $\underbrace{V^TA^TAV}_{\in \mathbb{R}^{k,k} \text{ regular}} y = V^TA^T\underline{b}$

Proof by contradiction.

$$x^+ = Vy$$

Theorem 3.1.3.6. Formula for generalized solution

Given $\mathbf{A} \in \mathbb{R}^{m,n}$, $\mathbf{b} \in \mathbb{R}^m$, the generalized solution $\mathbf{x}^\dagger$ of the linear system of equations $\mathbf{Ax} = \mathbf{b}$ is given by

$$\mathbf{x}^\dagger = \mathbf{V}(\mathbf{V}^\top\mathbf{A}^\top\mathbf{AV})^{-1}(\mathbf{V}^\top\mathbf{A}^\top\mathbf{b}),$$

where $\mathbf{V}$ is any matrix whose columns form a basis of $\mathcal{N}(\mathbf{A})^\perp$.

↓

Moore-Penrose pseudoinverse A^+

A^+ is independent of the choice of V!

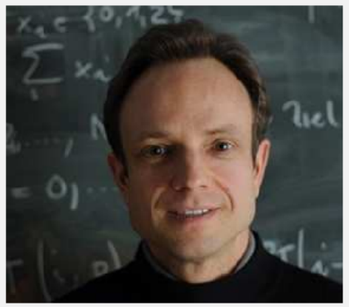
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 3.2: Normal Equation Methods

Prof. R. Hiptmair, SAM, ETH Zurich

Date: September 23, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 3.1.2, Section 2.5, [Section 2.8]

Duration: 15 minutes

III. Direct Methods for Linear Least Squares Problems

3.2. Normal Equation Methods

Setting: OD-LSE $A\underline{x} = \underline{b}$, $A \in \mathbb{R}^{m,n}$, $\underline{b} \in \mathbb{R}^m$, $m \geq n$

NEQ: $A^TAx = A^Tb \Leftrightarrow \{x\} = \mathrm{lsq}(A, \underline{b})$

Assume: FRC $\Leftrightarrow$ $\mathrm{rank}(A) = n$ $\Leftrightarrow$ A^TA regular

Algorithm: Normal equation method to solve full-rank least squares problem $\mathbf{Ax} = \mathbf{b}$

1. Compute regular matrix $\mathbf{C} := \mathbf{A}^\top\mathbf{A} \in \mathbb{R}^{n,n}$.
2. Compute right hand side vector $\mathbf{c} := \mathbf{A}^\top\mathbf{b}$.
3. Solve s.p.d. ($\to$ Def. 1.1.2.6) linear system of equations: $\mathbf{Cx} = \mathbf{c}$ $\to$ § 2.8.0.13

Definition 1.1.2.6. Symmetric positive definite (s.p.d.) matrices $\to$,

$\mathbf{M} \in \mathbb{K}^{n,n}$, $n \in \mathbb{N}$, is symmetric (Hermitian) positive definite (s.p.d.), if

$$\mathbf{M} = \mathbf{M}^H \quad \text{and} \quad \forall \mathbf{x} \in \mathbb{K}^n: \ \mathbf{x}^H\mathbf{M}\mathbf{x} > 0 \ \Leftrightarrow \ \mathbf{x} \neq 0 \, .$$

If $\mathbf{x}^H\mathbf{M}\mathbf{x} \geq 0$ for all $\mathbf{x} \in \mathbb{K}^n$ $\triangleright$ $\mathbf{M}$ positive semi-definite.

② ⏸

$$x^T C x = x^T A^T A x = (Ax)^T (Ax)$$
$$= \|Ax\|_2^2 > 0 \iff x \neq 0$$
$$\uparrow$$
FRC : $\mathcal{N}(A) = \{0\}$

C++ code 3.2.0.1: Solving a linear least squares problem via normal equations

```
//!  Solving the overdetermined linear system of equations
//!  Ax = b by solving normal equations (3.1.2.2)
//!  The least squares solution is returned by value
VectorXd normeqsolve(const MatrixXd &A,const VectorXd &b) {
  if (b.size() != A.rows()) throw runtime_error("Dimension mismatch");
  // Use Cholesky factorization for s.p.d.  system matrix, § 2.8.0.13
  VectorXd x = (A.transpose()*A).llt().solve(A.transpose()*b);
  return x;
}
```

① $O(mn^2)$ ③ ② $O(mn)$

↓

Special LU-type factorization for s.p.d. matrices : $O(n^3)$

$$\left[\mathbf{A}^\top\right]\left[\mathbf{A}\right]\left[\mathbf{x}\right] = \left[\mathbf{A}^\top\right]\left|\mathbf{b}\right|,$$

① ↓ ② ↓

$$\Leftrightarrow \quad \left[\mathbf{A}^\top\mathbf{A}\right]\left[\mathbf{x}\right] = \left[\mathbf{A}^\top\right]\left|\mathbf{b}\right| .$$

$\in \mathbb{R}^{n,n}$

step ❶: cost $O(mn^2)$
step ❷: cost $O(nm)$
step ❸: cost $O(n^3)$

} ▶ cost $O(n^2m + n^3)$ for $m, n \to \infty$.

↓

Linear in m, if n fixed/small

Ex 3.2.0.4: NEQ & roundoff errors

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ \delta & 0 \\ 0 & \delta \end{bmatrix} \Rightarrow \mathbf{A}^\top\mathbf{A} = \begin{bmatrix} 1+\delta^2 & 1 \\ 1 & 1+\delta^2 \end{bmatrix}$$

If $\delta \approx \sqrt{\text{EPS}}$, then $1 + \delta^2 = 1$ in $\mathbb{M}$.

Computed: $A^TA = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

$1 = (1+\delta^2)(1+\varepsilon) \Rightarrow |\varepsilon| = \frac{\delta^2}{1+\delta^2} < \text{EPS}$

→ complies w/ "axiom of floating point arithmetic"

C++-code 3.2.0.5:

```
int main() {
  MatrixXd A(3,2);
  // Inquire about machine precision → Ex. 1.5.3.12
  double eps = std::numeric_limits<double>::epsilon();
  // « initialization of matrix → § 1.2.1.3
  A << 1, 1, sqrt(eps), 0, 0, sqrt(eps);
  // Output rank of A^TA
  std::cout << "Rank of A: " << A.fullPivLu().rank() << std::endl
            << "Rank of A^TA: "
            << (A.transpose() * A).fullPivLu().rank() << std::endl;
  return 0;
}
```

Output:

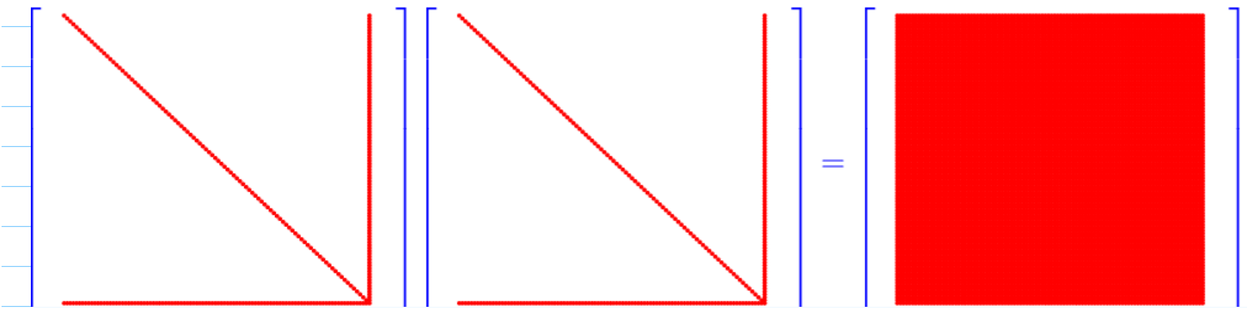
```
Rank of A: 2
Rank of A^T*A: 1
```

NumCSE@ETHZ

Rem 3.2.06 : NEQ : Loss of sparsity

[A concern only for large m, n]

A sparse $\not\Longleftrightarrow$ A^TA sparse



"out of memory"

A trick to avoid computing A^TA :

§ 3.2.07 Extended normal equations

NEQ $A^TA\underline{x} = A^T\underline{b} \Leftrightarrow A^T(\underline{b} - A\underline{x}) = 0$

residual $\underline{r}$

Idea: $\underline{r}$ as new unknown & split equations

$$\left.\begin{aligned} \underline{r} &= \underline{b} - A\underline{x} \\ A^T\underline{r} &= 0 \end{aligned}\right\} \Rightarrow \underbrace{\begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix}}_{\in \mathbb{R}^{m+n,m+n}, \text{ but sparse, if } A \text{ sparse}} \begin{bmatrix} \underline{r} \\ \underline{x} \end{bmatrix} = \begin{bmatrix} \underline{b} \\ 0 \end{bmatrix}$$

Review questions :

Answer review questions 3.2.0.11 without aids

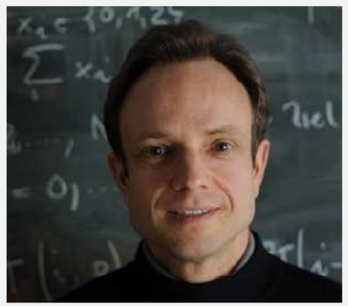
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 3.3: Orthogonal Transformation Methods

Prof. R. Hiptmair, SAM, ETH Zurich

Date: September 23, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.
- From LA: orthogonal matrices

Dependencies. Section 3.1.1

Duration: 15 minutes

III. Direct Methods for Linear Least Squares Problems

3.3. Orthogonal Transformation Methods

Setting: OD LSE $A\underline{x} = \underline{b}$, $A \in \mathbb{R}^{m,n}$, $m \geq n$

Assume: FRC $:\Leftrightarrow$ $\operatorname{rank}(A) = n$

LSQ sol.: $\{x\} = \operatorname{lsq}(A,\underline{b}) = \operatorname{argmin}_{y \in \mathbb{R}^n} \|\underline{b} - A\underline{y}\|_2^2$

3.3.1. Orthogonal Transformation Methods: Idea

As GE: Want to transform $A\underline{x} = \underline{b} \longrightarrow \tilde{A}\underline{x} = \tilde{\underline{b}}$

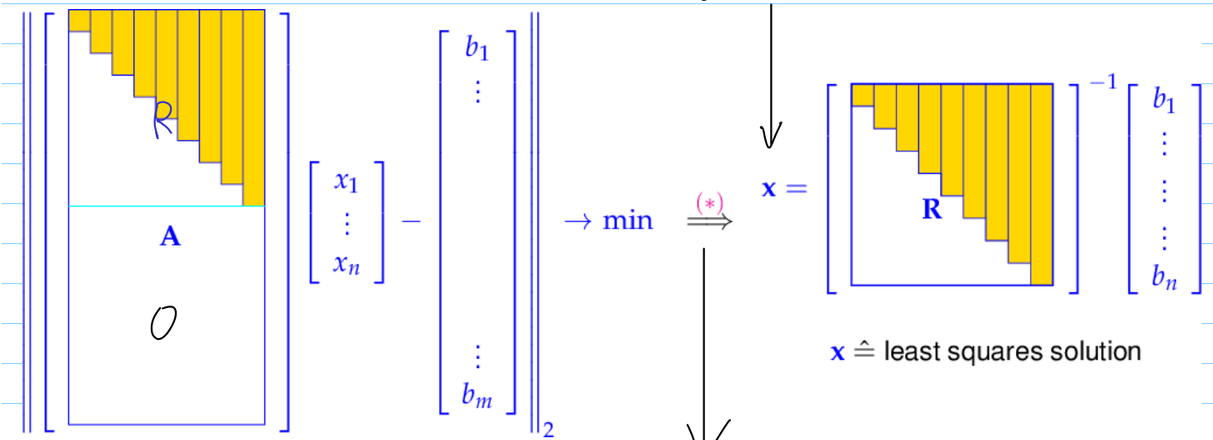
s.t. (i) LSQ sol. of $\tilde{A}\underline{x} = \tilde{\underline{b}}$ "easy to compute"

(ii) $\operatorname{lsq}(A,\underline{b}) = \operatorname{lsq}(\tilde{A},\tilde{\underline{b}})$

(i) for GE: triangular system

(2)

(i) A triangular



$\mathbf{x} \hat{=}$ least squares solution

rank(A) = n → R regular

Make first n component of $Ax - b$ vanish

(ii) Idea: $\|\cdot\|_2$-preserving linear fct.

Idea: If we have a (transformation) matrix $\mathbf{T} \in \mathbb{R}^{m,m}$ satisfying

$$\|\mathbf{Ty}\|_2 = \|\mathbf{y}\|_2 \quad \forall \mathbf{y} \in \mathbb{R}^m, \tag{3.3.1.3}$$

then $\underset{\mathbf{y}\in\mathbb{R}^n}{\operatorname{argmin}} \|\mathbf{Ay} - \mathbf{b}\|_2 = \underset{\mathbf{y}\in\mathbb{R}^n}{\operatorname{argmin}} \left\|\tilde{\mathbf{A}}\mathbf{y} - \tilde{\mathbf{b}}\right\|_2$, = lsq(A,b)

where $\tilde{\mathbf{A}} = \mathbf{TA}$ and $\tilde{\mathbf{b}} = \mathbf{Tb}$.

3.3.2. Orthogonal Matrices

$\|Tx\|_2 = \|x\|_2 \quad \forall x \in \mathbb{R}^m$ (*)

$x^T y = \frac{1}{2}(\|x+y\|_2^2 - \|x\|_2^2 - \|y\|_2^2)$ [Polarization identity]

$(Tx)^T(Ty) = x^T T^T T y \overset{(*)}{=} x^T y \quad \forall x, y \in \mathbb{R}^m$

$T^T T = I$

Definition 3.3.2.1. Unitary and orthogonal matrices →

- $\mathbf{Q} \in \mathbb{K}^{n,n}$, $n \in \mathbb{N}$, is unitary, if $\mathbf{Q}^{-1} = \mathbf{Q}^H$.
- $\mathbf{Q} \in \mathbb{R}^{n,n}$, $n \in \mathbb{N}$, is orthogonal, if $\mathbf{Q}^{-1} = \mathbf{Q}^T$. $\Leftrightarrow \|Qx\|_2 = \|x\|_2 \ \forall x \in \mathbb{R}^n$

→ Cols/rows form an ONB of $\mathbb{R}^n$

NumCSE@ETHZ

Review questions

▷ Answer Review Questions 3.3.2.3 without aids

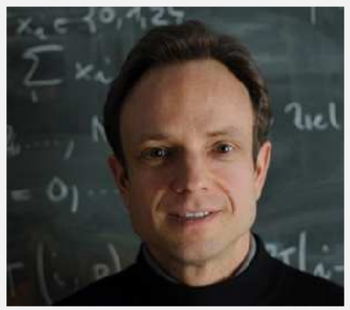
ETH Lectures 401-0663-00L Numerical Methods for Computer Science
401-2673-00L Numerical Methods for CSE

Course Video
Section 3.3.3.1: QR-Decomposition: Theory

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 9, 2021

(C) Seminar für Angewandte Mathematik, ETH Zürich



Tablet notes may have been corrected and/or extended after the recording of the video!

Prerequisites.

- From LA: Gram-Schmidt orthogonalization

Duration: 20 minutes

III. Direct Methods for Linear Least Squares Problems

3.3. Orthogonal Transformation Methods

3.3.3. QR-Decomposition

3.3.3.1. QR-Decomposition: Theory

Recall (from LA): Gram-Schmidt orthonormalization of vectors $\{a^1, \ldots, a^n\} \subset \mathbb{R}^m$, $m \geq n$, linearly independent

1: $\mathbf{q}^1 := \frac{\mathbf{a}^1}{\|\mathbf{a}^1\|_2}$; % 1st output vector
2: **for** $j = 2, \ldots, n$ **do**
{ % Orthogonal projection
3: $\mathbf{q}^j := \mathbf{a}^j$;
4: **for** $\ell = 1, 2, \ldots, j-1$ **do** (GS)
5: { $\mathbf{q}^j \leftarrow \mathbf{q}^j - \langle \mathbf{a}^j, \mathbf{q}^\ell \rangle \mathbf{q}^\ell$; }
6: **if** ($\mathbf{q}^j = \mathbf{0}$) **then** STOP
7: **else** { $\mathbf{q}^j \leftarrow \frac{\mathbf{q}^j}{\|\mathbf{q}^j\|_2}$; }
8: }

$(q^i)^T q^j = \delta_{ij}$

Theorem 3.3.3.2. Span property of G.S. vectors

If $\{\mathbf{a}^1, \ldots, \mathbf{a}^n\} \subset \mathbb{R}^m$ is linearly independent, then Algorithm (GS) computes orthonormal vectors $\mathbf{q}^1, \ldots, \mathbf{q}^n \in \mathbb{R}^m$ satisfying

$$\mathrm{Span}\{\mathbf{q}^1, \ldots, \mathbf{q}^\ell\} = \mathrm{Span}\{\mathbf{a}^1, \ldots, \mathbf{a}^\ell\}, \quad (1.5.1.2)$$

for all $\ell \in \{1, \ldots, n\}$.

(2)

$Q := [q^1, \dots, q^n] \in \mathbb{R}^{m,n}$, $Q^T Q = I_n$

$A := [a^1, \dots, a^n] \in \mathbb{R}^{m,n}$

(1.5.1.2): $\operatorname{Span}\{q^1, \dots, q^\ell\} = \operatorname{Span}\{a^1, \dots, a^\ell\}$ $\forall \ell = 1, \dots, n$

$\Rightarrow \exists t_{ik} \in \mathbb{R}: \quad q^\ell = \sum_{i=1}^{\ell} t_{i\ell} a^i, \quad \ell = 1, \dots, n$

$$\begin{aligned} q^1 &= t_{11} a^1 \\ q^2 &= t_{12} a^1 + t_{22} a^2 \\ q^3 &= t_{13} a^1 + t_{23} a^2 + t_{33} a^3 \\ &\vdots \\ q^n &= t_{1n} a^1 + t_{2n} a^2 + \cdots + t_{nn} a^n . \end{aligned}$$

$(T)_{ij} := \begin{cases} t_{ij}, & \text{if } i \le j \\ 0, & \text{else.} \end{cases}$

$\exists T \in \mathbb{R}^{n,n}$ upper triangular: $Q = AT$, (3.3.3.3)

regular, since $\operatorname{rank}(A) = n = \operatorname{rank}(Q)$

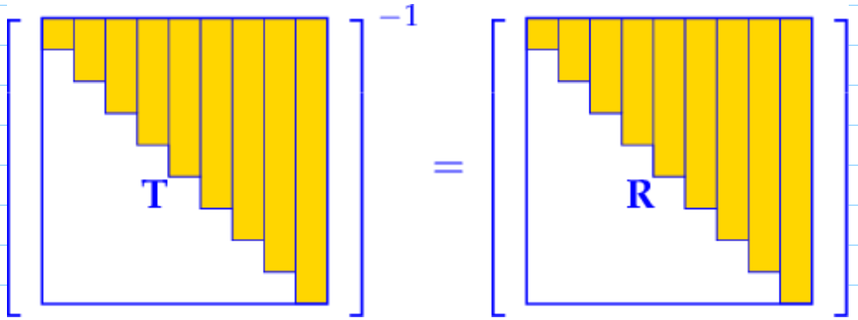
$(t_{ii} \neq 0) \Rightarrow \operatorname{rank}(T) = n$

$\Rightarrow \quad A = QT^{-1} = QR$

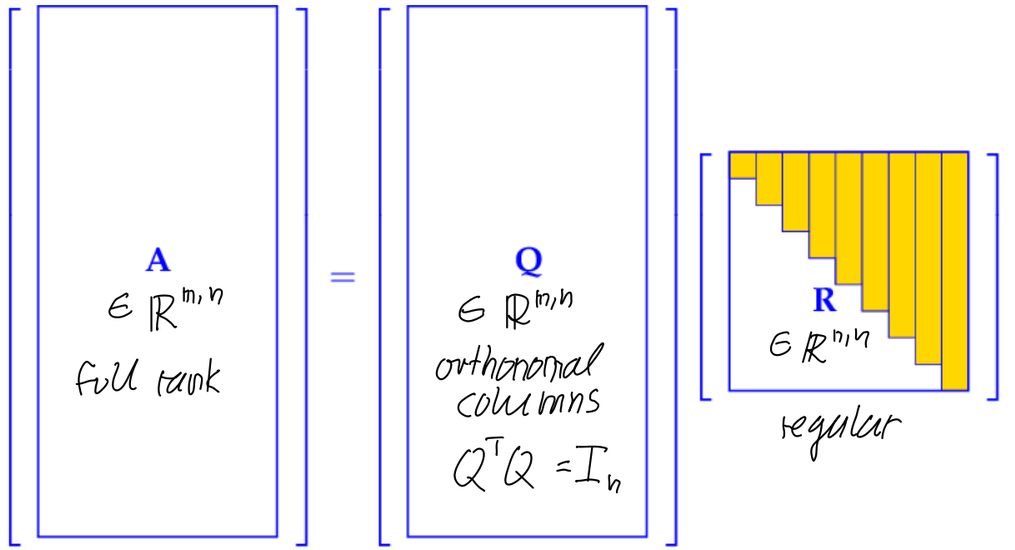
Lemma 1.3.1.9. Group of regular diagonal/triangular matrices

$$\mathbf{A}, \mathbf{B} \begin{cases} \text{diagonal} \\ \text{upper triangular} \\ \text{lower triangular} \end{cases} \Rightarrow \mathbf{AB} \text{ and } \mathbf{A}^{-1} \begin{cases} \text{diagonal} \\ \text{upper triangular} \\ \text{lower triangular} \end{cases}.$$

(assumes that $\mathbf{A}$ is regular)



$\mathbf{A} = \mathbf{QR} \leftrightarrow$

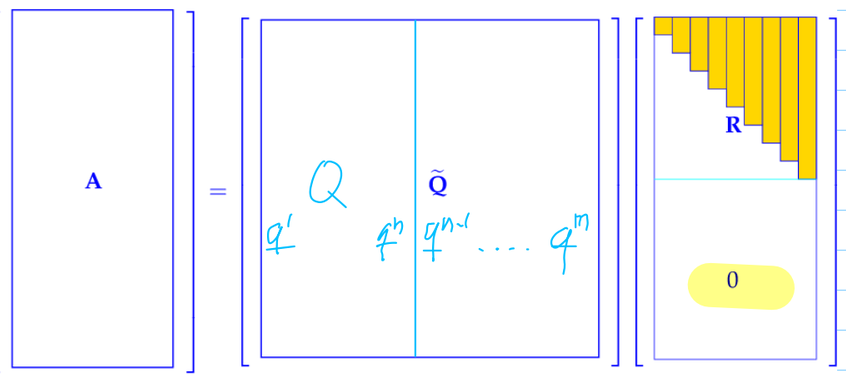


$\hat{=}$ economical / thin QR-decomposition

Full QR-decomposition: extend $\{q^1, \dots, q^n\}$ to an ONB of $\mathbb{R}^m$

$\mathbf{A} = \tilde{\mathbf{Q}} \begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix}$

$\updownarrow$



$\tilde{Q}$ is orthogonal matrix

$\Leftrightarrow \quad \tilde{\mathbf{Q}}^T \mathbf{A} = \begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix}.$

③

Theorem 3.3.3.4. QR-decomposition → ,

For any matrix $\mathbf{A} \in \mathbb{K}^{n,k}$ *with* $\operatorname{rank}(\mathbf{A}) = k$ *there exists*

(i) a unique Matrix $\mathbf{Q}_0 \in \mathbb{R}^{n,k}$ *that satisfies* $\mathbf{Q}_0^{\mathrm{H}}\mathbf{Q}_0 = \mathbf{I}_k$, *and a unique upper triangular Matrix* $\mathbf{R}_0 \in \mathbb{K}^{k,k}$ *with* $(\mathbf{R})_{i,i} > 0$, $i \in \{1,\ldots,k\}$, *such that*

$$\mathbf{A} = \mathbf{Q}_0 \cdot \mathbf{R}_0 \qquad (\textbf{\textit{"economical" QR-decomposition}}) ,$$

(ii) a unitary Matrix $\mathbf{Q} \in \mathbb{K}^{n,n}$ *and a unique upper triangular* $\mathbf{R} \in \mathbb{K}^{n,k}$ *with* $(\mathbf{R})_{i,i} > 0$, $i \in \{1,\ldots,n\}$, *such that*

$$\mathbf{A} = \mathbf{Q} \cdot \mathbf{R} \qquad (\textbf{\textit{full QR-decomposition}}) .$$

If $\mathbb{K} = \mathbb{R}$ *all matrices will be real and* $\mathbf{Q}$ *is then orthogonal.*

▷ Answer review questions 3.3.3.8 !

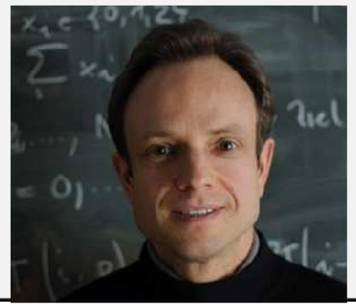
Course Video

Section 3.3.3.2 , Section 3.3.3.4: Computation of QR-Decomposition, QR-Decomposition in EIGEN

Prof. R. Hiptmair, SAM, ETH Zurich

Date: September 23, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- From LA: Orthogonal matrices

Dependencies. Section 3.3.3.1, Section 1.2.1

Duration: 55 minutes

III. Direct Methods for Linear Least Squares Problems

3.3. Orthogonal Transformation Methods

3.3.3. QR-Decomposition

$A \in \mathbb{R}^{m,n}$: $A = QR$, $Q \in \mathbb{R}^{m,m}$ orthogonal, $R \in \mathbb{R}^{m,n}$ upper triangular

3.3.3.2. Computation of QR-Decompositions

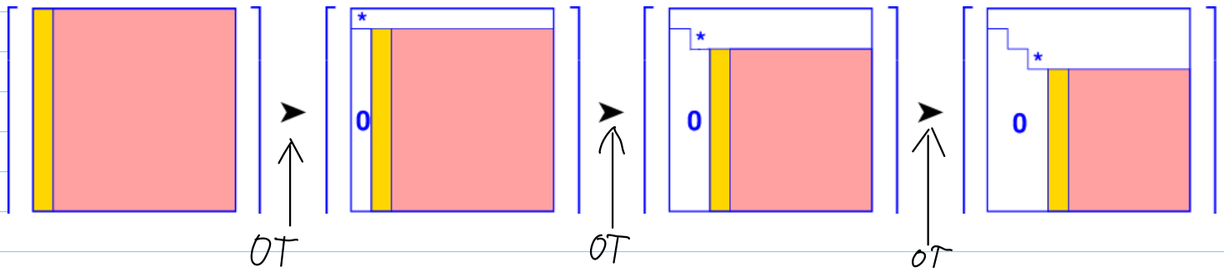
LA : Gram-Schmidt : unstable, affected by roundoff

Idea : similar to GE

Idea: find simple unitary/orthogonal (row) transformations rendering certain matrix elements zero:

$$\mathbf{Q}\begin{pmatrix} \blacksquare \end{pmatrix} = \begin{bmatrix} \blacksquare \\ 0 \end{bmatrix} \quad \text{with} \quad \mathbf{Q}^H = \mathbf{Q}^{-1} .$$

②



Corollary 3.3.3.9. Composition of orthogonal transformations

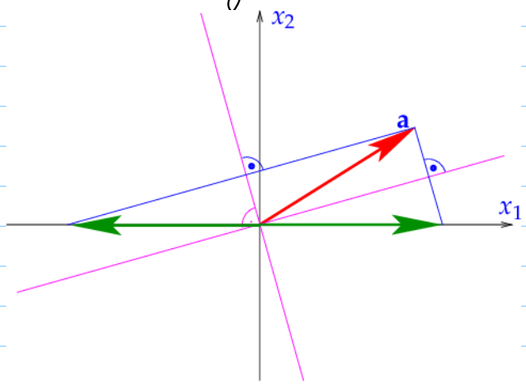
The product of two orthogonal/unitary matrices of the same size is again orthogonal/unitary.

Ex 3.3.3.9 : 2D annihilating OT

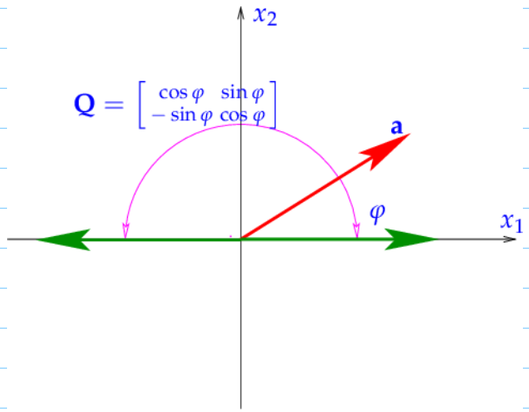
$$\underline{a} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \xrightarrow{Q} \begin{bmatrix} \pm\|a\| \\ 0 \end{bmatrix} \parallel \underline{e}_1$$

$$Q\,[\underline{a}, \underline{b}] = \begin{bmatrix} * & * \\ 0 & * \end{bmatrix}$$

Four ways :

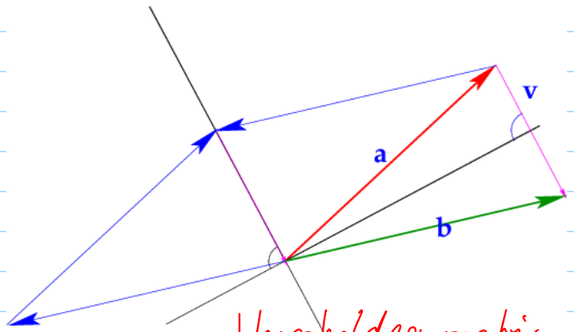


reflections at angle bisector,



rotations turning **a** onto x_1-axis.

§ 3.3.3.10 : Householder reflections



Householder matrix

$\underline{a}, \underline{b} \in \mathbb{R}^m$, $\|\underline{a}\|_2 = \|\underline{b}\|_2$

$\underline{v} := \underline{b} - \underline{a} \perp \underline{b} + \underline{a}$

$$\underline{b} = \underline{a} - \underline{v} = \underline{a} - \underline{v} \cdot 2\frac{\underline{v}^T\underline{a}}{\underline{v}^T\underline{v}} \quad (\underline{v}^T\underline{a} = \tfrac{1}{2}\underline{v}^T\underline{v})$$

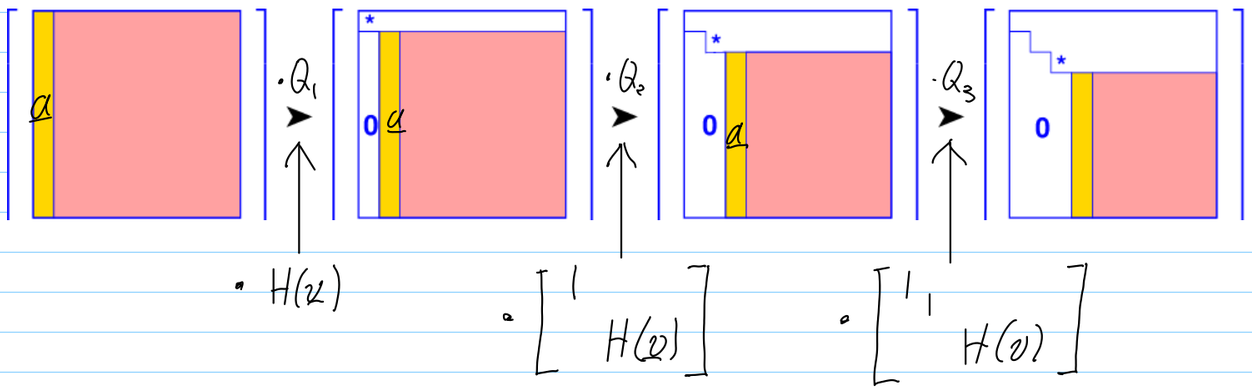
$$= H(\underline{v})\,\underline{a}$$

$$H(\underline{v}) = \left(\underline{I} - 2\frac{\underline{v}\underline{v}^T}{\underline{v}^T\underline{v}}\right)$$

↳ orthogonal, symmetric

Special : $\underline{v} = \underline{a} + \operatorname{sgn}(a_1)\|\underline{a}\|_2\,\underline{e}_1$

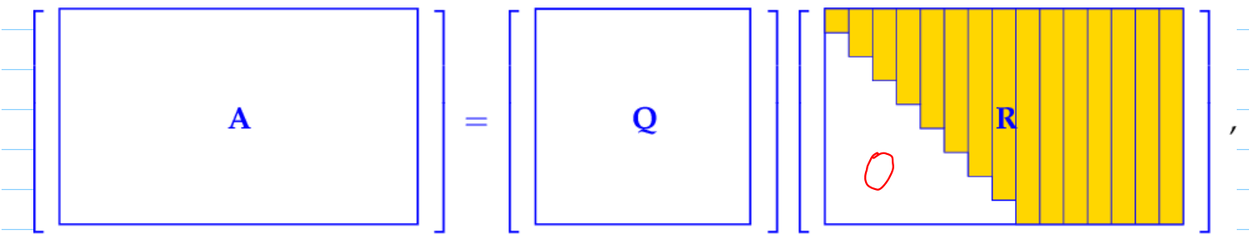
↑ avoids cancellation, Rem. 3.3.3.13



$$\mathbf{Q}_{n-1}\mathbf{Q}_{n-2}\cdots\cdot\mathbf{Q}_1\mathbf{A}=\mathbf{R}\,,$$

QR-decomposition (QR-factorization) of $\mathbf{A}\in\mathbb{C}^{n,n}$: $\mathbf{A}=\mathbf{QR}$, $\mathbf{Q}:=\mathbf{Q}_1^H\cdot\cdot\cdot\cdot\mathbf{Q}_{n-1}^H$ orthogonal matrix, $\mathbf{R}$ upper triangular matrix.

Rem.: 3.3.3.12 : HH-refl. applicable to any matrix



$$\mathbf{A}=\mathbf{QR}\quad,\quad \mathbf{Q}\in\mathbb{R}^{m,m}\,,\quad \mathbf{R}\in\mathbb{R}^{m,n}\,,$$

Rem.: 3.3.20 : Encode HH-hfs.

In practice: only $Q\times$ vector needed

→ sufficient to store Householder matrices

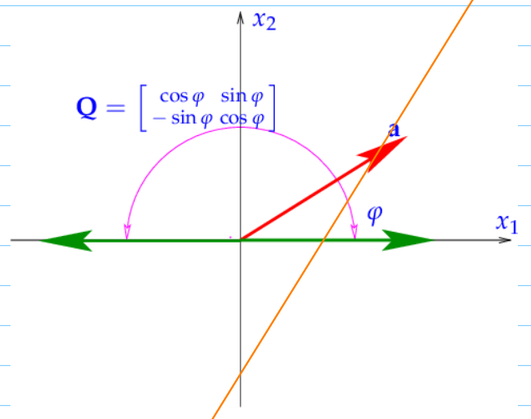
HHM: $H(v)=I-2vv^T\in\mathbb{R}^{m}$, $v\in\mathbb{R}^m$, $\|v\|_2=1$

→ store only unit vector $v\in\mathbb{R}^m$

"Compute QR-dec." ≙ compute R & encode Q

NumCSE@ETHZ

§ 3.3.3.14 : Givens rotations



rotations turning **a** onto x_1-axis.

Idea: 2D rotations attack two entries of a only

$$\begin{bmatrix}\gamma & \sigma\\ -\sigma & \gamma\end{bmatrix}\begin{bmatrix}a_1\\ a_k\end{bmatrix}=\begin{bmatrix}*\\ 0\end{bmatrix}$$

$$\mathbf{G}_{1k}(a_1,a_k)\mathbf{a}:=\begin{bmatrix}\gamma & \cdots & \sigma & \cdots & 0\\ \vdots & 1 & & & \vdots\\ -\sigma & \cdots & \gamma & \cdots & 0\\ \vdots & & \vdots & 1 & \vdots\\ 0 & \cdots & 0 & \cdots & 1\end{bmatrix}\begin{bmatrix}a_1\\ \vdots\\ a_k\\ \vdots\\ a_n\end{bmatrix}=\begin{bmatrix}a_1^{(1)}\\ \vdots\\ 0\\ \vdots\\ a_n\end{bmatrix}\quad\text{, if}\quad \begin{aligned}\gamma&=\frac{a_1}{\sqrt{|a_1|^2+|a_k|^2}}\,,\\ \sigma&=\frac{a_k}{\sqrt{|a_1|^2+|a_k|^2}}\,.\end{aligned}\qquad(3.3.3.16)$$

orthogonal

≙ unchanged

$$\mathbf{a}:=\begin{bmatrix}a_1\\ a_2\\ \vdots\\ \vdots\\ a_n\end{bmatrix}\xrightarrow{\mathbf{G}_{12}(a_1,a_2)}\begin{bmatrix}a_1^{(1)}\\ 0\\ a_3\\ \vdots\\ a_n\end{bmatrix}\xrightarrow{\mathbf{G}_{13}(a_1^{(1)},a_3)}\begin{bmatrix}a_1^{(2)}\\ 0\\ 0\\ a_4\\ \vdots\\ a_n\end{bmatrix}\xrightarrow{\mathbf{G}_{14}(a_1^{(2)},a_4)}\cdots\xrightarrow{\mathbf{G}_{1n}(a_1^{(n-2)},a_n)}\begin{bmatrix}a_1^{(n-1)}\\ 0\\ \vdots\\ \vdots\\ 0\end{bmatrix}\qquad(3.3.3.18)$$

(4)

Rem 3.3.3.20 : Storing Givens rtfs.

↳ storing two indices + angle

for $\mathbf{G} = \begin{bmatrix} \gamma & \sigma \\ -\sigma & \gamma \end{bmatrix}$ $\Rightarrow$ store $\rho := \begin{cases} \operatorname{sign}(\sigma) & \text{, if } \gamma = 0 \text{,} \\ \sqrt{2}\operatorname{sign}(\gamma)\sigma & \text{, if } |\sigma| < |\gamma| \text{,} \\ \frac{1}{2}\sqrt{2}\operatorname{sign}(\sigma)/\gamma & \text{, if } |\sigma| \geq |\gamma| \text{,} \end{cases}$ (3.3.3.22)

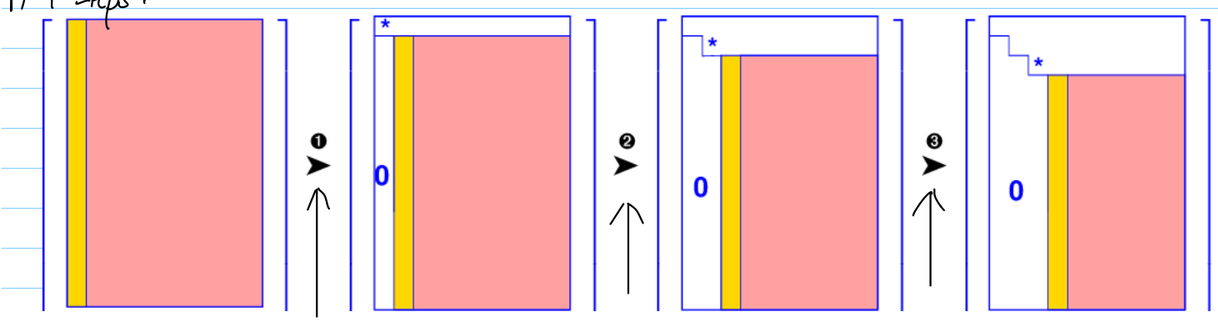
which means $\begin{cases} \rho = \pm 1 & \Rightarrow \quad \gamma = 0 \text{,} \quad \sigma = \pm 1 \text{,} \\ |\rho| < 1 & \Rightarrow \quad \sigma = \frac{1}{2}\sqrt{2}\rho \text{,} \quad \gamma = \sqrt{1-\sigma^2} \text{,} \\ |\rho| > 1 & \Rightarrow \quad \gamma = \frac{1}{2}\sqrt{2}/\rho \text{,} \quad \sigma = \sqrt{1-\gamma^2} \text{.} \end{cases}$ (3.3.3.23)

↑ no cancellation

§ 3.3.3.22 : Computational cost

Householder-based QR-dec. of $A \in \mathbb{R}^{m,n}$, $m \geq n$

$n-1$ steps:



#ops: $\sim m(n-1)$ $(m-1)(n-2)$ $(m-2)(n-3)$

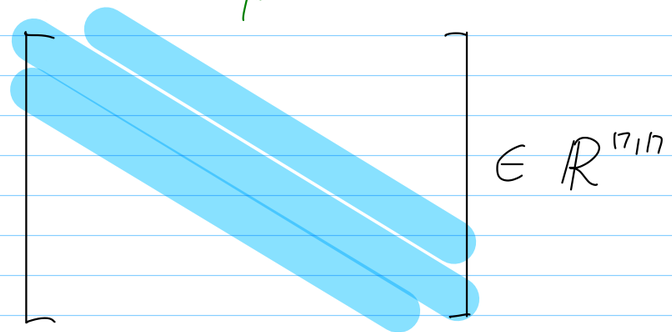
Note: $H(v)X = X - v\underbrace{(v^T X)}_{\in \mathbb{R}^k}$: cost $= O(mk)$

$X \in \mathbb{R}^{m,k}$

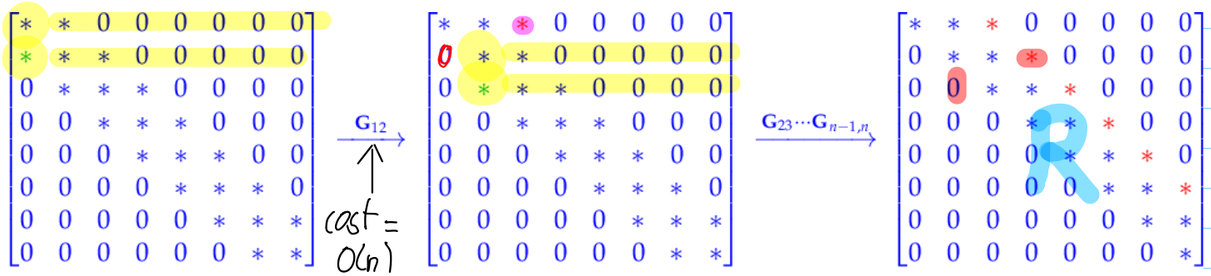
$$\text{cost}(\text{R-factor of } \mathbf{A} \text{ by Householder trf.}) = \sum_{k=1}^{n-1} 2(m-k+1)(n-k) \qquad (3.3.3.25)$$
$$= O(mn^2) \text{ for } m,n \to \infty \text{.}$$

Rem 3.3.3.26 : QR-dec. of banded matrices

Ex. : tridiagonal matrices



Givens-QR for tridiag. matrix $\in \mathbb{R}^{n,n}$



$*$ ≙ entry set to zero by Givens rotation, $*$ ≙ new non-zero entry

still tridiagonal

In fact : cost $O(1)$ for single rotation, because ≤ 3 non-zero entries in every row of the matrices

In general : $\operatorname{bw}(A) = \operatorname{bw}(R)$

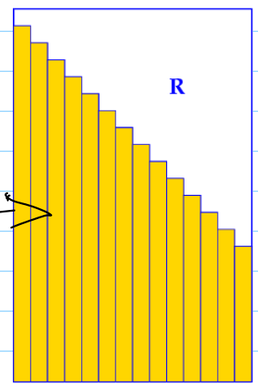
3.3.3.4. QR-Decomposition in Eigen

C++-code 3.3.3.34: QR-decompositions in EIGEN

```
2  # include <Eigen/QR>
3
4  // Computation of full QR-decomposition (3.3.3.1),
5  // dense matrices built for both QR-factors (expensive!)
6  std::pair<MatrixXd,MatrixXd> qr_decomp_full(const MatrixXd& A) {
7    Eigen::HouseholderQR<MatrixXd> qr(A);
8    MatrixXd Q = qr.householderQ(); //
9    MatrixXd R = qr.matrixQR().template triangularView<Eigen::Upper>();
10   return std::pair<MatrixXd,MatrixXd>(Q,R);
11 }
12
13 // Computation of economical QR-decomposition (3.3.3.1),
14 // dense matrix built for Q-factor (possibly expensive!)
15 std::pair<MatrixXd,MatrixXd> qr_decomp_eco(const MatrixXd& A) {
16   using index_t = MatrixXd::Index;
17   const index_t m = A.rows(),n = A.cols();
18   Eigen::HouseholderQR<MatrixXd> qr(A);
19   MatrixXd Q = (qr.householderQ()*MatrixXd::Identity(m,n)); //
20   MatrixXd R = qr.matrixQR().block(0,0,n,n).template triangularView<Eigen::Upper>();
       //
21   return std::pair<MatrixXd,MatrixXd>(Q,R);
22 }
```

multiply HHMs

compressed QR-dec.: HHMs stored in the lower triangular part



contains v-vector for HHMs $H(v)$

Cost $= O(mn^2)$ for $m, n \to \infty$

Review Questions

Please tackle Review Questions 3.3.3.28

ETH Lecture 401-0663-00L Numerical Methods for CSE

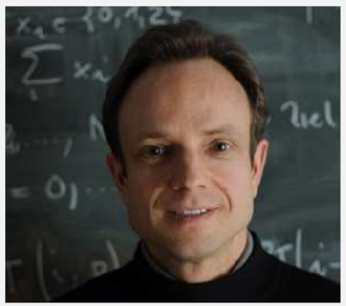
Course Video

Section 3.3.4: QR-Based Solver for Linear Least Squares Problems

Prof. R. Hiptmair, SAM, ETH Zurich

Date: September 23, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 3.1.1, Section 3.3.3.2, Section 3.3.3.4

Duration: 16 minutes

III. Direct Methods for Linear Least Squares Problems

3.3. Orthogonal Transformation Methods

3.3.4. QR-Based Solver for Linear Least Squares Problems

Given: $A \in \mathbb{R}^{m,n}$, $\operatorname{rank}(A) = n$, $\underline{b} \in \mathbb{R}^m$

$\underline{x} \in \mathbb{R}^n$: $\|A\underline{x} - \underline{b}\|_2 \longrightarrow \min$

QR-dec.: $A = QR$, $Q \in \mathbb{R}^{m,m}$ orthogonal, $Q^TQ = I$

$$\begin{aligned} \|A\underline{x} - \underline{b}\|_2 &= \|QR\underline{x} - \underline{b}\|_2 \\ &= \|Q(R\underline{x} - Q^T\underline{b})\|_2 \\ &= \|R\underline{x} - \tilde{\underline{b}}\|_2 \quad \hat{=} \text{ triangular LSQ} \end{aligned}$$

$\tilde{\underline{b}} := Q^T\underline{b}$

Note: $\operatorname{rank}(A) = n \Rightarrow \operatorname{rank}(R) = n$

(2)

$$\|\mathbf{Ax}-\mathbf{b}\|_2 = \left\|\mathbf{Q}(\mathbf{Rx}-\mathbf{Q}^\top\mathbf{b})\right\|_2 = \left\|\mathbf{Rx}-\tilde{\mathbf{b}}\right\|_2 \quad , \quad \tilde{\mathbf{b}} := \mathbf{Q}^\top\mathbf{b} .$$

R_0 regular

$$\|\mathbf{Ax}-\mathbf{b}\|_2 \to \min \quad \Leftrightarrow \quad \left\| \begin{bmatrix} \mathbf{R}_0 \\ \mathbf{0} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} - \begin{bmatrix} \tilde{b}_1 \\ \vdots \\ \tilde{b}_m \end{bmatrix} \right\|_2 \to \min .$$

$$\mathbf{x} = \begin{bmatrix} \mathbf{R}_0 \\ \mathbf{0} \end{bmatrix}^{-1} \begin{bmatrix} \tilde{b}_1 \\ \vdots \\ \tilde{b}_n \end{bmatrix} , \quad \text{with residual} \quad \mathbf{r} = \mathbf{Q} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \tilde{b}_{n+1} \\ \vdots \\ \tilde{b}_m \end{bmatrix} .$$

↑ LSQ sol !

We need only $Q^T b := \tilde{b}$

In Eigen : ($A \in \mathbb{R}^{m,n}$, $m \geq n$)

C++ code 3.3.4.2: EIGEN's built-in QR-based linear least squares solver

```
// Solving a full-rank least squares problem ||Ax - b||_2 -> min in EIGEN
double lsqsolve_eigen(const MatrixXd& A, const VectorXd& b,
                      VectorXd& x) {
  x = A.householderQr().solve(b);
  return ((A*x-b).norm());
}
```

← Two stage approach

cost = $O(n^2 m)$

- $Q^T b$: Application of $n-1$ HHMs to $\underline{b}$ ⇒ cost = $O(mn)$
- Solve LSE $R_0 x = (\tilde{b})_{1:n}$ ⇒ cost = $O(n^2)$ [backward elimination]

Rem.: QR-reuse option, same as for LU

Comparisons :

Normal equations vs. orthogonal transformations method

Superior numerical stability (→ Def. 1.5.5.19) of orthogonal transformations methods:

- Use orthogonal transformations methods for least squares problems (3.1.3.7), whenever $\mathbf{A} \in \mathbb{R}^{m,n}$ *dense* and n small.

SVD/QR-factorization cannot exploit sparsity:

- Use normal equations in the expanded form (3.2.0.8)/(3.2.0.9), when $\mathbf{A} \in \mathbb{R}^{m,n}$ sparse (→ Notion 2.7.0.1) and m, n big.

NumCSE@ETHZ

Review Questions:

Try to answer review questions 3.3.4.8 without aids.

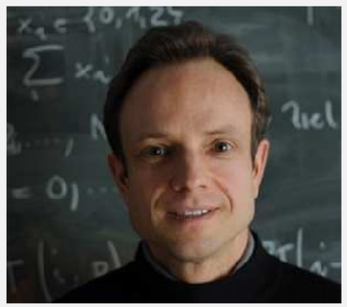
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 3.3.5: Modification Techniques for QR-Decomposition

Prof. R. Hiptmair, SAM, ETH Zurich

Date: September 23, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 3.3.3.2, Section 2.6

Duration: 45 minutes

III. Direct Methods for Linear Least Squares Problems

3.3. Orthogonal Transformation Methods

3.3.5. Modification Techniques for QR-Decomposition

Given: • $A \in \mathbb{R}^{m,n}$, $m \geq n$, $\operatorname{rank}(A) = n$

• QR-dec (full) $A = QR$, $Q \in \mathbb{R}^{m,m}$ orthogonal, $R \in \mathbb{R}^{m,n}$ upper triangular

↑ stored as a product of OTs *

Task: Efficiently compute $\tilde{A} = \tilde{Q}\tilde{R}$, $\tilde{A} \hat{=}$ "slight modification" of A

* product of $n-1$ Householder matrices

NumCSE@ETHZ

3.3.5.1 : Rank-1-Modification

$$\tilde{A} = A + \underline{u}\underline{v}^T, \quad \underline{u} \in \mathbb{R}^m, \underline{v} \in \mathbb{R}^n$$

$$\tilde{A} = QR + \underline{u}\underline{v}^T = Q(R + \underbrace{Q^T\underline{u}}_{=:\underline{w}}\underline{v}^T)$$

Step ①: Compute $\underline{w} := Q^T\underline{u}$

→ Cost = $O(mn)$ for $m, n \to \infty$

Step ②: $\underline{w} \to \begin{bmatrix} * \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^m$ by successive Givens rotations

$$\mathbf{w} = \begin{bmatrix} * \\ * \\ \vdots \\ * \\ * \\ * \\ * \end{bmatrix} \xrightarrow{\mathbf{G}_{m-1,m}} \begin{bmatrix} * \\ * \\ \vdots \\ * \\ * \\ * \\ 0 \end{bmatrix} \xrightarrow{\mathbf{G}_{m-2,m-1}} \begin{bmatrix} * \\ * \\ \vdots \\ * \\ * \\ 0 \\ 0 \end{bmatrix} \xrightarrow{\mathbf{G}_{m-3,m-2}} \cdots \xrightarrow{\mathbf{G}_{12}} \begin{bmatrix} * \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \|w\|_2 \underline{e}_1$$

→ Cost = $O(m)$

Action on R

$$\mathbf{R} = \begin{bmatrix} * & * & \cdots & * & * & * & * \\ 0 & * & \cdots & * & * & * & * \\ \vdots & & \ddots & & & & \vdots \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \\ 0 & \cdots & 0 & 0 & 0 & 0 & * \\ 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\ 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\ \vdots & & & & & & \vdots \\ 0 & \cdots & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{\mathbf{G}_{n,n+1}} \begin{bmatrix} * & * & \cdots & * & * & * & * \\ 0 & * & \cdots & * & * & * & * \\ \vdots & & \ddots & & & & \vdots \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \\ 0 & \cdots & 0 & 0 & 0 & 0 & * \\ 0 & \cdots & 0 & 0 & 0 & 0 & * \\ 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\ \vdots & & & & & & \vdots \\ 0 & \cdots & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{\mathbf{G}_{n-1,n}}$$

$$\to \begin{bmatrix} * & * & \cdots & * & * & * & * \\ 0 & * & \cdots & * & * & * & * \\ \vdots & & \ddots & & & & \vdots \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \\ 0 & \cdots & 0 & 0 & 0 & 0 & * \\ 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\ \vdots & & & & & & \vdots \\ 0 & \cdots & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{\mathbf{G}_{n-2,n-1}} \cdots \xrightarrow{\mathbf{G}_{1,2}} \begin{bmatrix} * & * & \cdots & * & * & * & * \\ * & * & \cdots & * & * & * & * \\ & & \ddots & & & & \vdots \\ 0 & \cdots & * & * & * & * & * \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \\ 0 & \cdots & 0 & 0 & 0 & 0 & * \\ 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\ \vdots & & & & & & \vdots \\ 0 & \cdots & 0 & 0 & 0 & 0 & 0 \end{bmatrix} =: \mathbf{R}_1$$

$R_1 + \|w\|_2 \underline{e}_1 \underline{v}^T$ will have the same structure as R_1

↑ attack with Givens rotations

③

$$\mathbf{R}_1 + \|\mathbf{w}\|_2 \mathbf{e}_1 \mathbf{v}^\top = \begin{bmatrix} * & * & \cdots & * & * & * & * \\ * & * & \cdots & * & * & * & * \\ & & \ddots & & & & \\ 0 & \cdots & * & * & * & * & * \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \end{bmatrix} \xrightarrow{\mathbf{G}_{12}} \begin{bmatrix} * & * & \cdots & * & * & * & * \\ 0 & * & \cdots & * & * & * & * \\ & & \ddots & & & & \\ 0 & \cdots & * & * & * & * & * \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \end{bmatrix} \xrightarrow{\mathbf{G}_{23}} \cdots$$

$$\xrightarrow{\mathbf{G}_{n-2,n-1}} \begin{bmatrix} * & * & \cdots & * & * & * & * \\ 0 & * & \cdots & * & * & * & * \\ & & \ddots & & & & \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \end{bmatrix} \xrightarrow{\mathbf{G}_{n-1,n}} \begin{bmatrix} * & * & \cdots & * & * & * & * \\ 0 & * & \cdots & * & * & * & * \\ & & \ddots & & & & \\ 0 & \cdots & 0 & * & * & * & * \\ 0 & \cdots & 0 & 0 & * & * & * \\ 0 & \cdots & 0 & 0 & 0 & * & * \\ 0 & \cdots & 0 & 0 & 0 & 0 & * \end{bmatrix} =: \tilde{\mathbf{R}} .$$

$$\underbrace{(\mathbf{G}_{n-1,n}\mathbf{G}_{n-2,n-1}\cdots\cdots\cdots\mathbf{G}_{23}\mathbf{G}_{12})}_{=: Q_2}(\mathbf{R}_1 + \|\mathbf{w}\|_2 \mathbf{e}_1 \mathbf{v}^\top) = \tilde{\mathbf{R}} \quad \text{(upper triangular!)} . \qquad (3.3.5.3)$$

$Q_2 \rightarrow$ cost $\approx O(n^2)$ for $n \to \infty$

$$\Rightarrow \quad \tilde{A} = Q\, Q_1^T Q_2^T \tilde{R}$$

Total cost $= O(mn + n^2)$ for large m, n

$\ll O(mn^2)$, if both m, n large

3.3.5.2. Adding a column

$$\mathbf{A} \in \mathbb{R}^{m,n} \longrightarrow \tilde{\mathbf{A}} = [\mathbf{a}_1, \ldots, \mathbf{a}_{k-1}, \mathbf{v}, \mathbf{a}_k, \ldots, \mathbf{a}_n] \in \mathbb{R}^{m,n+1} , \quad \mathbf{v} \in \mathbb{R}^m , \quad \mathbf{a}_j := (\mathbf{A})_{:,j} . \qquad (3.3.5.4)$$

Assume: $m \geq n+1$

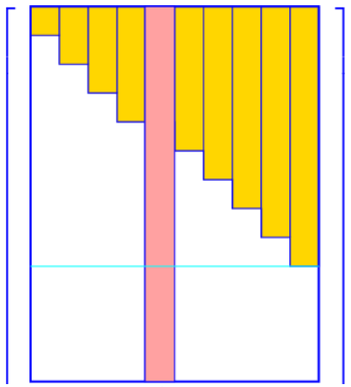
full QR-dec.

$$\mathbf{A} = \mathbf{QR} \;\Leftrightarrow\; \mathbf{Q}^\top \mathbf{A} = \left[\mathbf{Q}^\top \mathbf{a}_1, \ldots, \mathbf{Q}^\top \mathbf{a}_n\right] = \mathbf{R} =: \begin{bmatrix} \mathbf{R}_0 \\ \mathbf{O} \end{bmatrix} , \quad \mathbf{R}_0 \in \mathbb{R}^{n,n} ,$$

NumCSE@ETHZ

④

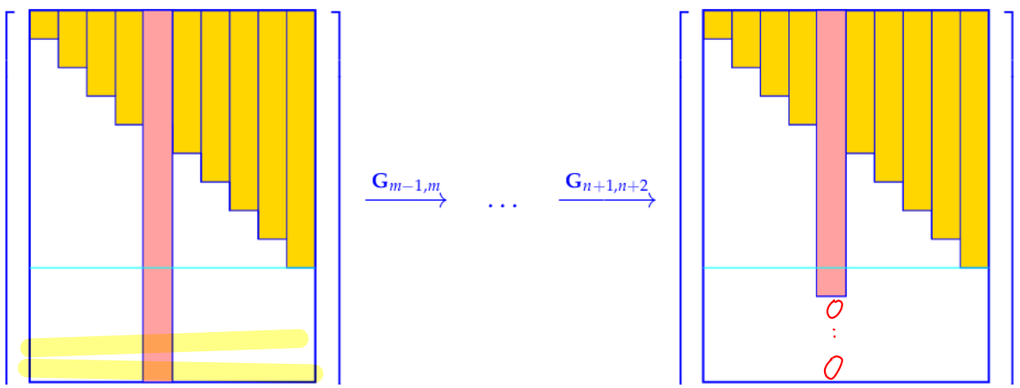
$$\mathbf{Q}^\top \tilde{\mathbf{A}} = \left[\mathbf{Q}^\top \mathbf{a}_1, \ldots, \mathbf{Q}^\top \mathbf{a}_{k-1}, \mathbf{Q}^\top \mathbf{v}, \mathbf{Q}^\top \mathbf{a}_k, \ldots, \mathbf{Q}^\top \mathbf{a}_n\right] = \quad =: \mathbf{W} \in \mathbb{R}^{m,n+1},$$



↑ k-th column : $Q^T v \in \mathbb{R}^m$

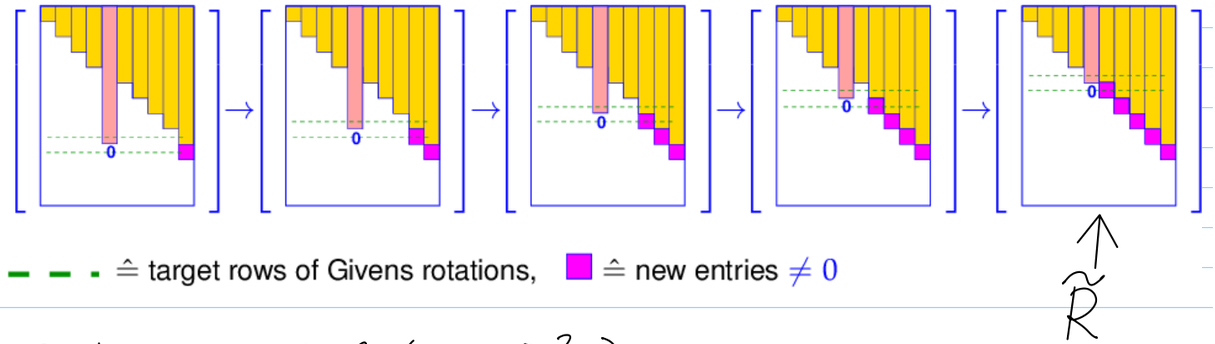
① Compute $w := Q^T v$: Cost = $O(mn)$

② $m-n-1$ Givens rot. ⇒ last $m-n-1$ rows of $W \to 0$

$\mathbf{W} =$ 

$=: \mathbf{T}$.

③ $T \to$ upper triangular $\hat{R} \in \mathbb{R}^{m,n+1}$ by $n-k$ Givens rots. (bottom up)



- - - - ≙ target rows of Givens rotations, ■ ≙ new entries $\neq 0$

↑ $\tilde{R}$

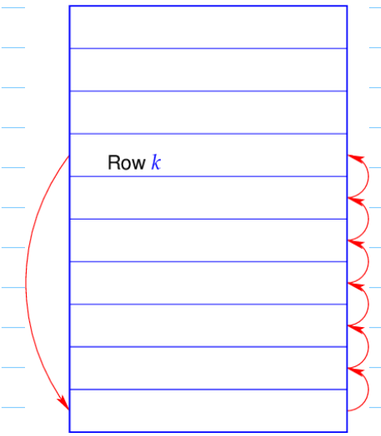
Cost = $O((n-k)^2)$

Total cost = $O(mn + (n-k)^2)$ for $m,n \to \infty$

3.3.5.3. Adding a row

$$\mathbf{A} \in \mathbb{R}^{m,n} \quad \mapsto \quad \tilde{\mathbf{A}} = \begin{bmatrix} (\mathbf{A})_{1,:} \\ \vdots \\ (\mathbf{A})_{k-1,:} \\ \mathbf{v}^T \\ (\mathbf{A})_{k,:} \\ \vdots \\ (\mathbf{A})_{m,:} \end{bmatrix}, \quad \text{with given} \quad \mathbf{v} \in \mathbb{R}^n . \qquad (3.3.5.5)$$

① Move new row to the bottom



by a cyclic permutation of rows of $\tilde{A}$

$\hat{=}$ orth. trf. $P \in \mathbb{R}^{m+1,m+1}$

Just reindexing : no cost

$$T = \begin{bmatrix} * & \cdots & & \cdots & * \\ 0 & * & & & \vdots \\ \vdots & 0 & \ddots & & \\ \vdots & \vdots & & * & \vdots \\ 0 & 0 & & 0 & * \\ 0 & 0 & & 0 & * \\ * & \cdots & \cdots & * & * \end{bmatrix} \xrightarrow{G_{1,m+1}} \begin{bmatrix} * & \cdots & & \cdots & * \\ 0 & * & & & \vdots \\ \vdots & 0 & \ddots & & \\ \vdots & \vdots & & * & \vdots \\ 0 & 0 & & 0 & * \\ 0 & 0 & & 0 & * \\ 0 & * & \cdots & * & * \end{bmatrix} \xrightarrow{G_{2,m+1}} \cdots$$

$$\cdots \xrightarrow{G_{m-1,m+1}} \begin{bmatrix} * & \cdots & & \cdots & * \\ 0 & * & & & \vdots \\ \vdots & 0 & \ddots & & \\ \vdots & \vdots & & * & \vdots \\ 0 & 0 & & 0 & * \\ 0 & 0 & & 0 & * \\ 0 & 0 & & 0 & * \end{bmatrix} \xrightarrow{G_{m,m+1}} \begin{bmatrix} * & \cdots & & \cdots & * \\ 0 & * & & & \vdots \\ \vdots & 0 & \ddots & & \\ \vdots & \vdots & & * & \vdots \\ 0 & 0 & & 0 & * \\ 0 & 0 & & 0 & * \\ 0 & \cdots & & 0 & 0 \end{bmatrix} := \tilde{R} \quad (3.3.5.6)$$

Cost = $O(n^2)$

$$Q^T A = R$$

$$P\tilde{A} = \begin{bmatrix} A \\ v^T \end{bmatrix} \blacktriangleright \begin{bmatrix} Q^T & 0 \\ 0 & 1 \end{bmatrix} P\tilde{A} = \begin{bmatrix} R \\ v^T \end{bmatrix} = \begin{bmatrix} R \\ v^T \end{bmatrix} =: T \in \mathbb{R}^{m+1,n}.$$

$\uparrow$ $\in \mathbb{R}^{m+1,m+1}$ orthogonal

→ to be annihil. later

Idea: Use n suitable Givens rotations

NumCSE@ETHZ

Review questions:

▷ Please answer review questions 3.3.5.7

ETH Lecture 401-0663-00L Numerical Methods for CSE

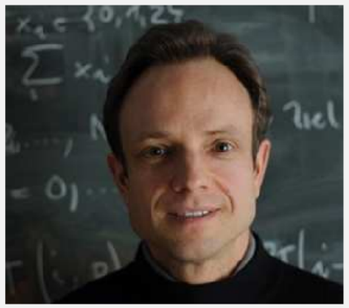
Course Video

Section 3.4.1: Singular Value Decomposition: Definition and Theory

Prof. R. Hiptmair, SAM, ETH Zurich

Date: September 29, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- From linear algebra: kernel, image space, and rank of a matrix

Dependencies. Section 1.1, Section 3.3.2

Duration: 24 minutes

III. Direct Methods for Linear Least Squares Problems

3.4. Singular Value Decomposition (SVD)

3.4.1. SVD: Definition and Theory

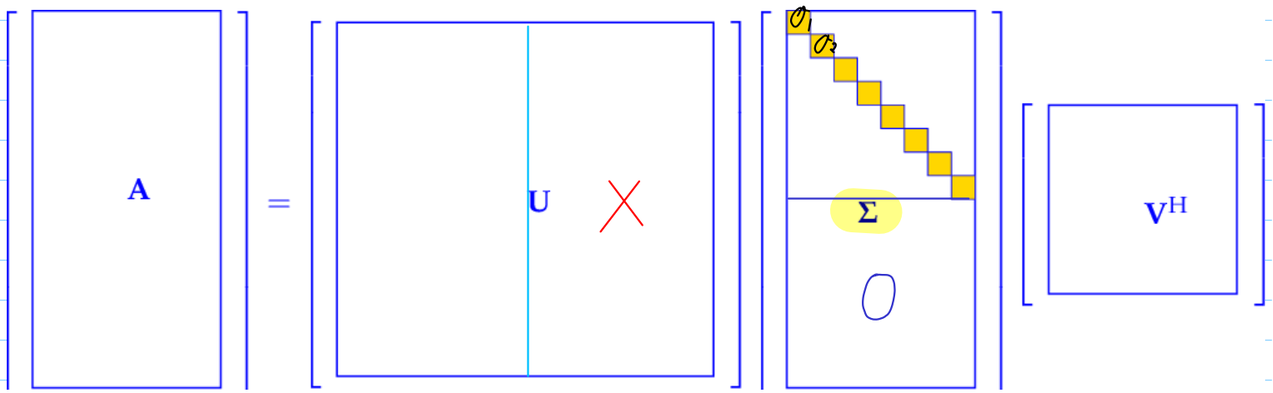
Theorem 3.4.1.1. Singular value decomposition (full)

For any $\mathbf{A} \in \mathbb{K}^{m,n}$ *there are* unitary/orthogonal *matrices* $\mathbf{U} \in \mathbb{K}^{m,m}$, $\mathbf{V} \in \mathbb{K}^{n,n}$ *and a (generalized) diagonal* (*) *matrix* $\mathbf{\Sigma} = \operatorname{diag}(\sigma_1, \ldots, \sigma_p) \in \mathbb{R}^{m,n}$, $p := \min\{m,n\}$, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$ *such that*

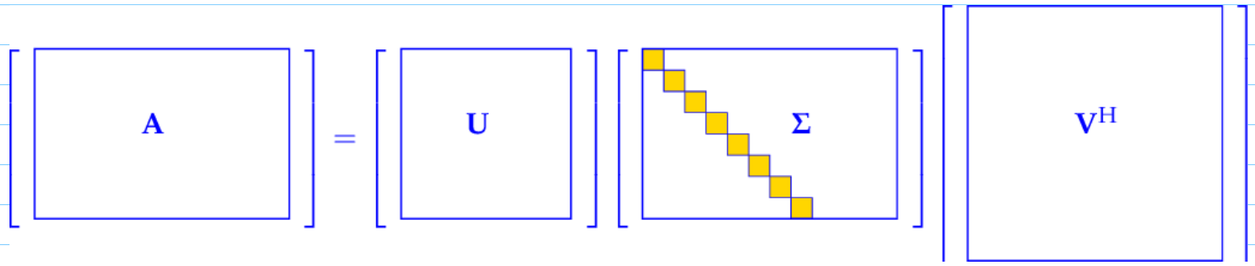
singular values

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{H}} .$$

$m \geq n$:



(2)



Lemma 3.4.1.6.

The squares σ_i^2 of the non-zero singular values of $\mathbf{A}$ are the non-zero eigenvalues of $\mathbf{A}^H\mathbf{A}$, $\mathbf{A}\mathbf{A}^H$ with associated eigenvectors $(\mathbf{V})_{:,1},\ldots,(\mathbf{V})_{:,p}$, $(\mathbf{U})_{:,1},\ldots,(\mathbf{U})_{:,p}$, respectively.

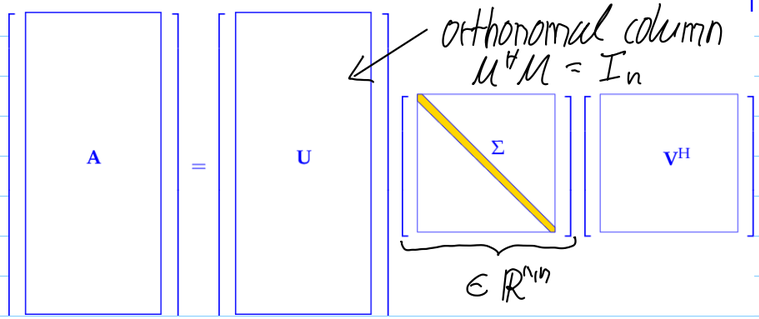
$$A^HA \overset{\text{SVD}}{=} (U\Sigma V^H)^H(U\Sigma V^H) = V\Sigma^H\underbrace{U^HU}_{=I}\Sigma V^H = V\underbrace{\Sigma^H\Sigma}_{\text{similar matrices}}V^H$$

$\Rightarrow$ same eigenvalues

$m \geq n$: $\Sigma^H\Sigma = \operatorname{diag}(\sigma_1^2,\ldots,\sigma_n^2)$

$m > n$: $m-n$ right cols. of U irrelevant

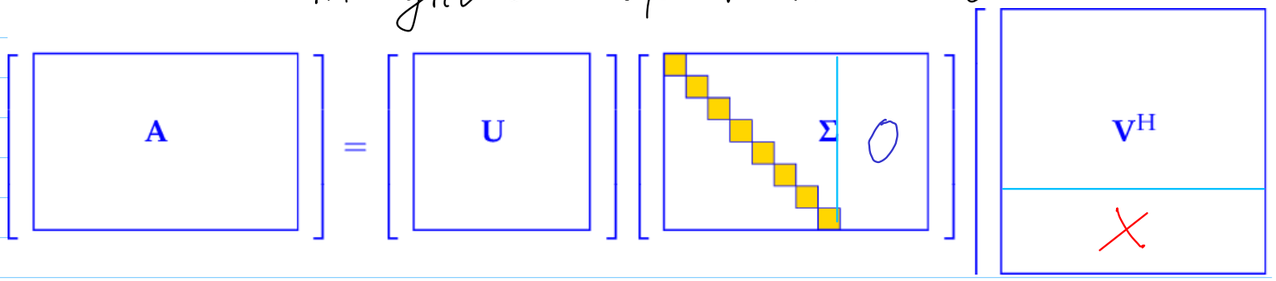
▷ economical SVD:



II

$n > m$:

$n-m$ right cols. of V irrelevant



Rem.: 3.4.1.7: SVD = additive rank-1 decomposition

Eco-SVD: $p := \min\{m,n\}$

$$A = U\Sigma V^H = \sum_{j=1}^{p}\sigma_j \underbrace{u_jv_j^H}_{m\times n \text{ tensor product, rank}=1}$$

$U = [u_i]_{i=1}^m$, $V = [v_i]_{i=1}^n$

Assume: $\sigma_{r+1} = \ldots = \sigma_p = 0$

$$A = U\Sigma V^H = \sum_{j=1}^{r}\sigma_j u_jv_j^H$$

$$Av_\ell = \sum_{j=1}^{r}\sigma_j u_j \underbrace{v_j^Hv_\ell}_{=\delta_{j\ell}} = \begin{cases}\sigma_\ell u_\ell, & \ell \leq r\\ 0, & \ell > r\end{cases}$$

▷ $\mathcal{R}(A) = \operatorname{Span}\{u_1,\ldots,u_r\} \Rightarrow \operatorname{rank}(A) = r$

③

▷ $\mathcal{N}(A) = \text{Span}\{v_{r+1}, \dots, v_n\}$

Lemma 3.4.1.13. SVD and rank of a matrix →

If, for some $1 \le r \le p := \min\{m,n\}$, the singular values of $\mathbf{A} \in \mathbb{K}^{m,n}$ satisfy $\sigma_1 \ge \cdots \ge \sigma_r > \sigma_{r+1} = \cdots \sigma_p = 0$, then

- $\operatorname{rank}(\mathbf{A}) = r$ *(no. of non-zero singular values)*,
- $\mathcal{N}(\mathbf{A}) = \operatorname{Span}\{(\mathbf{V})_{:,r+1}, \dots, (\mathbf{V})_{:,n}\}$,
- $\mathcal{R}(\mathbf{A}) = \operatorname{Span}\{(\mathbf{U})_{:,1}, \dots, (\mathbf{U})_{:,r}\}$.

$\mathcal{N}(A^H) = ?$

$\mathcal{R}(A^H) = ?$

Why $\operatorname{rank}(A) = n - \dim \mathcal{N}(A)$?

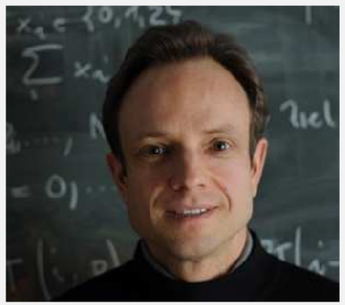
Review questions

Try to answer the review questions 3.4.1.15 without aids.

ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 3.4.2: SVD in EIGEN

Prof. R. Hiptmair, SAM, ETH Zurich



Date: September 29, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich

Dependencies. Section 3.4.1, Section 1.2.1

Duration: 15 minutes

III. Direct Methods for Linear Least Squares Problems

3.4. Singular Value Decomposition (SVD)

3.4.2. SVD in Eigen

Theorem 3.4.1.1. Singular value decomposition $\rightarrow$,

For any $\mathbf{A} \in \mathbb{K}^{m,n}$ *there are* unitary/orthogonal *matrices* $\mathbf{U} \in \mathbb{K}^{m,m}$, $\mathbf{V} \in \mathbb{K}^{n,n}$ *and a (generalized) diagonal* $^{(*)}$ *matrix* $\mathbf{\Sigma} = \operatorname{diag}(\sigma_1, \ldots, \sigma_p) \in \mathbb{R}^{m,n}$, $p := \min\{m,n\}$, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$ *such that*

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{H}} .$$

(2)

C++-code 3.4.2.1: Computing SVDs in EIGEN

```
#include <Eigen/SVD>

// Computation of (full) SVD A = UΣV^H → Thm. 3.4.1.1
// SVD factors are returned as dense matrices in natural order
std::tuple<MatrixXd,MatrixXd,MatrixXd> svd_full(const MatrixXd& A) {
  Eigen::JacobiSVD<MatrixXd> svd(A, Eigen::ComputeFullU | Eigen::ComputeFullV);
  MatrixXd U = svd.matrixU(); // get unitary (square) matrix U
  MatrixXd V = svd.matrixV(); // get unitary (square) matrix V
  VectorXd sv = svd.singularValues(); // get singular values as vector
  MatrixXd Sigma = MatrixXd::Zero(A.rows(), A.cols());
  const unsigned p = sv.size(); // no. of singular values
  Sigma.block(0,0,p,p) = sv.asDiagonal(); // set diagonal block of Σ
  return std::tuple<MatrixXd,MatrixXd,MatrixXd>(U,Sigma,V);
}

// Computation of economical (thin) SVD A = UΣV^H, see (3.4.1.5)
// SVD factors are returned as dense matrices in natural order
std::tuple<MatrixXd,MatrixXd,MatrixXd> svd_eco(const MatrixXd& A) {
  Eigen::JacobiSVD<MatrixXd> svd(A, Eigen::ComputeThinU | Eigen::ComputeThinV);
  MatrixXd U = svd.matrixU(); // get matrix U with orthonormal columns
  MatrixXd V = svd.matrixV(); // get matrix V with orthonormal columns
  VectorXd sv = svd.singularValues(); // get singular values as vector
  MatrixXd Sigma = sv.asDiagonal(); // build diagonal matrix Σ
  return std::tuple<MatrixXd,MatrixXd,MatrixXd>(U,Sigma,V);
}
```

sorted in descending order

$$\text{Cost}(\text{eco-SVD of } A \in \mathbb{K}^{m,n}) = O(\min\{m,n\}^2 \cdot \max\{m,n\})$$

for $m,n \to \infty$

$$m = n : \quad \text{cost}(\text{SVD}) = O(n^3)$$

Ex. 3.4.2.3 : SVD-based computation of rank

Lemma 3.4.1.13. SVD and rank of a matrix →

If, for some $1 \le r \le p := \min\{m,n\}$, the singular values of $\mathbf{A} \in \mathbb{K}^{m,n}$ satisfy $\sigma_1 \ge \cdots \ge \sigma_r > \sigma_{r+1} = \cdots \sigma_p = 0$, then

- $\operatorname{rank}(\mathbf{A}) = r$ *(no. of non-zero singular values)*,
- $\mathcal{N}(\mathbf{A}) = \operatorname{Span}\{(\mathbf{V})_{:,r+1}, \ldots, (\mathbf{V})_{:,n}\}$,
- $\mathcal{R}(\mathbf{A}) = \operatorname{Span}\{(\mathbf{U})_{:,1}, \ldots, (\mathbf{U})_{:,r}\}$.

∇ Sing. vals. σ_j are affected by roundoff

⇒ no testing $\sigma_j == 0.0$

Instead: test for relative smallness

C++-code 3.4.2.5: Computing rank of a matrix through SVD

```
// Computation of the numerical rank of a non-zero matrix by means of
// singular value decomposition, cf. (3.4.2.4).
MatrixXd::Index rank_by_svd(const MatrixXd &A, double tol = EPS) {
  if (A.norm() == 0) return MatrixXd::Index(0);
  Eigen::JacobiSVD<MatrixXd> svd(A);
  const VectorXd sv = svd.singularValues(); // Get sorted singular values as
    vector
  MatrixXd::Index n = sv.size();
  MatrixXd::Index r = 0;
  // Test relative size of singular values
  while ((r<n) && (sv(r) >= sv(0)*tol)) r++;
  return r;
}
```

"numerical rank"

Largest sing. val.

③

C++-code 3.4.2.6: Using `rank()` in EIGEN

```
// Computation of the numerical rank of a matrix by means of SVD
MatrixXd::Index rank_eigen(const MatrixXd &A, double tol = EPS) {
  return A.jacobiSvd().setThreshold(tol).rank();
}
```

Ex 3.4.2.7 : Computation of $\mathcal{N}(A)$ / $\mathcal{R}(A)$

↳ provide ON basis

Lemma 3.4.1.13. SVD and rank of a matrix →

If, for some $1 \le r \le p := \min\{m,n\}$, the singular values of $\mathbf{A} \in \mathbb{K}^{m,n}$ satisfy $\sigma_1 \ge \cdots \ge \sigma_r > \sigma_{r+1} = \cdots \sigma_p = 0$, then

- $\operatorname{rank}(\mathbf{A}) = r$ *(no. of non-zero singular values)*,
- $\mathcal{N}(\mathbf{A}) = \operatorname{Span}\{(\mathbf{V})_{:,r+1}, \ldots, (\mathbf{V})_{:,n}\}$
- $\mathcal{R}(\mathbf{A}) = \operatorname{Span}\{(\mathbf{U})_{:,1}, \ldots, (\mathbf{U})_{:,r}\}$.

} ONBs

use numerical rank

C++-code 3.4.2.8: ONB of $\mathcal{N}(\mathbf{A})$ through SVD

```
// Computation of an ONB of the kernel of a matrix
MatrixXd nullspace(const MatrixXd &A, double tol = EPS) {
  using index_t = MatrixXd::Index;
  Eigen::JacobiSVD<MatrixXd> svd(A,Eigen::ComputeFullV);
  index_t r = svd.setThreshold(tol).rank();
  // Rightmost columns of V provide ONB of N(A)
  MatrixXd Z = svd.matrixV().rightCols(A.cols()-r);
  return Z;
}
```

$= n$

C++-code 3.4.2.9: ONB of $\mathcal{R}(\mathbf{A})$ through SVD

```
// Computation of an ONB of the image space of a matrix
MatrixXd rangespace(const MatrixXd &A, double tol = EPS) {
  using index_t = MatrixXd::Index;
  Eigen::JacobiSVD<MatrixXd> svd(A,~~Eigen::ComputeFullV~~);
  index_t r = svd.setThreshold(tol).rank();
  // r left columns of U provide ONB of R(A)
  return svd.matrixU().leftCols(r);
}
```

II

Correct option ?

NumCSE@ETHZ

Review questions :

Examine the review questions 3.4.2.10

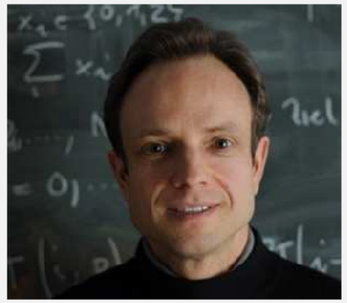
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 3.4.3: Solving General Least-Squares Problems by SVD

Prof. R. Hiptmair, SAM, ETH Zurich

Date: September 29, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 3.1.1, Section 3.1.3, Section 3.4.1, Section 1.2.1, Section 3.1.2

Duration: 24 minutes

III. Direct Methods for Linear Least Squares Problems

3.4. Singular Value Decomposition (SVD)

3.4.3. Solving General Least-Squares Problems by SVD

(General linear) least squares problem:

$r := \operatorname{rank}(A) \le \min\{m,n\}$

given: $\mathbf{A} \in \mathbb{R}^{m,n}$, $m,n \in \mathbb{N}$, $\mathbf{b} \in \mathbb{R}^m$, → not necessarily full rank

find: $\mathbf{x}^+ \in \mathbb{R}^n$ such that

$x^+ \in \mathrm{lsq}(A,b) \Leftrightarrow$ (i) $\left\|\mathbf{A}\mathbf{x}^+ - \mathbf{b}\right\|_2 = \min\{\|\mathbf{A}\mathbf{y} - \mathbf{b}\|_2 : \mathbf{y} \in \mathbb{R}^n\}$, (3.1.3.7)

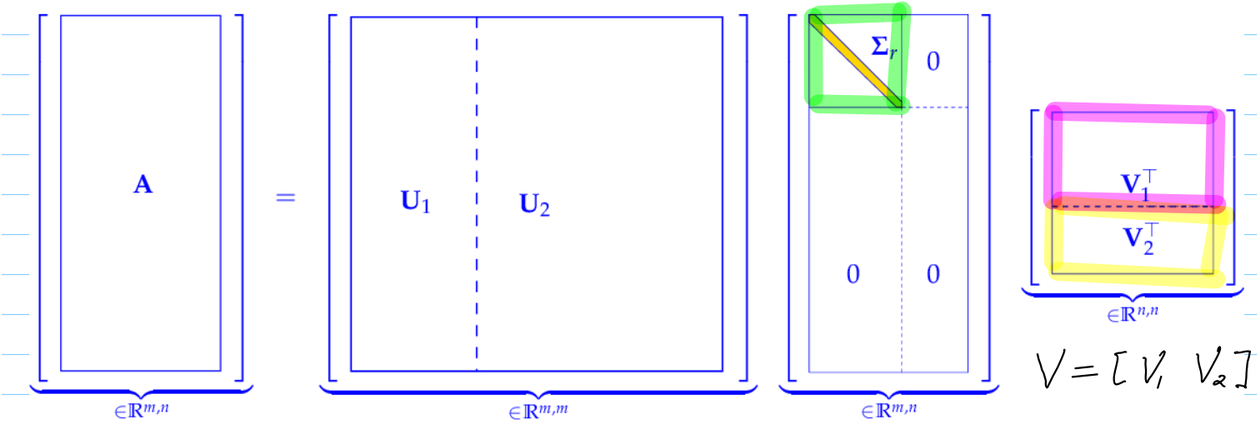
(ii) $\left\|\mathbf{x}^+\right\|_2$ is minimal under the condition (i).

$x^+ = A^+ b$, $A^+ \hat{=}$ pseudo-inverse

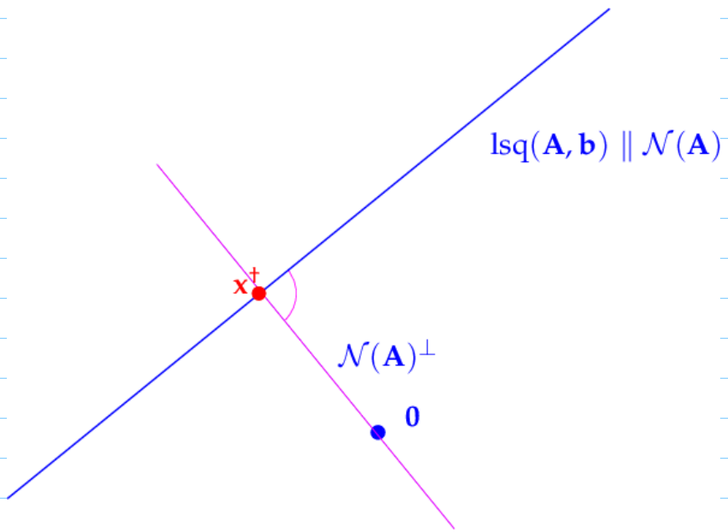
Tool : economical SVD

② $A \in \mathbb{R}^{m,n}$, full SVD

$$\mathbf{A} = [\mathbf{U}_1 \quad \mathbf{U}_2] \begin{bmatrix} \mathbf{\Sigma}_r & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^\top \\ \mathbf{V}_2^\top \end{bmatrix}$$



$$\mathbf{V} = [\mathbf{V}_1 \; \mathbf{V}_2] \tag{3.4.3.1}$$



$x^+ \in \mathcal{N}(A)^\perp$

↑ can be retrieved from SVD

Lemma 3.4.1.13. SVD and rank of a matrix →

If, for some $1 \le r \le p := \min\{m,n\}$, the singular values of $\mathbf{A} \in \mathbb{K}^{m,n}$ satisfy $\sigma_1 \ge \cdots \ge \sigma_r > \sigma_{r+1} = \cdots \sigma_p = 0$, then

- $\operatorname{rank}(\mathbf{A}) = r$ *(no. of non-zero singular values)*,
- $\mathcal{N}(\mathbf{A}) = \operatorname{Span}\{(\mathbf{V})_{:,r+1}, \ldots, (\mathbf{V})_{:,n}\}$,
- $\mathcal{R}(\mathbf{A}) = \operatorname{Span}\{(\mathbf{U})_{:,1}, \ldots, (\mathbf{U})_{:,r}\}$.

▷ [$V \in \mathbb{R}^{n,n}$ orthogonal]:

$\mathcal{N}(A)^\perp = \operatorname{Span}\{v_1, \ldots, v_r\} = \mathcal{R}(V_1) = \{V_1 y, y \in \mathbb{R}^r\}$

$x^+ = V_1 y^+$ for some $y \in \mathbb{R}^r$ (1)

Recall: $\operatorname{lsq}(A,b) = \{x \in \mathbb{R}^n : A^T A x = A^T b\}$ (2)

Theorem 3.1.2.1. Obtaining least squares solutions by solving normal equations

The vector $\mathbf{x} \in \mathbb{R}^n$ is a least squares solution (→ Def. 3.1.1.1) of the linear system of equations $\mathbf{Ax} = \mathbf{b}$, $\mathbf{A} \in \mathbb{R}^{m,n}$, $\mathbf{b} \in \mathbb{R}^m$, if and only if *it solves the normal equations (NEQ)*

$$\mathbf{A}^\top \mathbf{A} \mathbf{x} = \mathbf{A}^\top \mathbf{b} \,. \tag{3.1.2.2}$$

(1) & (2) $\quad A^T A V_1 y^+ = A^T b$

Trick: $\quad V_1^T A^T A V_1 y^+ = V_1^T A^T b$

Plug in SVD: $A = U \Sigma V^T$

③

$$V_1^T V \Sigma^T \underbrace{U^T U}_{=I} \Sigma V^T V_1 y^+ = V_1^T V \Sigma^T U^T b$$

$$\underbrace{V_1^T [V_1, V_2]}_{[I\ 0]} \underbrace{\begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix}}_{\begin{bmatrix} \Sigma_r^2 & 0 \\ 0 & 0 \end{bmatrix}} \underbrace{\begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} V_1}_{\begin{bmatrix} I \\ 0 \end{bmatrix}} y^+$$

$$= V_1^T [V_1, V_2] \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} U_1^T \\ U_2^T \end{bmatrix} b$$

$$\Rightarrow \quad \Sigma_r^2 y^+ = \Sigma_r U_1^T b$$

$\Sigma_r \in \mathbb{R}^{r,r}$ regular $\Rightarrow$ $y^+ = \Sigma_r^{-1} U_1^T b$

economical SVD sufficient

$$x^+ = V_1 \Sigma_r^{-1} U_1^T b$$

C++-code 3.4.3.13: Computing generalized solution of $\mathbf{Ax} = \mathbf{b}$ via SVD

```
#include <Eigen/SVD>

VectorXd lsqsvd(const MatrixXd &A, const VectorXd &b) {
  // Compute economical SVD, compare Code 3.4.2.1
  Eigen::JacobiSVD<MatrixXd> svd(A, Eigen::ComputeThinU | Eigen::ComputeThinV);
  VectorXd sv = svd.singularValues();
  unsigned int r = svd.rank(); // Numerical rank, default tolerance
  MatrixXd U = svd.matrixU(), V = svd.matrixV();
  // x^+ = V_1 Σ_r^{-1} U_1^H b, see
  // (3.4.3.5)
  return V.leftCols(r) * (sv.head(r).cwiseInverse().asDiagonal() *
                          (U.leftCols(r).adjoint() * b));
}
```

just a scaling

C++-code 3.4.3.14: Computing generalized solution of $\mathbf{Ax} = \mathbf{b}$ via SVD

```
VectorXd lsqsvd_eigen(const MatrixXd &A, const VectorXd &b) {
  Eigen::JacobiSVD<MatrixXd> svd(A, Eigen::ComputeThinU | Eigen::ComputeThinV);
  return svd.solve(b);
}
```

$$x^+ = V_1 \Sigma_r^{-1} U_1^T b = A^+ b$$

Theorem 3.4.3.16. Pseudoinverse and SVD

If $\mathbf{A} \in \mathbb{K}^{m,n}$ has the SVD decomposition $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H$ partitioned as in (3.4.3.1), then its Moore-Penrose pseudoinverse (→ Thm. 3.1.3.6) is given by $\mathbf{A}^+ = \mathbf{V}_1\mathbf{\Sigma}_r^{-1}\mathbf{U}_1^H$.

Review questions

Please answer review questions 3.4.3.17

ETH Lecture 401-0663-00L Numerical Methods for CSE

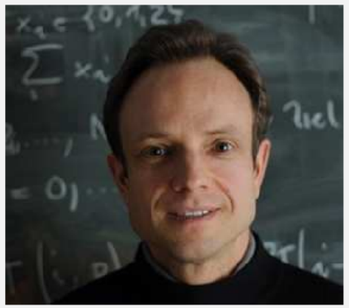
Course Video

Section 3.4.4.1: Norm-Constrained Extrema of Quadratic Forms

Prof. R. Hiptmair, SAM, ETH Zurich

Date: September 29, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 3.4.1, Section 3.4.2

Duration: 21 minutes

III. Direct Methods for Linear Least Squares Problems

3.4. Singular Value Decomposition (SVD)

3.4.4. SVD-Based Optimization and Approximation

3.4.4.1. Norm-Constrained Extrema of Quadratic Forms

given $\mathbf{A} \in \mathbb{K}^{m,n}$, $m \geq n$, find $\mathbf{x} \in \mathbb{K}^n$, $\|\mathbf{x}\|_2 = 1$, $\|\mathbf{Ax}\|_2^2 \to \max$,

↳ Eco-SVD: $A = U \Sigma V^T$

$$\|Ax\|_2 = \|U \Sigma V^T x\|_2 = \|\Sigma V^T x\|_2 \to \max$$

(cols. are orthonormal: "Pythagoras")

$$y = V^T x \Rightarrow \|Ax\|_2^2 = \sum_{k=1}^{n} \sigma_k^2 y_k^2 \longrightarrow \max$$

orthogonal ⇒ under constraint $\sum_{k=1}^{n} y_k^2 = 1$

(2)

Solution $y = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = e_1 \Rightarrow x = V e_1 = (V)_{:,1}$

Maximal value : $\|Ax\|_2 = \sigma_1$

Euclidean matrix norm $\|A\|_2 = \max_{\|x\|=1} \|Ax\| = \sigma_1$

↳ can be computed via SVD

Solution for $\|Ax\|_2 \longrightarrow \min$, $\|x\|_2 = 1$

$x = (V)_{:,n}$, $\|Ax\|_2 = \sigma_n$

Ex. 3.4.4.5 : Fitting of hyperplanes

↳ in Hesse normal form

$\mathcal{H} := \{x \in \mathbb{R}^d : c + n^T x = 0\}$ for $\|n\|_2 = 1$, $c \in \mathbb{R}$

Goal: given the point coordinate vectors $\mathbf{y}_1, \ldots, \mathbf{y}_m \in \mathbb{R}^d$, $m > d$, find $\mathcal{H} \leftrightarrow \{c \in \mathbb{R}, \mathbf{n} \in \mathbb{R}^d, \|\mathbf{n}\|_2 = 1\}$, such that

$$\sum_{j=1}^{m} \operatorname{dist}(\mathcal{H}, \mathbf{y}_j)^2 = \sum_{j=1}^{m} |c + \mathbf{n}^T \mathbf{y}_j|^2 \to \min . \qquad (3.4.4.8)$$

(3.4.4.8) $\Leftrightarrow$ $\left\| \underbrace{\begin{bmatrix} 1 & y_{1,1} & \cdots & y_{1,d} \\ 1 & y_{2,1} & \cdots & y_{2,d} \\ \vdots & \vdots & & \vdots \\ 1 & y_{m,1} & \cdots & y_{m,d} \end{bmatrix}}_{=:\mathbf{A}} \underbrace{\begin{bmatrix} c \\ n_1 \\ \vdots \\ n_d \end{bmatrix}}_{=:\mathbf{x}} \right\|_2 \to \min$ under the constraint $\|\mathbf{n}\|_2 = 1$.

New: c not subject to constraint

(Preliminary) idea : c : $(Ax)_2 = 0$

Trick : QR-decomposition of A

$$\mathbf{A} := \begin{bmatrix} 1 & y_{1,1} & \cdots & y_{1,d} \\ 1 & y_{2,1} & \cdots & y_{2,d} \\ \vdots & \vdots & & \vdots \\ 1 & y_{m,1} & \cdots & y_{m,d} \end{bmatrix} = \mathbf{QR} \quad , \quad \mathbf{R} := \begin{bmatrix} r_{11} & r_{12} & \cdots & \cdots & r_{1,d+1} \\ 0 & r_{22} & \cdots & \cdots & r_{2,d+1} \\ \vdots & & \ddots & & \vdots \\ 0 & & & & r_{d+1,d+1} \\ 0 & \cdots & & \cdots & 0 \\ \vdots & & & & \vdots \\ 0 & \cdots & & \cdots & 0 \end{bmatrix} \in \mathbb{R}^{m,d+1} .$$

Q orthogonal

$$\|Ax\|_2 = \|QRx\|_2 = \|Rx\| \longrightarrow \min$$

$$\|\mathbf{Ax}\|_2 \to \min \Leftrightarrow \|\mathbf{Rx}\|_2 = \left\| \begin{bmatrix} r_{11} & r_{12} & \cdots & \cdots & r_{1,d+1} \\ 0 & r_{22} & \cdots & \cdots & r_{2,d+1} \\ \vdots & & \ddots & & \vdots \\ 0 & & & & r_{d+1,d+1} \\ 0 & \cdots & & \cdots & 0 \\ \vdots & & & & \vdots \\ 0 & \cdots & & \cdots & 0 \end{bmatrix} \begin{bmatrix} c \\ n_1 \\ \vdots \\ n_d \end{bmatrix} \right\|_2 \to \min . \qquad (3.4.4.9)$$

($\tilde{A}$)

(I) $n \in \mathbb{R}^d$, $\|n\| = 1$: $\|\tilde{A} n\|_2 \longrightarrow \min$

"in standard form"

(II) $c := -\frac{1}{r_{11}} \sum_{\ell=1}^{d} n_\ell r_{1\ell} \Rightarrow (Ax)_1 = 0$

③

C++-code 3.4.4.11: (Generalized) distance fitting of a hyperplane: solution of (3.4.4.12)

```
// Fitting of a hyperplane, returning Hesse normal form
void clsq(const MatrixXd& A, const unsigned dim,
          double& c, VectorXd& n) {
  unsigned p = A.cols(), m = A.rows();
  if (p < dim + 1) { cerr << "not enough unknowns\n"; return; }
  if (m < dim)  { cerr << "not enough equations\n"; return; }

  m = std::min(m, p);
  // First step:  orthogonal transformation, see Code 3.3.4.1
  MatrixXd R = A.householderQr().matrixQR().template triangularView<Eigen::Upper>();
  // compute matrix V from SVD composition of R, solve (3.4.4.10)
  MatrixXd V = R.block(p - dim, p - dim, m + dim - p, dim)
                .jacobiSvd(Eigen::ComputeFullV).matrixV();
  n = V.col(dim - 1);

  MatrixXd R_topleft = R.topLeftCorner(p - dim, p - dim);
  c = -(R_topleft.template triangularView<Eigen::Upper>()
        .solve(R.block(0, p - dim, p - dim, dim)) * n)(0);
}
```

$\tilde{A}$

$A \in \mathbb{R}^{m,p}$: solves $\|A\begin{bmatrix} c \\ \underline{n} \end{bmatrix}\|_2 \longrightarrow \min$

$\underline{n} \in \mathbb{R}^d, \|n\|_2 = 1$

Review questions :

Please look at Review Questions 3.4.4.13

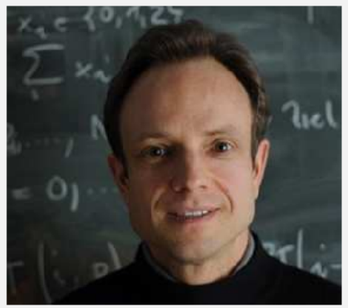
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 3.4.4.2: Best Low-Rank Approximation

Prof. R. Hiptmair, SAM, ETH Zurich

Date: September 29, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.
- Matrix norms

Dependencies. Section 3.3.2, Section 3.4.1, Section 3.4.2

Duration: 25 minutes

III. Direct Methods for Linear Least Squares Problems

3.4. Singular Value Decomposition (SVD)

3.4.4. SVD-Based Optimization and Approximation

3.4.4.2. Best Low-Rank Approximation

Context: Matrix compression = approximation by data-sparse matrix

- sparse matrices
- low-rank matrices

$r := \operatorname{rank}(A) \ll \min\{m,n\}$ for $A \in \mathbb{R}^{m,n}$

If $\operatorname{rank}(A) = r$:

[by SVD] $\Rightarrow$ $A = \sum_{\ell=1}^{r} \sigma_\ell (U)_{:,\ell} (V)_{:,\ell}^T$

Storage $\approx r(m+n) \ll m \cdot n$

(2)

Theorem 3.4.4.18. Best low rank approximation →

Let $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{H}}$ be the SVD of $\mathbf{A} \in \mathbb{K}^{m,n}$ (→ Thm. 3.4.1.1). For $1 \le k \le \operatorname{rank}(\mathbf{A})$ set

$$\mathbf{A}_k := \underbrace{\mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^{\mathrm{H}}}_{\text{truncated SVD}} = \sum_{\ell=1}^{k} \sigma_\ell (\mathbf{U})_{:,\ell}(\mathbf{V})_{:,\ell}^{\mathrm{H}} \quad \text{with} \quad \begin{aligned} \mathbf{U}_k &:= \left[(\mathbf{U})_{:,1},\ldots,(\mathbf{U})_{:,k}\right] \in \mathbb{K}^{m,k}, \\ \mathbf{V}_k &:= \left[(\mathbf{V})_{:,1},\ldots,(\mathbf{V})_{:,k}\right] \in \mathbb{K}^{n,k}, \\ \mathbf{\Sigma}_k &:= \operatorname{diag}(\sigma_1,\ldots,\sigma_k) \in \mathbb{K}^{k,k}. \end{aligned}$$

Then, for $\|\cdot\| = \|\cdot\|_F$ and $\|\cdot\| = \|\cdot\|_2$, holds true

$$\|\mathbf{A} - \mathbf{A}_k\| \le \|\mathbf{A} - \mathbf{F}\| \quad \forall \mathbf{F} \in \mathcal{R}_k(m,n),$$

that is, $\mathbf{A}_k$ is the rank-k best approximation of $\mathbf{A}$ in the matrix norms $\|\cdot\|_F$ and $\|\cdot\|_2$.

$\mathcal{R}_k(m,n) = \{ M \in \mathbb{K}^{m,n} : \operatorname{rank}(M) \le k \}$

Definition 3.4.4.16. Frobenius norm

The Frobenius norm of $\mathbf{A} \in \mathbb{K}^{m,n}$ is defined as

$$\|\mathbf{A}\|_F^2 := \sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2 .$$

≙ Euclidean vector norm for matrices

Approximation error norm: ($m \ge n$)

Full SVD: $A = U \Sigma V^H = U \begin{bmatrix} \Sigma_k & 0 \\ 0 & \sigma_{k+1} \ddots \sigma_n \\ 0 & 0 \end{bmatrix} V^H$

$$A_k = U \begin{bmatrix} \Sigma_k & 0 \\ 0 & 0 \end{bmatrix} V^H$$

$$\|A - A_k\|_{2/F} = \left\| U \begin{bmatrix} 0 & 0 \\ 0 & \sigma_{k+1} \ddots \sigma_n \\ 0 & 0 \end{bmatrix} V^H \right\|_{2/F} = \sigma_{k+1}$$

$\|\cdot\|_{F} = \sigma_{k+1}^2 + \cdots + \sigma_{(1)}^2$

because $\|\cdot\|_{2/F}$ is invariant w.r.t. multiplication with orthogonal matrices!

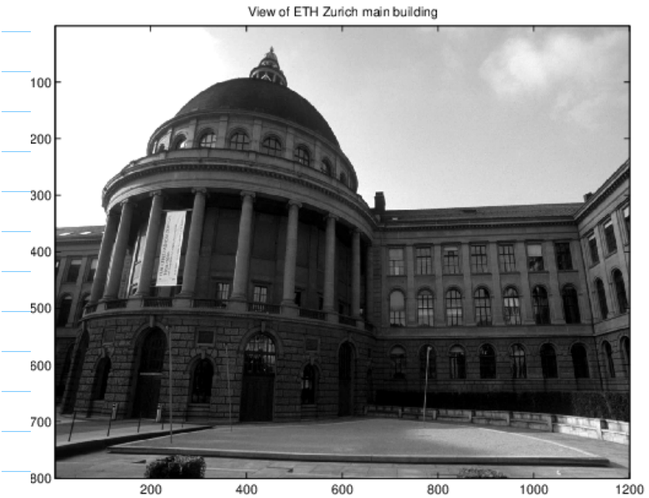
In Euclidean matrix norm: $$\left\|\mathbf{A} - \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^{\mathrm{H}}\right\|_2 = \sigma_{k+1}, \tag{3.4.4.21}$$

in Frobenius norm: $$\left\|\mathbf{A} - \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^{\mathrm{H}}\right\|_F^2 = \sum_{j=k+1}^{\min\{m,n\}} \sigma_j^2 . \tag{3.4.4.22}$$

▷ Decay of SV predicts quality of low-rank approximation

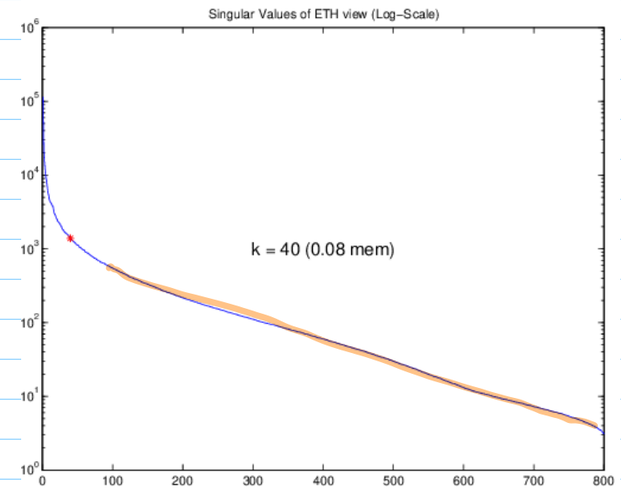
Relative compression error $\dfrac{\|A - A_k\|_2}{\|A\|_2} = \dfrac{\sigma_{k+1}}{\sigma_1} \overset{!}{\le} \text{tol}$

Ex 3.4.4.23 SVD for image compression



BMP grayscale image

$\leftrightarrow$ matrix $A \in \{0,\dots,255\}^{600,1200}$



◁ lin-log scale

! Fast (exponential) decay of the singular value

$\sigma_j \sim q^j$

for $0 < q < 1$

Review questions

Please answer review questions 3.4.4.25

ETH Lecture 401-0663-00L Numerical Methods for CSE

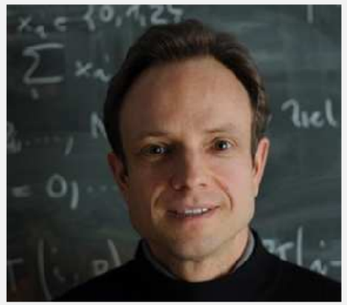
Course Video

Section 3.4.4.3: Principal Component Data Analysis (PCA)

Prof. R. Hiptmair, SAM, ETH Zurich

Date: September 29, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 3.4.1, Section 3.4.2

Duration: 54 minutes

III. Direct Methods for Linear Least Squares Problems

3.4. Singular Value Decomposition (SVD)

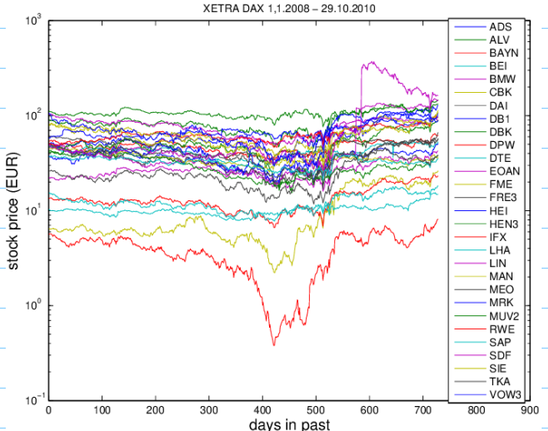
3.4.4. SVD-Based Optimization and Approximation

3.4.4.3. Principal Component Analysis (PCA)

Trend detection

Data classification

Ex. 3.4.4.25 (Trend analysis)



△ Prices of n stocks for m days

$a_k \in \mathbb{R}^m$, $k = 1, \ldots, n$

$A = [a_1, \ldots, a_n] \in \mathbb{R}^{m,n}$

(2)

Task: Identify dominant/independent trends

LA phrasing: Find orthonormal vectors $\boldsymbol{u}_1, \dots, \boldsymbol{u}_p \in \mathbb{R}^m$, $p \ll m, n$ such that

$\rightarrow$ $\underline{a}_k \in \operatorname{Span}\{\boldsymbol{u}_1, \dots, \boldsymbol{u}_p\}$

not realistic, due to "random perturbation"

Realistic goal:

Find orthonormal vectors $\boldsymbol{u}_1, \dots, \boldsymbol{u}_p \in \mathbb{R}^m$, $p \ll m, n$ such that

$\underline{a}_k \in \operatorname{Span}\{\boldsymbol{u}_1, \dots, \boldsymbol{u}_p\}$ + "small remainder"

SVD: $A = \sum_{\ell=1}^{p} \sigma_\ell \boldsymbol{u}_\ell \boldsymbol{v}_\ell^T + \sum_{\ell=p+1}^{\min\{n,m\}} \sigma_\ell \boldsymbol{u}_\ell \boldsymbol{v}_\ell^T$

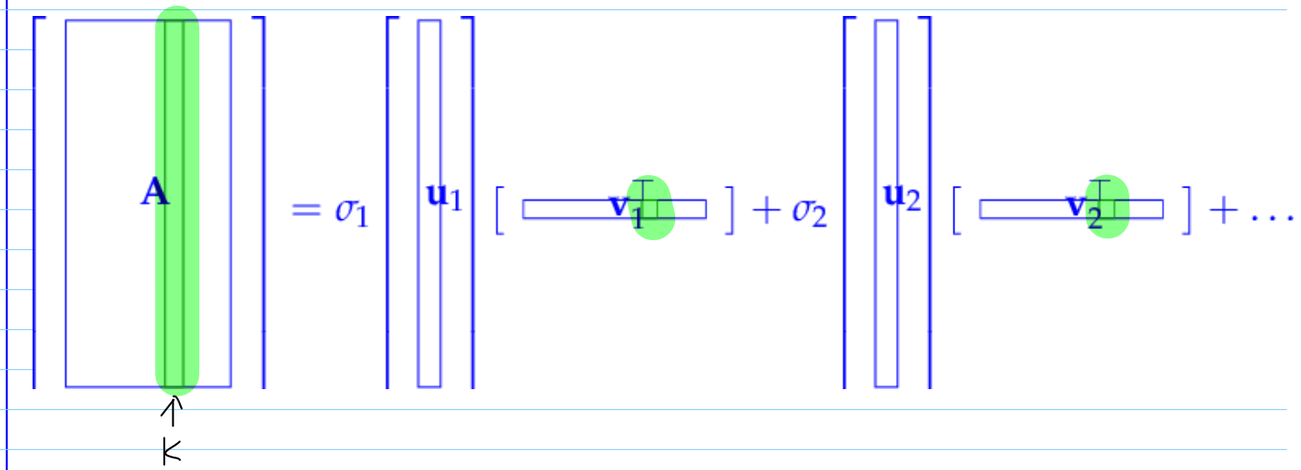
[$A = U \Sigma V^T$: $\boldsymbol{u}_\ell := (U)_{:,\ell}$, $\boldsymbol{v}_\ell = (V)_{:,\ell}$]

can be achieved, if SV decay fast, because

$\left\| \sum_{\ell=p+1}^{\min\{n,m\}} \sigma_\ell \boldsymbol{u}_\ell \boldsymbol{v}_\ell^T \right\|_2 = \sigma_{p+1}$

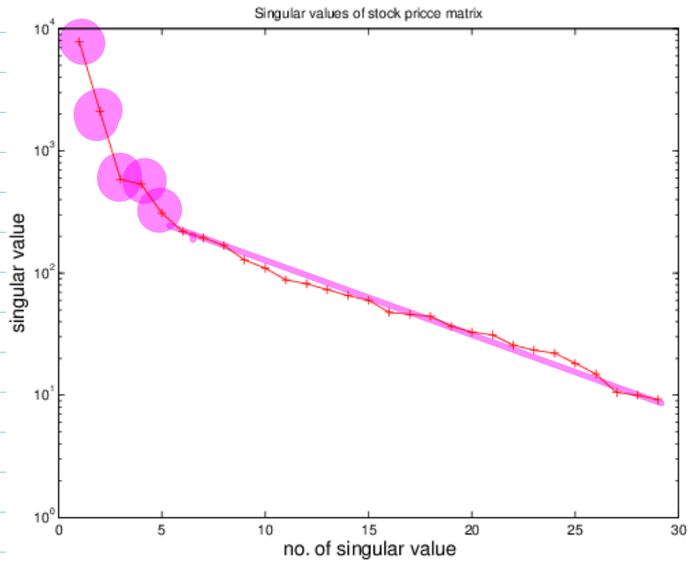
Trends $\leftrightarrow$ p leftmost columns of U

- $\sigma_\ell \leftrightarrow$ importance of ℓ-th trend



$(\boldsymbol{v}_\ell)_k \leftrightarrow$ strength of the ℓ-th trend in $\underline{a}_k$

Ex. 3.4.4.25/3.4.4.30: PCA of stock prices



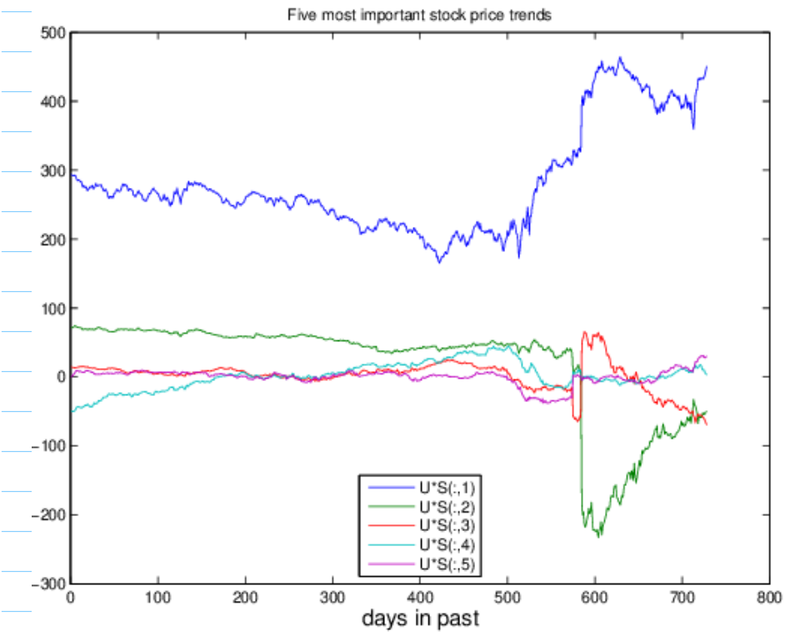
◁ Lin-log plot of σ_j's

σ_j decay exponentially

$\sigma_j = O(q^j)$

for some $0 < q < 1$

$p \approx 5$ trends capture the time-dependence of stock prices.



◁ $\sigma_j \mathbf{u}_j$

§ 3.4.4.33 Proper Orthogonal Decomposition (POD)

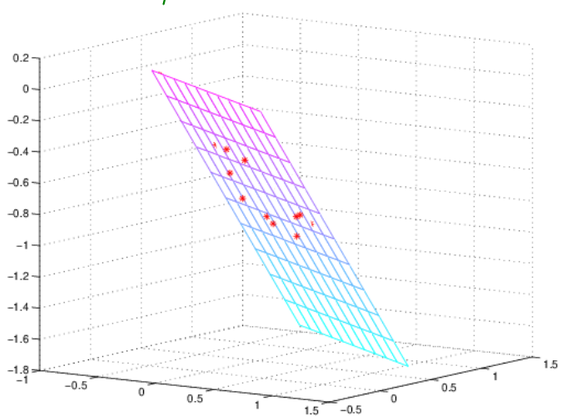
Ex. 3.4.4.32 : (Detect data points $\approx$ subspace)

Data points • $\leftrightarrow \mathbf{a}_i \in \mathbb{R}^3$ "almost" located on on a plane $\hat{=}$ 2D subspace ▷

Non-zero singular values of $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_n] \in \mathbb{R}^{3,n}$

3.1378
1.8092
0.1792

} 2 dominant SVs



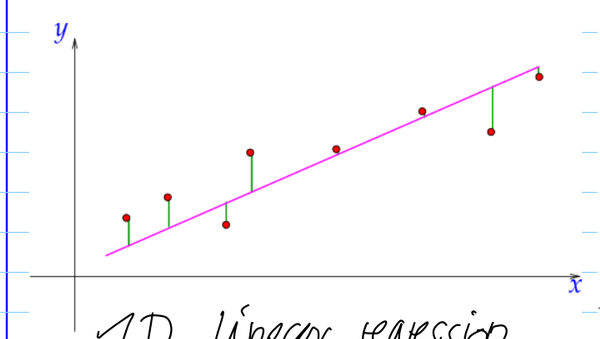
▷ SVD detects, if data approximately $\in$ subspace

Problem of proper orthogonal decomposition (POD): $\hat{=}$ provide ONB

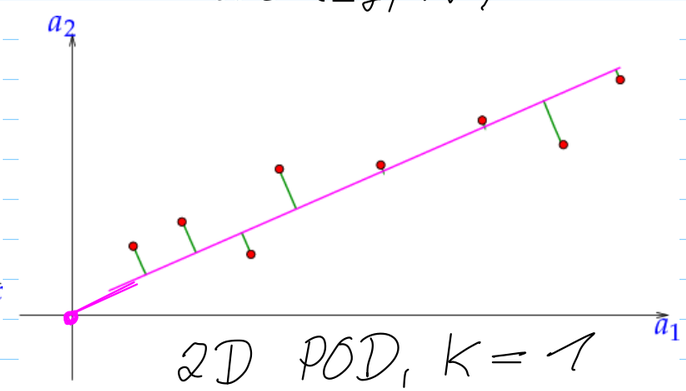
Given: Data points $\mathbf{a}_1, \dots, \mathbf{a}_n \in \mathbb{R}^m$, $m, n \in \mathbb{N}$ (often $n \gg m$)
Sought: For $k \le \min\{m, n\}$, find a *subspace* $U_k \subset \mathbb{R}^m$ such that

$$U_k = \operatorname*{argmin}_{W \subset \mathbb{R}^m, \dim W = k} \sum_{j=1}^{n} \inf_{\mathbf{w} \in W} \|\mathbf{a}_j - \mathbf{w}\|_2^2 , \qquad (3.4.4.34)$$

$\inf_{\mathbf{w} \in W} \|\mathbf{a}_j - \mathbf{w}\|_2^2 = \operatorname{dist}(\mathbf{a}_j, W)$



1D linear regression



2D POD, $k = 1$

Subspace $W \subset \mathbb{R}^m$, $\dim W = k$, ONB: $W = \operatorname{Span}\{\mathbf{w}_1, \dots, \mathbf{w}_k\}$,

$$W = [\mathbf{w}_1, \dots, \mathbf{w}_k] \in \mathbb{R}^{m,k}, \quad k \le m$$

$$\operatorname{dist}(\mathbf{x}, W) = \min_{\mathbf{y} \in \mathbb{R}^k} \|\mathbf{x} - W\mathbf{y}\|_2$$

$\hat{=}$ A linear least squares problem, minimizer $\mathbf{y} = W^T\mathbf{x}$

④

$$\operatorname{dist}(x, W) = \| x - WW^T x \|_2$$

$$\sum_{j=1}^{n} \inf_{w \in W} \| a_j - w \|_2^2 = \sum_{j=1}^{n} \| a_j - WW^T a_j \|_2^2 = \| A - WW^T A \|_F^2$$

$$\text{POD} \longleftrightarrow \operatorname*{argmin}_{W \in \mathbb{R}^{m,k},\, W^T W = I} \| A - \underbrace{WW^T A}_{\operatorname{rank} \le k} \|_F^2$$

Definition 3.4.4.16. Frobenius norm

The Frobenius norm of $\mathbf{A} \in \mathbb{K}^{m,n}$ is defined as

$$\|\mathbf{A}\|_F^2 := \sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2 .$$

Theorem 3.4.4.18. Best low rank approximation →

Let $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{H}}$ be the SVD of $\mathbf{A} \in \mathbb{K}^{m,n}$ (→ Thm. 3.4.1.1). For $1 \le k \le \operatorname{rank}(\mathbf{A})$ set

$$\mathbf{A}_k := \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^{\mathrm{H}} = \sum_{\ell=1}^{k} \sigma_\ell (\mathbf{U})_{:,\ell} (\mathbf{V})_{:,\ell}^{\mathrm{H}} \quad \text{with} \quad \begin{aligned} \mathbf{U}_k &:= \left[(\mathbf{U})_{:,1}, \ldots, (\mathbf{U})_{:,k} \right] \in \mathbb{K}^{m,k} , \\ \mathbf{V}_k &:= \left[(\mathbf{V})_{:,1}, \ldots, (\mathbf{V})_{:,k} \right] \in \mathbb{K}^{n,k} , \\ \mathbf{\Sigma}_k &:= \operatorname{diag}(\sigma_1, \ldots, \sigma_k) \in \mathbb{K}^{k,k} . \end{aligned}$$

Then, for $\|\cdot\| = \|\cdot\|_F$ and $\|\cdot\| = \|\cdot\|_2$, holds true

$$\|\mathbf{A} - \mathbf{A}_k\| \le \|\mathbf{A} - \mathbf{F}\| \quad \forall \mathbf{F} \in \mathcal{R}_k(m,n) ,$$

that is, $\mathbf{A}_k$ is the rank-k best approximation of $\mathbf{A}$ in the matrix norms $\|\cdot\|_F$ and $\|\cdot\|_2$.

$$\left\| \mathbf{A} - \mathbf{W}\mathbf{W}^T \mathbf{A} \right\|_F \ge \|\mathbf{A} - \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k\|_F \quad \forall \mathbf{W} \in \mathbb{R}^{m,k} , \quad \mathbf{W}^T \mathbf{W} = \mathbf{I} .$$

Can we find $W \in \mathbb{R}^{m,k}$ with

(i) $W^T W = I$

(ii) $WW^T A = U_k \Sigma_k V_k$

Lucky: $W = U_k$

$$\mathbf{W}\mathbf{W}^T \mathbf{A} = \mathbf{U}_k \mathbf{U}_k^T \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \mathbf{U}_k \left[\mathbf{I}_k \;\; \mathbf{O} \right] \mathbf{\Sigma}\mathbf{V}^T = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T .$$

U_k has orthonormal columns

Theorem 3.4.4.37. Solution of POD problem

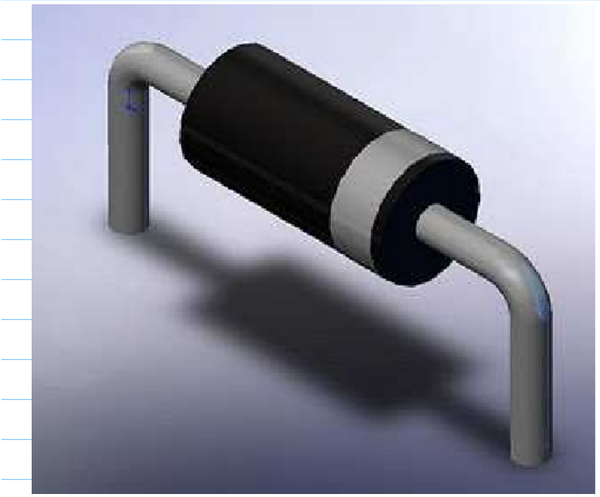
The subspace U_k spanned by the first k left singular vectors of $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_n] \in \mathbb{R}^{m,n}$ solves the POD problem (3.4.4.34):

$$U_k = \mathcal{R}((\mathbf{U})_{:,1:k}) \quad \text{with} \quad \mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \quad \text{the SVD of } \mathbf{A}.$$

POD estimate:

$$\sum_{j=1}^{n} \inf_{w \in U_k} \|\mathbf{a}_j - \mathbf{w}\|_2^2 = \sum_{\ell=k+1}^{p} \sigma_\ell^2 . \tag{3.4.4.38}$$

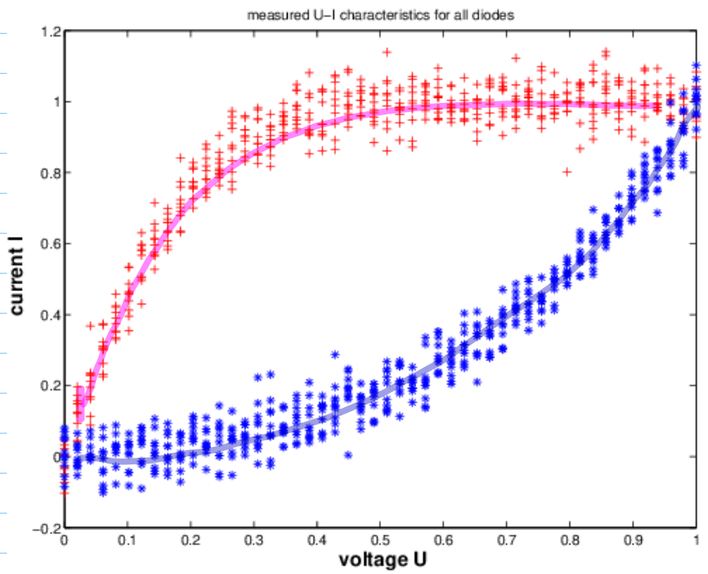
Ex. 3.4.4.26 : Data classification



Given: measured U-I characteristics of n diodes in a box
(data points $(U_j, I_j^{(k)})$, $j = 1,\ldots,m$, $k = 1,\ldots,n$)

Classification problem: find out
- how many different types of diodes in box,
- the U-I characteristic of each type.

Measurement errors !
Manufacturing tolerances !

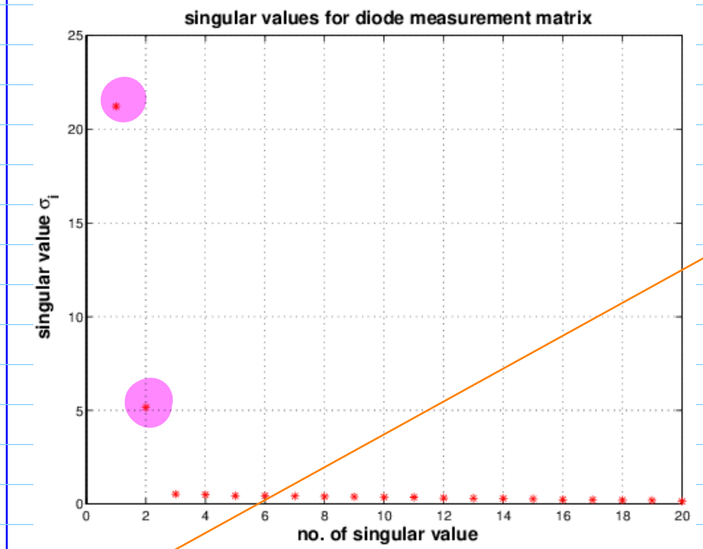


△ Two types
"two trends"

Data vector $a_k \in \mathbb{R}^m$
[in voltages]

$A = [a_1, \ldots, a_n] \in \mathbb{R}^{m,n}$

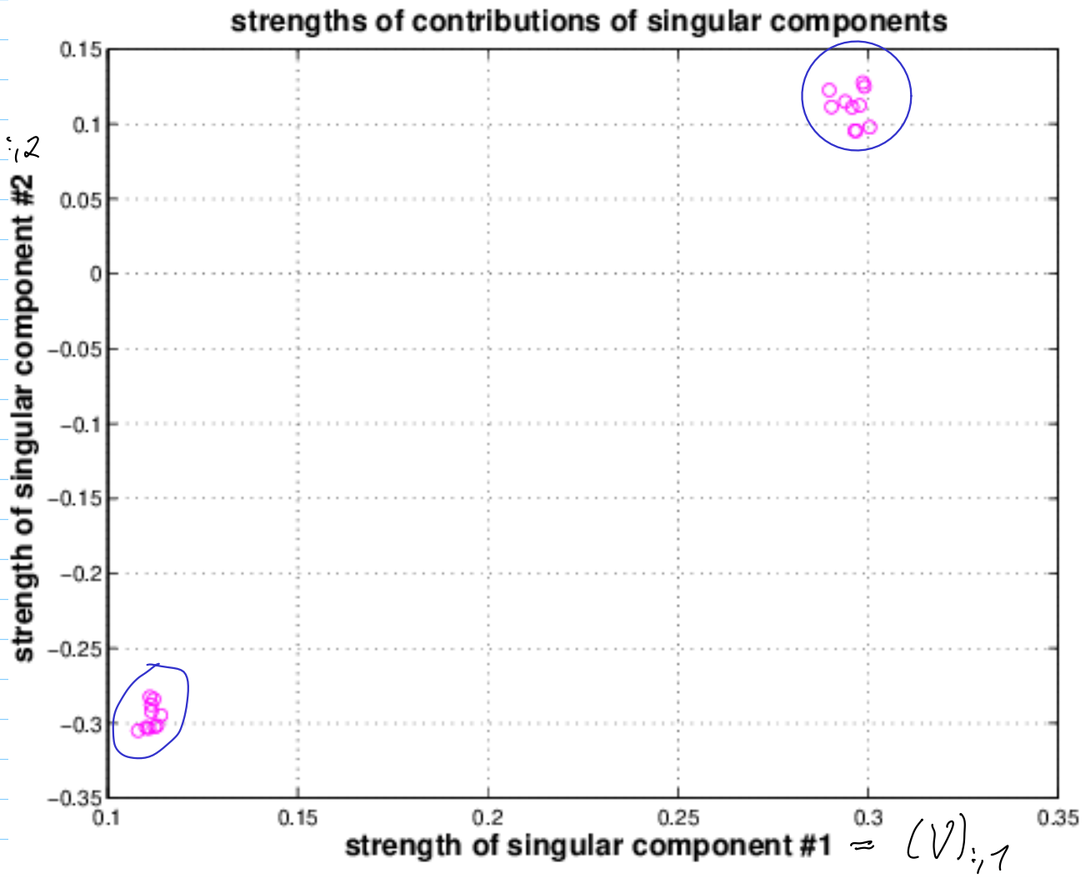
$n \hat{=}$ no. of diodes



△ two dominant SVs

Idea : Classification by comparing components of $(V)_{:,1}$ and $(V)_{:,2}$

↑ strength of presence of $(U)_{:,1}$ / $(U)_{:,2}$ in a particular measurement



$(V)_{:,2}$

$\approx (V)_{:,1}$

Next : cluster analysis → lecture document

Review questions :

Try to answer review questions 3.4.4.51

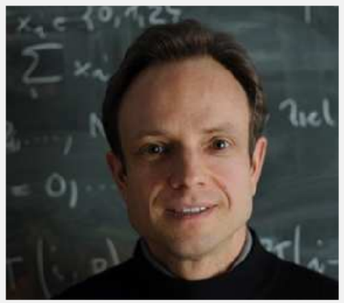
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 3.6: Constrained Least Squares

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 1, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- Gradient of a function $\mathbb{R}^n \to \mathbb{R}$

Dependencies. Section 3.1, Section 3.4.1

Duration: 45 minutes

III. Direct Methods for Linear Least Squares Problems

3.6. Constrained Least Squares

$m \times n$ OD-LSE, but $p < n$ eqns. have to be satisfied exactly.

$$\begin{matrix} m \updownarrow \\ p \updownarrow \end{matrix} \begin{bmatrix} A \\ C \end{bmatrix} x = \begin{bmatrix} b \\ d \end{bmatrix}, \quad m \geq n$$

Linear least squares problem with linear constraint:

Given: $\mathbf{A} \in \mathbb{R}^{m,n}$, $m \geq n$, $\operatorname{rank}(\mathbf{A}) = n$, $\mathbf{b} \in \mathbb{R}^m$,
$\mathbf{C} \in \mathbb{R}^{p,n}$, $p < n$, $\operatorname{rank}(\mathbf{C}) = p$, $\mathbf{d} \in \mathbb{R}^p$

Find: $\mathbf{x} \in \mathbb{R}^n$ such that $\|\mathbf{Ax} - \mathbf{b}\|_2 \to \min$ and $\boxed{\mathbf{Cx} = \mathbf{d}}$.

(3.6.0.1)

Linear constraint

3.6.1.: Solution via Lagrange Multipliers

↳ essential tool in constrained optimization

Idea: coupling the constraint using the Lagrange multiplier $\mathbf{m} \in \mathbb{R}^p$

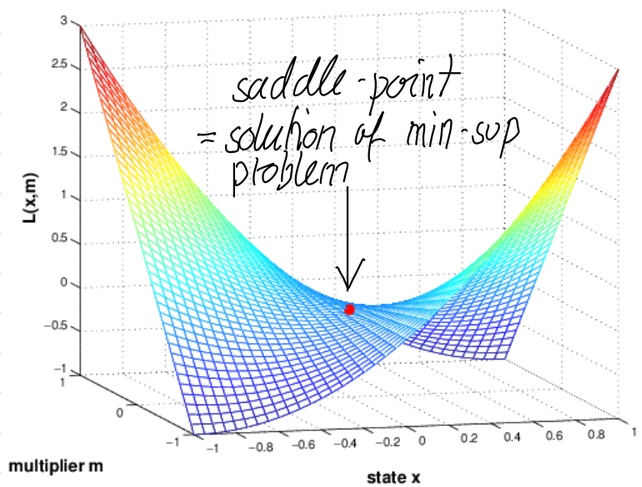
$$\mathbf{x} = \operatorname{argmin}_{\mathbf{y}\in\mathbb{R}^n} \sup_{\mathbf{m}\in\mathbb{R}^p} L(\mathbf{y},\mathbf{m}) , \tag{3.6.1.2}$$

$$L(\mathbf{y},\mathbf{m}) := \frac{1}{2}\|\mathbf{A}\mathbf{y}-\mathbf{b}\|_2^2 + \mathbf{m}^\top(\mathbf{C}\mathbf{y}-\mathbf{d}) . \tag{3.6.1.3}$$

Lagrange functional $= \infty$, unless $Cy - d = 0$

Constraint satisfied by any minimizer

(3.6.1.2) is called a saddle-point problem



$L(x,m) = x^2 - 2xm$
$[n = p = 1]$

△ Graph of L is flat* in the saddle point

* all partial derivatives vanish in a saddle point

If (x,q) is a saddle point of L

$$\Rightarrow \frac{\partial L}{\partial y}(x,q) = \frac{\partial L}{\partial m}(x,q) = 0$$

Computing partial derivatives

$$L(y,m) = \frac{1}{2}\sum_{j=1}^{m}\Big(\sum_{i=1}^{n}(A)_{j,i}y_i - b_j\Big)^2 + \sum_{k=1}^{p} m_k\Big(\sum_{i=1}^{n}(C)_{k,i}y_i - d_k\Big)$$

$y = [y_1, \ldots, y_n]^\top$, $m = [m_1, \ldots, m_p]^\top$

A quadratic polynomial in y_i, m_k

Eqns.: $\frac{\partial L}{\partial y_1}(x,q) = \ldots = \frac{\partial L}{\partial y_n}(x,q) = \frac{\partial L}{\partial m_1}(x,q) = \ldots = \frac{\partial L}{\partial m_p}(x,q) = 0$

$$\frac{\partial L}{\partial \mathbf{x}}(\mathbf{x},\mathbf{q}) = \mathbf{A}^\top(\mathbf{A}\mathbf{x}-\mathbf{b}) + \mathbf{C}^\top\mathbf{q} \stackrel{!}{=} 0 , \tag{3.6.1.5a}$$

$$\frac{\partial L}{\partial \mathbf{m}}(\mathbf{x},\mathbf{q}) = \mathbf{C}\mathbf{x}-\mathbf{d} \stackrel{!}{=} 0 . \tag{3.6.1.5b}$$

$(n+p)\times(n+p)$ LSE:

$$\begin{bmatrix} \mathbf{A}^\top\mathbf{A} & \mathbf{C}^\top \\ \mathbf{C} & 0 \end{bmatrix}\begin{bmatrix} \mathbf{x} \\ \mathbf{q} \end{bmatrix} = \begin{bmatrix} \mathbf{A}^\top\mathbf{b} \\ \mathbf{d} \end{bmatrix}$$

Augmented normal equations (matrix saddle point problem) (3.6.1.6)

↑ symmetric with zero right-lower block

③

Unique solution, if • $\operatorname{rank}(A) = n$

• $\operatorname{rank}(C) = p$

Algorithm based on block-LU decomposition

$$\begin{bmatrix} A^TA & C^T \\ C & 0 \end{bmatrix} = \begin{bmatrix} I & 0 \\ Y & I \end{bmatrix} \begin{bmatrix} U_1 & X \\ 0 & U_2 \end{bmatrix}$$

- $U_1 = A^TA$
- $X = C^T$
- $YU_1 = C \Leftrightarrow U_1^T Y^T = C^T$ [p nxn-LSE]
- $YX + U_2 = 0$

Rem: If A large, sparse ⇒ do not compute A^TA, but solve extended equation with auxiliary unknown $r := Ax - b$ (residual)

$$\begin{bmatrix} -\mathbf{I} & \mathbf{A} & 0 \\ \mathbf{A}^\top & 0 & \mathbf{C}^\top \\ 0 & \mathbf{C} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{r} \\ \mathbf{x} \\ \mathbf{m} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ 0 \\ \mathbf{d} \end{bmatrix} \hat{=} \text{ Extended augmented normal equations .} \quad (3.6.1.8)$$

3.6.2. Solution via SVD

Constraint: $Cx = d$ with $C \in \mathbb{R}^{p,n}$ $(p < n)$, of full rank, $\operatorname{rank}(C) = p$

$$x \in x_0 + \mathcal{N}(C)$$

x_0: particular solution: $Cx_0 = d$, accessible via SVD

Lemma 3.4.1.13. SVD and rank of a matrix →

If, for some $1 \le r \le p := \min\{m,n\}$, the singular values of $\mathbf{A} \in \mathbb{K}^{m,n}$ satisfy $\sigma_1 \ge \cdots \ge \sigma_r > \sigma_{r+1} = \cdots \sigma_p = 0$, then

$A = U\Sigma V$:

- $\operatorname{rank}(\mathbf{A}) = r$ *(no. of non-zero singular values)*,
- $\mathcal{N}(\mathbf{A}) = \operatorname{Span}\{(\mathbf{V})_{:,r+1}, \ldots, (\mathbf{V})_{:,n}\}$,
- $\mathcal{R}(\mathbf{A}) = \operatorname{Span}\{(\mathbf{U})_{:,1}, \ldots, (\mathbf{U})_{:,r}\}$.

Full SVD:

$$C = U \begin{bmatrix} \Sigma_p & 0 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix}$$

$U \in \mathbb{R}^{p,p}$ orth., $\Sigma \in \mathbb{R}^{p,p}$ diag., $V \in \mathbb{R}^{n,n}$ orth.

$\mathcal{N}(C) = \mathcal{R}(V_2)$ $\qquad \Sigma = V$

$$x = x_0 + V_2 y, \quad y \in \mathbb{R}^{n-p}$$

$$\|Ax - b\|_2 = \|A(x_0 + V_2 y) - b\|_2 = \|\underbrace{AV}_{m \times (n-p)} y - (b - Ax_0)\|_2 \longrightarrow \min$$

(4)

$\hat{=}$ Std. linear LSQ in $y \in \mathbb{R}^{n-p}$

Rem: Finding $x_0 \in \mathbb{R}^n$

Idea: $x_0 \in \mathcal{N}(C)^\perp = \mathcal{R}(V_1)$

$x_0 = V_1 z$, $z \in \mathbb{R}^p$

$[Cx = d]$ $\quad C V_1 z = d$

$\Leftrightarrow \quad U \underbrace{[\Sigma_p \; 0] \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} V_1}_{= \begin{bmatrix} I_p \\ 0 \end{bmatrix}} z = d$

$\Sigma_p \in \mathbb{R}^{p,p}$ is regular, since $\operatorname{rank}(C) = p$

$\Leftrightarrow \quad U \Sigma_p z = d$

$\Leftrightarrow \quad z = \Sigma_p^{-1} U^T d$

Σ just scaling

$x_0 = V_1 \Sigma_p^{-1} U^T d$

Review questions

Study the review questions 3.6.2.1

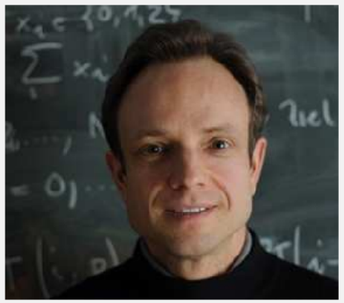
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 4.1.2: LT-FIR Linear Mappings

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 4, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 4.1.1

Duration: 24 minutes

IV. Filtering Algorithms
4.1. Filters and Convolutions
4.1.2. LT-FIR Linear Mappings

Definition 4.1.1.2. Finite channel/filter

A channel/filter $F : \ell^\infty(\mathbb{Z}) \to \ell^\infty(\mathbb{Z})$ is called **finite**, if every input signal of finite duration produces an output signal of finite duration,

$$\{\, \exists M \in \mathbb{N}:\ |j| > M \ \Rightarrow\ x_j = 0 \,\} \ \Rightarrow\ \exists N \in \mathbb{N}:\ |k| > N \ \Rightarrow\ (F((x_j)_{j\in\mathbb{Z}}))_k = 0\,, \tag{4.1.1.3}$$

Definition 4.1.1.5. Time-invariant channel/filter

A filter $F : \ell^\infty(\mathbb{Z}) \to \ell^\infty(\mathbb{Z})$ is called **time-invariant (TI)**, if shifting the input in time leads to the same output shifted in time by the same amount; it *commutes with the time shift operator* from (4.1.1.4):

$$\forall (x_j)_{j\in\mathbb{Z}} \in \ell^\infty(\mathbb{Z}),\ \forall m \in \mathbb{Z}:\quad F(\mathsf{S}_m((x_j)_{j\in\mathbb{Z}})) = \mathsf{S}_m(F((x_j)_{j\in\mathbb{Z}}))\,. \tag{4.1.1.6}$$

Definition 4.1.1.7. Linear channel/filter

A filter $F : \ell^\infty(\mathbb{Z}) \to \ell^\infty(\mathbb{Z})$ is called **linear**, if F is a linear mapping:

$$F(\alpha(x_j)_{j\in\mathbb{Z}} + \beta(y_j)_{j\in\mathbb{Z}}) = \alpha F((x_j)_{j\in\mathbb{Z}}) + \beta F((y_j)_{j\in\mathbb{Z}}) \tag{4.1.1.8}$$

for all sequences $(x_j)_{j\in\mathbb{Z}}$, $(y_j)_{j\in\mathbb{Z}} \in \ell^\infty(\mathbb{Z})$ and real numbers $\alpha, \beta \in \mathbb{R}$.

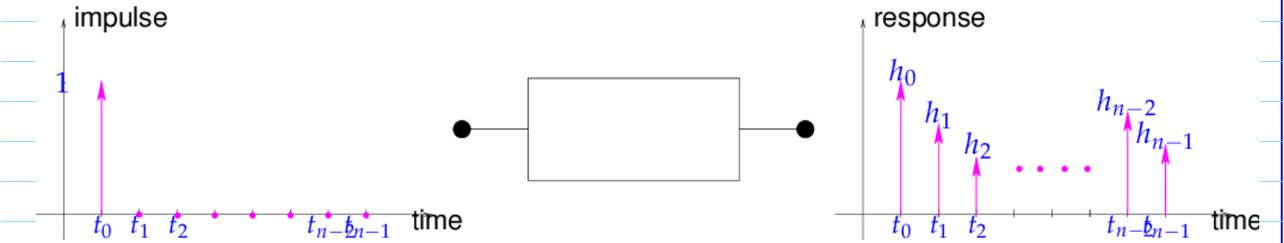
Definition 4.1.1.9. Causal channel/filter

A filter $F : \ell^\infty(\mathbb{Z}) \to \ell^\infty(\mathbb{Z})$ is called **causal** (or physical, or nonanticipative), if the output does not start before the input

$$\forall M \in \mathbb{N}: \quad (x_j)_{j\in\mathbb{Z}} \in \ell^\infty(\mathbb{Z}),\ x_j = 0 \quad \forall j \le M \quad \Rightarrow \quad F((x_j)_{j\in\mathbb{Z}})_k = 0 \quad \forall k \le M\,. \tag{4.1.1.10}$$

Definition 4.1.1.12. Impulse response

The **impulse response** (IR) of a channel/filter is the output for a single unit impulse at $t = 0$ as input, that is, the input signal is $x_j = \delta_{j,0} := \begin{cases} 1 & \text{, if } j = 0 \\ 0 & \text{else} \end{cases}$ (Kronecker symbol).



$$F((\delta_{j,0})_{j\in\mathbb{Z}}) = (\dots, 0, h_0, h_1, \dots, h_{n-1}, 0, \dots)\,.$$

TI: $F \circ S_m = S_m \circ F \quad \forall m \in \mathbb{Z}$

$$F((\delta_{j,k})_{j\in\mathbb{Z}}) = (h_{j-k})_{j\in\mathbb{Z}} = (\cdots\ 0\ \underset{\uparrow}{h_0}\ h_1\ \cdots\ \underset{\uparrow}{h_{n-1}}\ 0\ \cdots)\,.$$

$t_k = k\Delta t$ $\qquad$ $t = (k+n-1)\Delta t$

↑ time-shifted unit pulse

For finite signal (x_j):

$$(x_j)_{j\in\mathbb{Z}} = \sum_{k=0}^{m-1} x_k (\delta_{j,k})_{j\in\mathbb{Z}} = \sum_{k=0}^{m-1} x_k S_k\big((\delta_{j,0})_{j\in\mathbb{Z}}\big)\,, \quad \text{≙ linear combination of pulses} \tag{4.1.2.1}$$

$$F((x_j)) \overset{\text{linearity}}{=} \sum_k x_k F(S_k(\delta_{j,0})_{j\in\mathbb{Z}}) \overset{\text{T.I.}}{=} \sum x_k S_k \underbrace{F((\delta_{j,0})_j)}_{= \text{I.R.}}$$

Superposition of shifted IR:

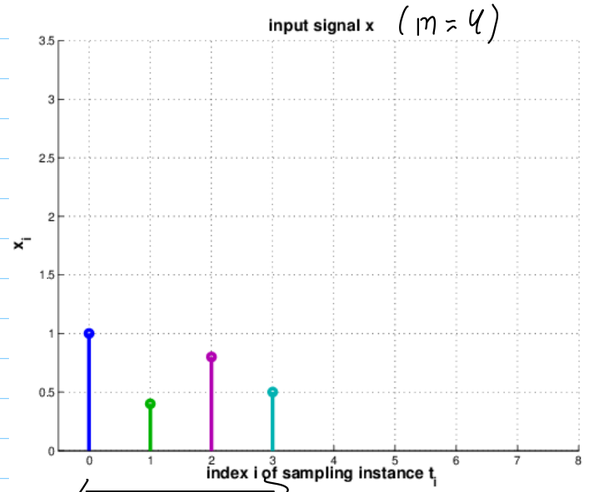
$$\begin{bmatrix} \vdots \\ 0 \\ y_0 \\ y_1 \\ \vdots \\ y_n \\ \vdots \\ y_{m+n-3} \\ y_{m+n-2} \\ 0 \\ \vdots \end{bmatrix} = x_0 \begin{bmatrix} \vdots \\ 0 \\ h_0 \\ \vdots \\ h_{n-1} \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix} + x_1 \begin{bmatrix} \vdots \\ 0 \\ 0 \\ h_0 \\ \vdots \\ h_{n-1} \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix} + x_2 \begin{bmatrix} \vdots \\ 0 \\ 0 \\ 0 \\ h_0 \\ \vdots \\ h_{n-1} \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \end{bmatrix} + \cdots + x_{m-1} \begin{bmatrix} \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \\ h_0 \\ \vdots \\ h_{n-1} \\ 0 \\ \vdots \end{bmatrix}\,. \tag{4.1.2.3}$$

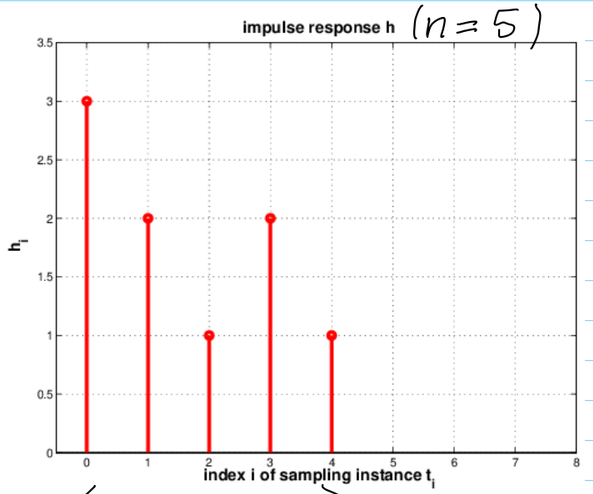
$$y_k = \sum_{j=0}^{m+n-2} h_{k-j} x_j\,, \quad k = 0, \dots, m+n-2$$

[Recall: $h_k = 0$ for $k < 0$ & $k \ge n$]

NumCSE@ETHZ

Example:

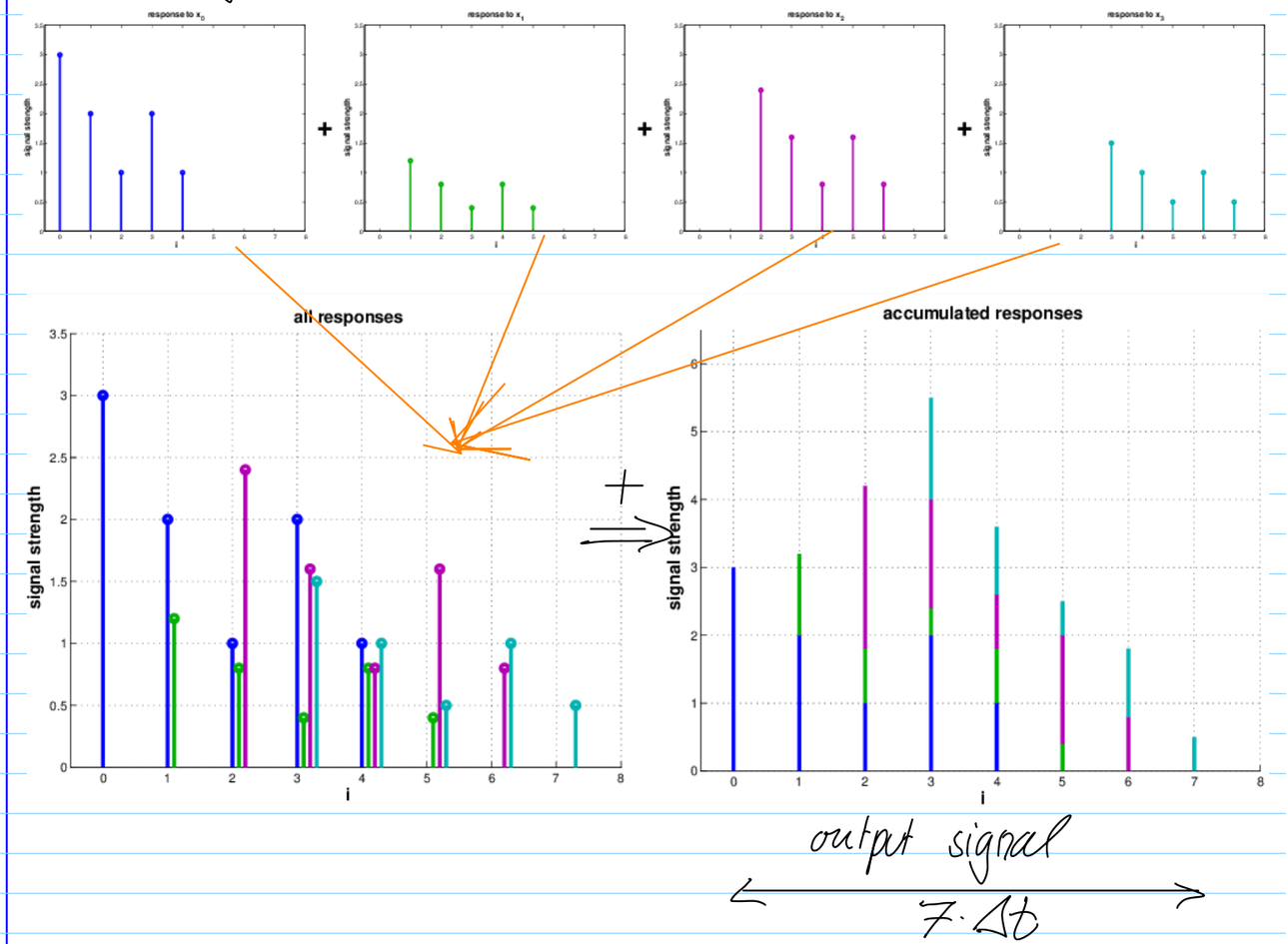




$$(4.1.2.3) \quad \stackrel{\Delta}{=} \quad \begin{bmatrix} \vdots \\ 0 \\ y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ 0 \\ \vdots \end{bmatrix} = x_0 \begin{bmatrix} \vdots \\ 0 \\ h_0 \\ h_1 \\ h_2 \\ h_3 \\ h_4 \\ 0 = h_5 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix} + x_1 \begin{bmatrix} \vdots \\ 0 \\ 0 \\ h_0 \\ h_1 \\ h_2 \\ h_3 \\ h_4 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix} + x_2 \begin{bmatrix} \vdots \\ 0 \\ 0 \\ 0 \\ h_0 \\ h_1 \\ h_2 \\ h_3 \\ h_4 \\ 0 \\ 0 \\ \vdots \end{bmatrix} + x_3 \begin{bmatrix} \vdots \\ 0 \\ 0 \\ 0 \\ 0 \\ h_0 \\ h_1 \\ h_2 \\ h_3 \\ h_4 \\ 0 \\ \vdots \end{bmatrix}$$

at t_{m+n-2} →

x_k · weighted, $k\Delta t$-time shifted IR:



Duration (output) = duration (input) + duration (IR)

Extension to infinite signals:

$$y_k = \sum_{j=0}^{m-1} h_{k-j} x_j = \sum_{j\in\mathbb{Z}} h_{k-j} x_j$$

Definition 4.1.2.7. Convolution of sequences

Given two sequences $(h_k)_{k\in\mathbb{Z}}, (x_k)_{k\in\mathbb{Z}}$, at least one of which is finite or decays sufficiently fast, their **convolution** is another sequence $(y_k)_{k\in\mathbb{Z}}$, defined as

$$y_k = \sum_{j\in\mathbb{Z}} h_{k-j} x_j \,, \quad k \in \mathbb{Z} \,. \tag{4.1.2.8}$$

Notation $(y_j) = (h_k) * (x_k)$

Theorem 4.1.2.9. Convolution of sequences commutes

If well-defined, the convolution of sequences commutes

$$(x_k) * (h_k) = (h_k) * (x_k) \,.$$

Review questions 4.1.2.10

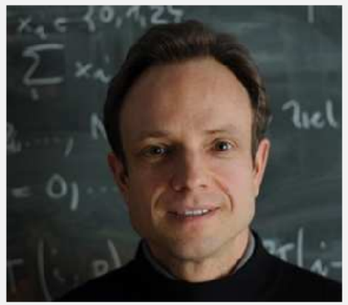
▷ Tackle them without relying on notes or the lecture document.

ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video

Section 4.1.1: Discrete Finite Linear Time-Invariant Causal Channels/Filters

Prof. R. Hiptmair, SAM, ETH Zurich



Date: October 4, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich

Prerequisites.

- Matrix-vector calculus from linear algebra

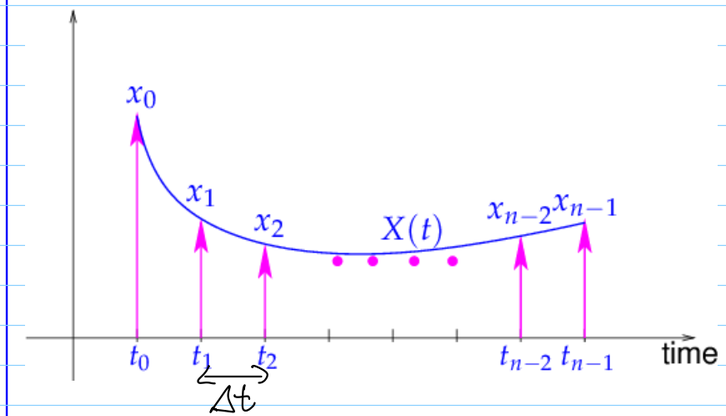
Duration: 20 minutes

IV. Filtering Algorithms

4.1. Filters and Convolutions

→ Numerical signal processing : LA perspective

Focus : Time-discrete, finite, analog signals



→ vector $\in \mathbb{R}^n$ by sampling at equidistant points in time

Time-discrete, analog signals $\leftrightarrow$ sequence $\in \mathbb{R}^{\mathbb{Z}}$

Notation : $(x_k)_{k\in\mathbb{Z}}$

NumCSE@ETHZ

4.1.1. Discrete Finite Linear Time-Invariant Causal Channels/Filters

Filter/channel $\hat{=}$ F : {input signals} $\longrightarrow$ {output signals}

$F : \ell^\infty(\mathbb{Z}) \longrightarrow \ell^\infty(\mathbb{Z})$

$\ell^\infty(\mathbb{Z}) \hat{=}$ vector space of bounded sequences $\in \mathbb{R}^{\mathbb{Z}}$
↳ ∞-dimensional

Definition 4.1.1.2. Finite channel/filter

A channel/filter $F : \ell^\infty(\mathbb{Z}) \to \ell^\infty(\mathbb{Z})$ is called **finite**, if every input signal of finite duration produces an output signal of finite duration,

$$\{\, \exists M \in \mathbb{N}: \ |j| > M \ \Rightarrow \ x_j = 0 \,\} \ \Rightarrow \ \exists N \in \mathbb{N}: \ |k| > N \ \Rightarrow \ (F((x_j)_{j\in\mathbb{Z}}))_k = 0\,, \tag{4.1.1.3}$$

It should not matter when input arrives

Definition 4.1.1.5. Time-invariant channel/filter

A filter $F : \ell^\infty(\mathbb{Z}) \to \ell^\infty(\mathbb{Z})$ is called **time-invariant (TI)**, if shifting the input in time leads to the same output shifted in time by the same amount; it *commutes with the time shift operator* from (4.1.1.4):

$$\forall (x_j)_{j\in\mathbb{Z}} \in \ell^\infty(\mathbb{Z}),\ \forall m \in \mathbb{Z}: \ F(\mathsf{S}_m((x_j)_{j\in\mathbb{Z}})) = \mathsf{S}_m(F((x_j)_{j\in\mathbb{Z}}))\,. \tag{4.1.1.6}$$

$S_m \hat{=}$ time-shift : $S_m((x_j)) = (x_{j-m})_{j\in\mathbb{Z}}$

In math :

A commuting diagram :

$$\begin{array}{ccc} \ell^\infty(\mathbb{Z}) & \xrightarrow{F} & \ell^\infty(\mathbb{Z}) \\ \downarrow S_m & & \downarrow S_m \\ \ell^\infty(\mathbb{Z}) & \xrightarrow{F} & \ell^\infty(\mathbb{Z}) \end{array}$$

$\ell^\infty(\mathbb{Z})$ = a vector space $\rhd$ linear mapping defined

Definition 4.1.1.7. Linear channel/filter

A filter $F : \ell^\infty(\mathbb{Z}) \to \ell^\infty(\mathbb{Z})$ is called **linear**, if F is a linear mapping:

$$F(\alpha(x_j)_{j\in\mathbb{Z}} + \beta(y_j)_{j\in\mathbb{Z}}) = \alpha F((x_j)_{j\in\mathbb{Z}}) + \beta F((y_j)_{j\in\mathbb{Z}}) \tag{4.1.1.8}$$

for all sequences $(x_j)_{j\in\mathbb{Z}}, (y_j)_{j\in\mathbb{Z}} \in \ell^\infty(\mathbb{Z})$ and real numbers $\alpha, \beta \in \mathbb{R}$.

Output only after onset of input

Definition 4.1.1.9. Causal channel/filter

A filter $F : \ell^\infty(\mathbb{Z}) \to \ell^\infty(\mathbb{Z})$ is called **causal** (or physical, or nonanticipative), if the output does not start before the input

$$\forall M \in \mathbb{N}: \ (x_j)_{j\in\mathbb{Z}} \in \ell^\infty(\mathbb{Z}),\ x_j = 0 \ \ \forall j \le M \ \Rightarrow \ F((x_j)_{j\in\mathbb{Z}})_k = 0 \ \ \forall k \le M\,. \tag{4.1.1.10}$$

Acronym : **LT-FIR** $\hat{=}$ linear, time-invariant, finite (& causal) filters

NumCSE@ETHZ

Remember from LA: Linear mapping uniquely defined by action on basis vectors

For $\ell^\infty(\mathbb{Z})$: basis $e_k := (\delta_{jk})_{j\in\mathbb{Z}}$: impuls

Note: $S_{\cdot k}(e_k) = e_0 \overset{TI}{\Longrightarrow}$ knowing $F(e_0)$ defines LT-F/IR

$F(e_0) \triangleq$ impulse response (IR)

$$F((\delta_{j,0})_{j\in\mathbb{Z}}) = (\ldots, 0, h_0, h_1, \ldots, h_{n-1}, 0, \ldots) .$$

Review questions 4.1.1.13

Please answer them with consulting any notes.

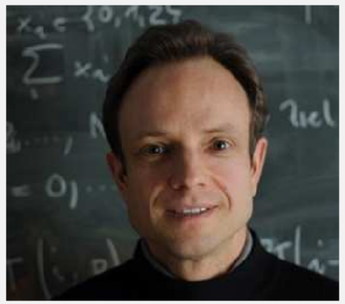
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 4.1.3: Discrete Convolutions

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 4, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 4.1.1, Section 4.1.2

Duration: 17 minutes

Note: Possible minor *mismatch of video and tablet notes*!

[Corrections and updates can be incorporated into tablet notes only]

IV. Filtering Algorithms

4.1. Filters and Convolutions

4.1.3. Discrete Convolutions

LT-FIR channel F, with $\mathbb{R} = (\ldots, 0, h_0, h_1, \ldots, h_{n-1}, 0, \ldots)$

$$\begin{bmatrix} \vdots \\ 0 \\ y_0 \\ y_1 \\ \vdots \\ y_n \\ \vdots \\ y_{m+n-3} \\ y_{m+n-2} \\ 0 \\ \vdots \end{bmatrix} = x_0 \begin{bmatrix} \vdots \\ 0 \\ h_0 \\ \vdots \\ h_{n-1} \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \end{bmatrix} + x_1 \begin{bmatrix} \vdots \\ 0 \\ 0 \\ h_0 \\ \vdots \\ h_{n-1} \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \end{bmatrix} + x_2 \begin{bmatrix} \vdots \\ 0 \\ 0 \\ 0 \\ h_0 \\ \vdots \\ h_{n-1} \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \end{bmatrix} + \cdots + x_{m-1} \begin{bmatrix} \vdots \\ 0 \\ 0 \\ \vdots \\ \vdots \\ \vdots \\ 0 \\ h_0 \\ \vdots \\ h_{n-1} \\ 0 \\ \vdots \end{bmatrix} . \qquad (4.1.2.3)$$

(2)

LT-FIR action for $m = 4$, $n = 5$

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = x_0 \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \\ h_4 \\ 0 \\ 0 \\ 0 \end{bmatrix} + x_1 \begin{bmatrix} 0 \\ h_0 \\ h_1 \\ h_2 \\ h_3 \\ h_4 \\ 0 \\ 0 \end{bmatrix} + x_2 \begin{bmatrix} 0 \\ 0 \\ h_0 \\ h_1 \\ h_2 \\ h_3 \\ h_4 \\ 0 \end{bmatrix} + x_3 \begin{bmatrix} 0 \\ 0 \\ 0 \\ h_0 \\ h_1 \\ h_2 \\ h_3 \\ h_4 \end{bmatrix}.$$

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} h_0 & 0 & 0 & 0 \\ h_1 & h_0 & 0 & 0 \\ h_2 & h_1 & h_0 & 0 \\ h_3 & h_2 & h_1 & h_0 \\ h_4 & h_3 & h_2 & h_1 \\ 0 & h_4 & h_3 & h_2 \\ 0 & 0 & h_4 & h_3 \\ 0 & 0 & 0 & h_4 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_4 \end{bmatrix}.$$

↓ = cyclic permutation

$$\begin{bmatrix} \vdots \\ 0 \\ y_0 \\ y_1 \\ \vdots \\ y_n \\ \vdots \\ \vdots \\ y_{m+n-3} \\ y_{m+n-2} \\ 0 \\ \vdots \end{bmatrix} = x_0 \begin{bmatrix} \vdots \\ 0 \\ h_0 \\ \vdots \\ h_{n-1} \\ 0 \\ 0 \\ \vdots \\ \vdots \\ 0 \\ 0 \\ \vdots \end{bmatrix} + x_1 \begin{bmatrix} \vdots \\ 0 \\ 0 \\ h_0 \\ \vdots \\ h_{n-1} \\ 0 \\ \vdots \\ \vdots \\ 0 \\ 0 \\ \vdots \end{bmatrix} + x_2 \begin{bmatrix} \vdots \\ 0 \\ 0 \\ 0 \\ h_0 \\ \vdots \\ h_{n-1} \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \end{bmatrix} + \cdots + x_{m-1} \begin{bmatrix} \vdots \\ 0 \\ 0 \\ \vdots \\ \vdots \\ 0 \\ h_0 \\ \vdots \\ h_{n-1} \\ 0 \\ \vdots \end{bmatrix}. \qquad (4.1.2.3)$$

$(m \geq n)$

$$\mathbf{y} := \begin{bmatrix} y_0 \\ \vdots \\ \\ \\ \\ \vdots \\ y_{m+n-2} \end{bmatrix} = \begin{bmatrix} h_0 & 0 & \cdots & 0 \\ h_1 & h_0 & & \vdots \\ & h_1 & \ddots & 0 \\ h_{n-1} & & \ddots & h_0 \\ 0 & \ddots & & h_1 \\ \vdots & \ddots & h_{n-1} & \\ 0 & \cdots & 0 & h_{n-1} \end{bmatrix} \begin{bmatrix} x_0 \\ \vdots \\ \\ \\ \\ \vdots \\ x_{m-1} \end{bmatrix} =: \mathbf{C}\mathbf{x}. \qquad (4.1.3.2)$$

m+n-1 (rows), m (columns)

③

Output sequence (y_k) : $y_k = \sum_{j\in\mathbb{Z}} h_{k-j} x_j$

(convolution of sequences)

$$y_k = \sum_{j=0}^{m-1} h_{k-j} x_j \quad , k = 0, \dots, m+n-2$$

Definition 4.1.3.3. Discrete convolution

Given $\mathbf{x} = [x_0, \dots, x_{m-1}]^\top \in \mathbb{K}^m$, $\mathbf{h} = [h_0, \dots, h_{n-1}]^\top \in \mathbb{K}^n$ their discrete convolution is the vector $\mathbf{y} \in \mathbb{K}^{m+n-1}$ with components

$$y_k = \sum_{j=0}^{m-1} h_{k-j} x_j \,, \quad k = 0, \dots, 2n-2 \,, \tag{4.1.3.4}$$

where we have adopted the convention $h_j := 0$ for $j < 0$ or $j \geq n$.

Notation: $\underline{y} = \underline{h} * \underline{x} = \underline{x} * \underline{h}$

↑ commutes !

4.1.3.6. Multiplication of polynomials

$p, q \triangleq$ polynomials in t of degree $\leq n-1$

$p \cdot q$ = a polynomial of degree $\leq 2n-2$

$$p(t) := \sum_{j=0}^{n-1} a_j t^j \,, \quad q(t) := \sum_{j=0}^{n-1} b_j t^j \,, \quad a_j, b_j \in \mathbb{K} \,. \quad \longleftrightarrow \quad \underline{a}, \underline{b} \in \mathbb{K}^n$$

discrete convolution $\underline{c} = \underline{a} * \underline{b}$

$$(pq)(t) = \sum_{k=0}^{2n-2} c_k t^k \,, \quad c_k := \sum_{\ell=\max\{0,k-(n-1)\}}^{\min\{k,n-1\}} a_\ell b_{k-\ell} \,, \quad k = 0, \dots, 2n-2 \,. \tag{4.1.3.7}$$

Review questions 4.1.3.12 :

Please answer without recourse to lecture/tablet notes.

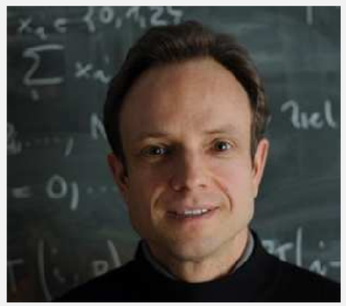
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 4.1.4: Periodic Convolutions

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 4, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 4.1.2, **??**

Duration: 24 minutes

IV. Filtering Algorithms
4.1. Filters and Convolutions
4.1.4. Periodic Convolutions

Definition 4.1.2.7. Convolution of sequences

Given two sequences $(h_k)_{k\in\mathbb{Z}}, (x_k)_{k\in\mathbb{Z}}$, at least one of which is finite or decays sufficiently fast, their **convolution** is another sequence $(y_k)_{k\in\mathbb{Z}}$, defined as

$$y_k = \sum_{j\in\mathbb{Z}} h_{k-j} x_j \,, \quad k \in \mathbb{Z} \,. \qquad : (y_k) = (h_k) * (x_k) \tag{4.1.2.8}$$

Definition 4.1.3.3. Discrete convolution

Given $\mathbf{x} = [x_0, \ldots, x_{m-1}]^\top \in \mathbb{K}^m$, $\mathbf{h} = [h_0, \ldots, h_{n-1}]^\top \in \mathbb{K}^n$ their discrete convolution is the vector $\mathbf{y} \in \mathbb{K}^{m+n-1}$ with components

$$y_k = \sum_{j=0}^{m-1} h_{k-j} x_j \,, \quad k = 0, \ldots, 2n-2 \,, \tag{4.1.3.4}$$

where we have adopted the convention $h_j := 0$ for $j < 0$ or $j \geq n$.

(2)

Now: infinite signal, but periodic

Definition 4.1.4.1. Periodic time-discrete signal

An n-**periodic** signal, $n \in \mathbb{N}$, is a sequence $(x_j)_{j\in\mathbb{Z}} \in \ell^\infty(\mathbb{Z})$ satisfying

$$x_{j+n} = x_j \quad \forall j \in \mathbb{Z}\,.$$

$\rightarrow$ vector $\underline{x} = \begin{bmatrix} x_0 \\ \vdots \\ x_{n-1} \end{bmatrix} \in \mathbb{R}^n$

Action LT-FIR filter F with IR $(h_k)_{k\in\mathbb{Z}}$ on n-periodic sequence (x_k): $(y_k) := F((x_k))$

$$y_k = \sum_{j\in\mathbb{Z}} h_{k-j} x_j = \sum_{j\in\mathbb{Z}} h_j x_{k-j}$$

$$= \sum_{\nu=0}^{n-1} \sum_{\ell\in\mathbb{Z}} h_{\nu+\ell n}\, x_{k-\nu-\ell n} \quad \rightarrow \text{by periodicity}$$

$$= \sum_{\nu=0}^{n-1} \underbrace{\left(\sum_{\ell\in\mathbb{Z}} h_{\nu+\ell n}\right)}_{=:p_\nu} x_{k-\nu}$$

$\rightarrow$ n-periodic seq. $(p_\nu)_{\nu\in\mathbb{Z}}$

$$= \sum_{\nu=0}^{n-1} p_\nu x_{k-\nu} = (p_k) *_n (x_k)$$

(y_k) is also n-periodic

Definition 4.1.4.7. Discrete periodic convolution

The **discrete periodic convolution** of two n-periodic sequences $(p_k)_{k\in\mathbb{Z}}$, $(x_k)_{k\in\mathbb{Z}}$ yields the n-periodic sequence

$$(y_k) := (p_k) *_n (x_k) \quad , \quad y_k := \sum_{j=0}^{n-1} p_{k-j} x_j = \sum_{j=0}^{n-1} x_{k-j} p_j \,, \quad k \in \mathbb{Z}\,.$$

$\longleftrightarrow$ $\underline{y} = \underline{p} *_n \underline{x}$, $\underline{y}, \underline{p}, \underline{x} \in \mathbb{R}^n$

§ 4.1.4.11 : Circulant matrices

$$\begin{bmatrix} y_0 \\ \vdots \\ y_{n-1} \end{bmatrix} = \underbrace{\begin{bmatrix} p_0 & p_{n-1} & p_{n-2} & \cdots & \cdots & p_1 \\ p_1 & p_0 & p_{n-1} & \ddots & & \vdots \\ p_2 & p_1 & p_0 & \ddots & \ddots & \\ \vdots & \ddots & \ddots & \ddots & \ddots & \\ & \ddots & \ddots & \ddots & p_{n-1} & \\ p_{n-1} & \cdots & & p_1 & p_0 \end{bmatrix}}_{=:\mathbf{P}} \begin{bmatrix} x_0 \\ \vdots \\ x_{n-1} \end{bmatrix} . \tag{4.1.4.6}$$

↳ cyclic permutation

Definition 4.1.4.12. Circulant matrix $\rightarrow$

A matrix $\mathbf{C} = [c_{ij}]_{i,j=1}^n \in \mathbb{K}^{n,n}$ is **circulant**

$:\Leftrightarrow$ $\exists\, (p_k)_{k\in\mathbb{Z}}$ n-periodic sequence: $c_{ij} = p_{j-i}, 1 \le i, j \le n$.

③

→ C.M. completely defined by $p \in \mathbb{K}^{17}$
↳ "data sparse"

4.1.4.15 Convolution by periodic convolution

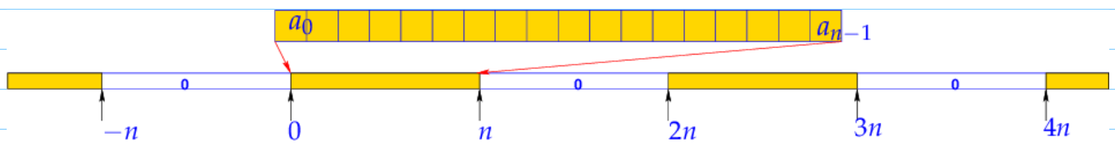
$$\underline{a} = \begin{bmatrix} a_0 \\ \vdots \\ a_{n-1} \end{bmatrix}, \quad \underline{b} = \begin{bmatrix} b_0 \\ \vdots \\ b_{n-1} \end{bmatrix}, \quad n \in \mathbb{N}$$

$$\underline{z} = \underline{a} * \underline{b} : z_k = \sum_{j=0}^{n-1} a_{j-k} b_j, \quad k = 0, \dots, 2n-2$$

Idea: Suppress interaction of different periods by zero padding

$$x_k := \begin{cases} a_k & \text{, if } 0 \le k < n, \\ 0 & \text{, if } n \le k < 2n-1 \end{cases}, \quad y_k := \begin{cases} b_k & \text{, if } 0 \le k < n, \\ 0 & \text{, if } n \le k < 2n-1, \end{cases} \tag{4.1.4.16}$$

→ associated $2n-1$-periodic sequence



$$\underline{z} = \underline{a} * \underline{b} = \underline{X} *_{2n-1} \underline{Y}$$

$$\begin{bmatrix} z_0 \\ \vdots \\ z_{2n-2} \end{bmatrix} = \begin{bmatrix} b_0 & 0 & \cdots & 0 & b_{n-1} & \cdots & b_1 \\ b_1 & \ddots & & & 0 & & b_0 \\ \vdots & \ddots & b_0 & & & & \vdots \\ b_{n-1} & & b_1 & b_0 & 0 & & 0 \\ 0 & & b_{n-1} & b_1 & b_0 & & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & b_{n-1} & \cdots & b_1 & b_0 \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_{n-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{4.1.4.18}$$

a $(2n-1) \times (2n-1)$ circulant matrix!

Review questions 4.1.4.19

▷ Please answer without aids

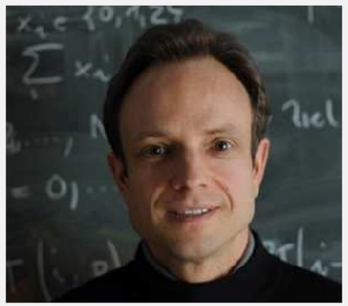
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 4.2.1: Diagonalizing Circulant Matrices

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 4, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.
- From linear algebra: eigenvalues and eigenvectors
- Computing with complex numbers

Dependencies. Section 4.1.4

Duration: 30 minutes

IV. Filtering Algorithms
4.2. Discrete Fourier Transform (DFT)
4.2.1. Diagonalizing Circulant Matrices

Recall: discrete conv. $\longleftrightarrow$ discr. periodic conv.
= multiplication with circulant matrix

Definition 4.1.4.12. Circulant matrix →

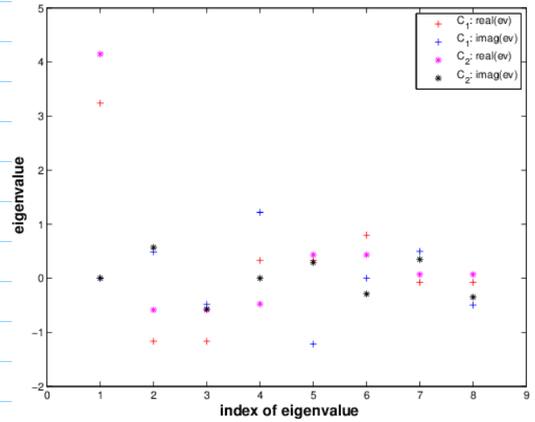
A matrix $\mathbf{C} = [c_{ij}]_{i,j=1}^{n} \in \mathbb{K}^{n,n}$ is **circulant**
$:\Leftrightarrow$ $\exists\, (p_k)_{k\in\mathbb{Z}}$ n-periodic sequence: $c_{ij} = p_{j-i}$, $1 \le i,j \le n$.

$$\operatorname{circul}(\mathbf{p}) = \begin{bmatrix} p_0 & p_1 & p_2 & \cdots & & \cdots & p_{n-1} \\ p_{n-1} & p_0 & & & & & p_{n-2} \\ p_{n-2} & & & & & & \vdots \\ \vdots & & & & & & \\ & & & & & & \vdots \\ p_2 & & & & & & p_1 \\ p_1 & p_2 & \cdots & & \cdots & p_{n-1} & p_0 \end{bmatrix}$$

(2)

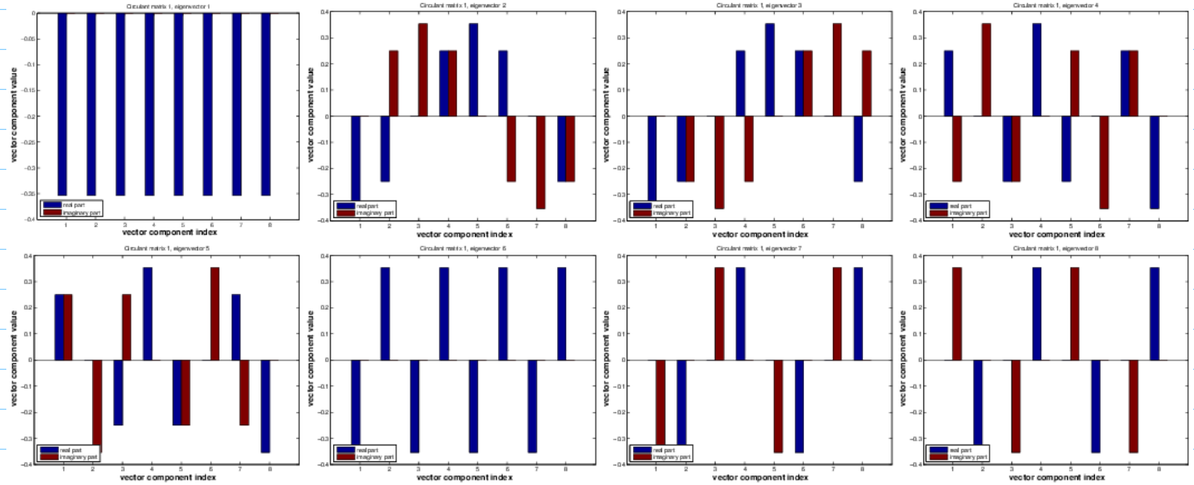
Exp.: 4.2.0.1 Eigenvectors (EVs) of circulant matrices

$C = \text{circul}(\text{VectorXd::Random}(8))$

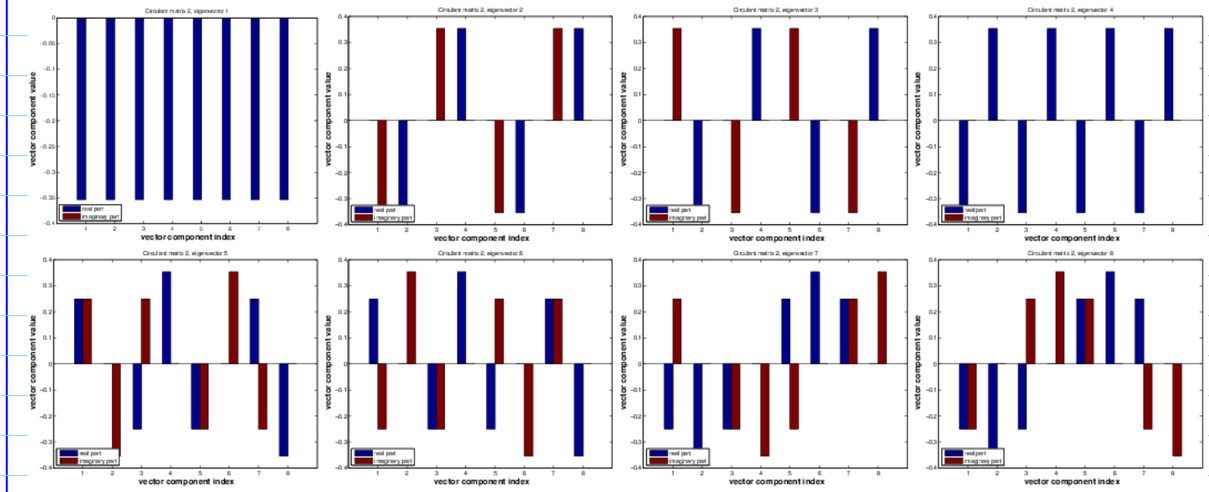


△ different (complex) eigenvalues

Eigenvectors of matrix $\mathbf{C}_1$, visualized through the size of the real and imaginary parts of their components.

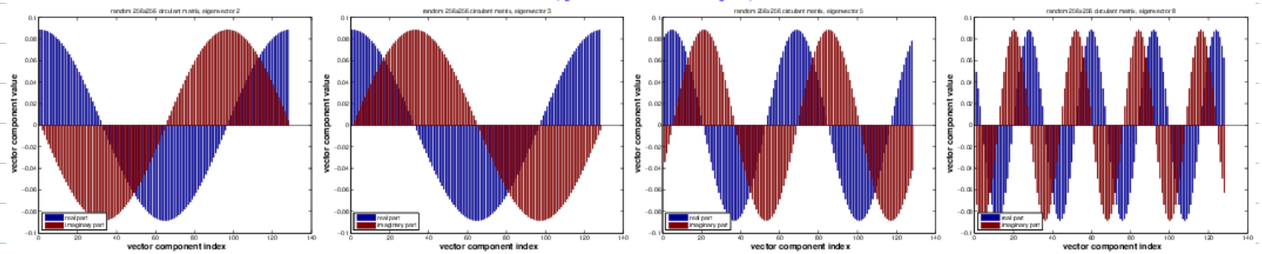


C_2 :



▷ All $n \times n$ circulant matrices have the same eigenvectors $\in \mathbb{C}^n$

Eigenvectors of circulant matrix $\mathbf{C} = \text{circul}([1, 2, \ldots, 128]^\top)$:



▷ Reminds of sin/cos

③

§ 4.2.0.6 EV circulant matrices

Recall: $e^{it} = \cos t + i \sin t$

n-th root of unity $\omega_n = e^{-\frac{2\pi i}{n}}$, $n \in \mathbb{N}$

satisfies $\overline{\omega}_n = \omega_n^{-1}$, $\omega_n^n = 1$, $\omega_n^{n/2} = -1$, $\omega_n^k = \omega_n^{k+n}$ $\forall k \in \mathbb{Z}$, (4.2.1.7)

$$\sum_{k=0}^{n-1} \omega_n^{kj} = \sum_{k=0}^{n-1} (\omega_n^j)^k = \begin{cases} n & \text{, if } j = 0 \mod n, \\ 0 & \text{, if } j \neq 0 \mod n. \end{cases} \quad \text{(4.2.1.8)}$$

by geometric sum formula

$\omega_n^{lk} = (\omega_n^l)^k$

$C \in \mathbb{C}^{n,n}$: $(C)_{i,j} = u_{i-j}$, (u_k) n-periodic

$v_k := [\omega_n^{-kj}]_{j=0}^{n-1} \in \mathbb{C}^n$, $k = 0, \dots, n-1$

$$(Cv_k)_j = \sum_{\ell=0}^{n-1} (C)_{j,\ell} (v_k)_\ell$$

$$= \sum_{\ell=0}^{n-1} u_{j-\ell} \omega_n^{-k\ell}$$

$l \mapsto j - l$

$$= \sum_{l=j-n+1}^{j} u_l \omega_n^{(l-j)k}$$

$$= \omega_n^{-jk} \cdot \underbrace{\sum_{\ell=0}^{n-1} u_\ell \omega_n^{\ell k}}_{= \lambda_k} = \lambda_k (v_k)_j$$

$j = 0, \dots, n-1$

$Cv_k = \lambda_k v_k$: EV!

Set of EVs of C:

$$\{v_0, \dots, v_{n-1}\} = \left\{ \begin{bmatrix} \omega_n^0 \\ \vdots \\ \omega_n^0 \end{bmatrix}, \begin{bmatrix} \omega_n^0 \\ \omega_n^{-1} \\ \vdots \\ \omega_n^{1-n} \end{bmatrix}, \dots, \begin{bmatrix} \omega_n^0 \\ \omega_n^{2-n} \\ \omega_n^{2(2-n)} \\ \vdots \\ \omega_n^{-(n-1)(n-2)} \end{bmatrix}, \begin{bmatrix} \omega_n^0 \\ \omega_n^{1-n} \\ \omega_n^{2(1-n)} \\ \vdots \\ \omega_n^{-(n-1)^2} \end{bmatrix} \right\}. \quad \text{(4.2.1.11)}$$

↓ orthogonal

$$v_k^H v_m = \sum_{j=0}^{n-1} \omega_n^{kj} \omega_n^{-mj} = \sum_{n=0}^{n-1} \omega_n^{(k-m)j} = 0, \text{ if } k \neq m$$

→ Fourier matrix → diagonalizes circulant matrices

$$\mathbf{F}_n = \begin{bmatrix} \omega_n^0 & \omega_n^0 & \cdots & \omega_n^0 \\ \omega_n^0 & \omega_n^1 & \cdots & \omega_n^{n-1} \\ \omega_n^0 & \omega_n^2 & \cdots & \omega_n^{2n-2} \\ \vdots & \vdots & & \vdots \\ \omega_n^0 & \omega_n^{n-1} & \cdots & \omega_n^{(n-1)^2} \end{bmatrix} = \left[\omega_n^{lj}\right]_{l,j=0}^{n-1} \in \mathbb{C}^{n,n}. \quad \text{(4.2.1.13)}$$

Lemma 4.2.1.14. Properties of Fourier matrix

The scaled Fourier-matrix $\frac{1}{\sqrt{n}}\mathbf{F}_n$ is unitary (→ Def. 6.2.1.2) : $\mathbf{F}_n^{-1} = \frac{1}{n}\mathbf{F}_n^H = \frac{1}{n}\overline{\mathbf{F}}_n$.

$C v_k = C (\overline{F_n})_{:,k} = \lambda_k (\overline{F_n})_{:,k}$

$C \overline{F_n} = \overline{F_n} D$,

$D = \operatorname{diag}(F_n u)$

(4)

Lemma 4.2.1.16. Diagonalization of circulant matrices (→ Def. 4.1.4.12)

For any circulant matrix $\mathbf{C} \in \mathbb{K}^{n,n}$, $c_{ij} = u_{i-j}$, $(u_k)_{k\in\mathbb{Z}}$ n-periodic sequence, holds true

$$\mathbf{C}\overline{\mathbf{F}}_n = \overline{\mathbf{F}}_n \operatorname{diag}(d_1,\ldots,d_n) \quad , \quad [d_0,\ldots,d_{n-1}]^\top = \mathbf{F}_n[u_0,\ldots,u_{n-1}]^\top .$$

▷ $C = F_n^{-1} D F_n$

→ All operations w/ circulant matrices can be realized by means of Fourier matrices

Definition 4.2.1.18. Discrete Fourier transform (DFT)

The linear map $\mathrm{DFT}_n : \mathbb{C}^n \mapsto \mathbb{C}^n$, $\mathrm{DFT}_n(\mathbf{y}) := \mathbf{F}_n\mathbf{y}$, $\mathbf{y} \in \mathbb{C}^n$, is called **discrete Fourier transform** (DFT), i.e. for $[c_0,\ldots,c_{n-1}] := \mathrm{DFT}_n(\mathbf{y})$

$$c_k = \sum_{j=0}^{n-1} y_j \omega_n^{kj} \quad , \quad k = 0,\ldots,n-1 . \tag{4.2.1.19}$$

$F_n^{-1} = \frac{1}{n}\overline{F}_n \Rightarrow c = \mathrm{DFT}(y) \Leftrightarrow y_k = \frac{1}{n}\sum_{j=0}^{n-1} c_j \omega_n^{-jk}$

Another "DFT"!

§ 4.2.1.21 DFT in Eigen

#include <unsupported/Eigen/FFT>

C++ code 4.2.1.22: Demo: discrete Fourier transform in EIGEN → GITLAB

```
2  int main() {
3    using Comp = complex<double>;
4    const VectorXcd::Index n = 5;
5    VectorXcd y(n),c(n),x(n);
6    y << Comp(1,0),Comp(2,1),Comp(3,2),Comp(4,3),Comp(5,4);
7    FFT<double> fft; // DFT transform object
8    c = fft.fwd(y);  // DTF of y, see Def. 4.2.1.18
9    x = fft.inv(c);  // inverse DFT of c, see (4.2.1.20)
10
11   cout << "y = " << y.transpose() << endl
12        << "c = " << c.transpose() << endl
13        << "x = " << x.transpose() << endl;
14   return 0;
15 }
```

Review questions 4.2.1.23

(Answer without aids)

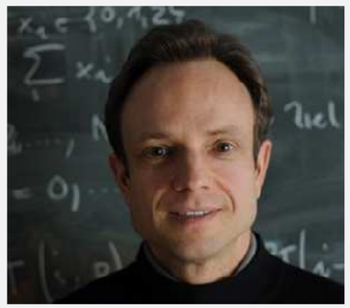
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 4.2.2: Discrete Convolution via DFT

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 4, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.
- Orthogonal diagonalization (linear algebra)

Dependencies. Section 4.2.1

Duration: 14 minutes

IV. Filtering Algorithms

4.2. Discrete Fourier Transform (DFT)

4.2.2. Discrete Convolution via DFT

Recall: discrete periodic convolution:

$$(y_k) = (p_k) *_n (x_k) \Leftrightarrow y_k = \sum_{j=0}^{n-1} p_{k-j} x_j \,, \quad k = 0, \ldots n-1$$

C++11 code 4.2.2.1: Discrete periodic convolution: straightforward implementation
➙ GITLAB

```
2  Eigen::VectorXcd pconv(const Eigen::VectorXcd &u, const Eigen::VectorXcd &x) {
3    const int n = x.size();
4    Eigen::VectorXcd z = VectorXcd::Zero(n);
5    // "naive" two-loop implementation of discrete periodic convolution
6    for (int k = 0; k < n; ++k) {
7      for (int j = 0, l = k; j <= k; ++j, --l)  z[k] += u[l] * x[j];
8      for (int j = k + 1, l = n - 1; j < n; ++j, --l)  z[k] += u[l] * x[j];
9    }
10   return z;
11 }
```

Cost $= O(n^2)$

Assume: library DFT "ultra fast"

(2)

$$z_k = \sum_{j=0}^{n-1} c_{k-j} x_j \quad \Longleftrightarrow \quad \underline{z} = C\underline{x}\,, \quad C = [c_{i-j}]_{i,j=0}^{n-1}$$

$$\| \quad \text{componentwise} \cdot$$

Idea: (4.2.1.17) ➢ $\mathbf{z} = \mathbf{F}_n^{-1}\,\mathrm{diag}(\mathbf{F}_n\mathbf{c})\,\mathbf{F}_n\mathbf{x}$

(DFT, DFT, DFT)

Lemma 4.2.1.16. Diagonalization of circulant matrices (→ Def. 4.1.4.12)

For any circulant matrix $\mathbf{C} \in \mathbb{K}^{n,n}$, $c_{ij} = u_{i-j}$, $(u_k)_{k\in\mathbb{Z}}$ n-periodic sequence, holds true

$$\mathbf{C}\overline{\mathbf{F}}_n = \overline{\mathbf{F}}_n\,\mathrm{diag}(d_1,\ldots,d_n) \quad , \quad [d_0,\ldots,d_{n-1}]^\top = \mathbf{F}_n[u_0,\ldots,u_{n-1}]^\top .$$

Theorem 4.2.2.2. Convolution theorem

The discrete periodic convolution $*_n$ between n-dimensional vectors $\mathbf{u}$ and $\mathbf{x}$ is equal to the inverse DFT of the component-wise product between the DFTs of $\mathbf{u}$ and $\mathbf{u}$; i.e.:

$$(\mathbf{u}) *_n (\mathbf{x}) := \left[\sum_{j=0}^{n-1} u_{k-j}x_j\right]_{k=0}^{n-1} = \mathbf{F}_n^{-1}\left[(\mathbf{F}_n\mathbf{u})_j(\mathbf{F}_n\mathbf{x})_j\right]_{j=1}^{n} . \qquad (4.2.2.3)$$

A commuting diagram:

$$\begin{array}{ccc} \mathbb{C}^n \times \mathbb{C}^n & \xrightarrow{\text{per. conv.}} & \mathbb{C}^n \\ \downarrow \text{DFT} \times \text{DFT} & & \uparrow \text{DFT}^{-1} \| \text{DFT} \\ \mathbb{C}^n \times \mathbb{C}^n & \xrightarrow{\text{compw. } \cdot} & \mathbb{C}^n \end{array}$$

C++11 code 4.2.2.4: Discrete periodic convolution: DFT implementation ➙ GITLAB

```
2  Eigen::VectorXcd pconvfft(const Eigen::VectorXcd &u,
3                            const Eigen::VectorXcd &x) {
4    Eigen::FFT<double> fft;
5    return fft.inv(((fft.fwd(u)).cwiseProduct(fft.fwd(x))).eval());
6  }
```

Discrete conv. by periodic convolution & zero padding

$$\underline{a}, \underline{b} \in \mathbb{C}^n : \quad \underline{a} * \underline{b} = \begin{bmatrix} \underline{a} \\ 0 \end{bmatrix} *_{2n-1} \begin{bmatrix} \underline{b} \\ 0 \end{bmatrix} \in \mathbb{R}^{2n-1}$$

C++11 code 4.2.2.5: Implementation of discrete convolution (→ Def. 4.1.3.3) based on periodic discrete convolution ➙ GITLAB

```
2  Eigen::VectorXcd fastconv(const Eigen::VectorXcd &h,
3                            const Eigen::VectorXcd &x) {
4    assert(x.size() == h.size());
5    const Eigen::Index n = h.size();
6    // Zero padding, cf. (4.1.4.16), and periodic discrete convolution
7    // of length 2n-1, Code 4.2.2.4
8    return pconvfft(
9        (Eigen::VectorXcd(2 * n - 1) << h, Eigen::VectorXcd::Zero(n - 1))
10           .finished(),
11       (Eigen::VectorXcd(2 * n - 1) << x, Eigen::VectorXcd::Zero(n - 1))
12           .finished());
13 }
```

→ Review questions 4.2.2.6

(Try to answer without consulting tablet or lecture notes)

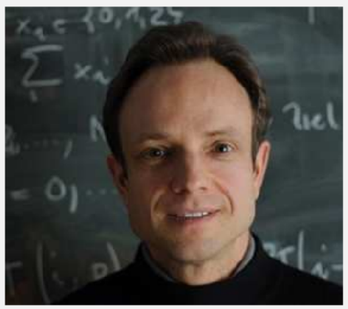
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 4.2.3: Frequency filtering via DFT

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 17, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 4.2.1, Section 4.2.2

Duration: 30 minutes

IV. Filtering Algorithms

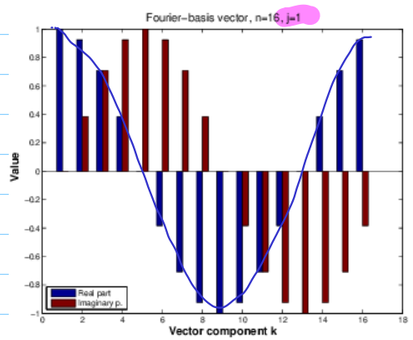
4.2. Discrete Fourier Transform (DFT)

4.2.3. Frequency Filtering via DFT

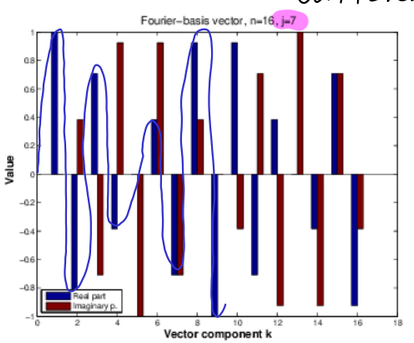
DFT: $\underline{c} = F_n \underline{y} \Leftrightarrow \underline{y} = \frac{1}{n} \overline{F}_n \underline{c}$ $\hat{=}$ a basis trf.

New new basis vector: cols. of Fourier matrix F_n

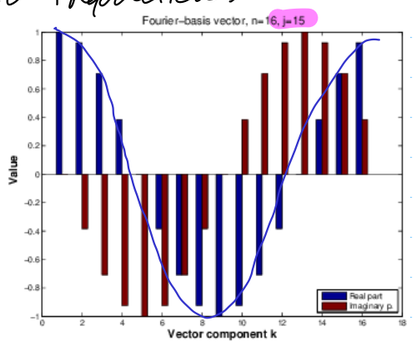
↳ represent oscillations with different frequencies



"slow oscillation/low frequency"



"fast oscillation/high frequency"



"slow oscillation/low frequency"

$$(F_n)_{:,k} = [\omega_n^{kj}]_{j=0}^{n-1} = [e^{-\frac{2\pi i}{n}kj}]_j$$

$$= [\cos(2\pi k j/n)]_j - i\,[\sin(2\pi k j/n)]_j$$

$\Rightarrow$ k-th column $\hat{=}$ sampled trigonometric functions of frequency k

② DFT

discrete Fourier coefficient

$$c_k = \sum_{j=0}^{n-1} y_j \omega_n^{kj} \iff y_j = \frac{1}{n} \sum_{k=0}^{n-1} c_k \omega_n^{-kj}$$

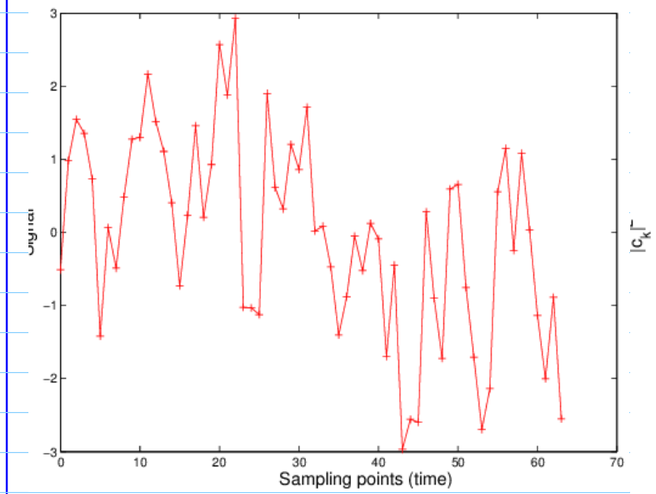
For $y_j \in \mathbb{R}^n$, $n = 2m+1$

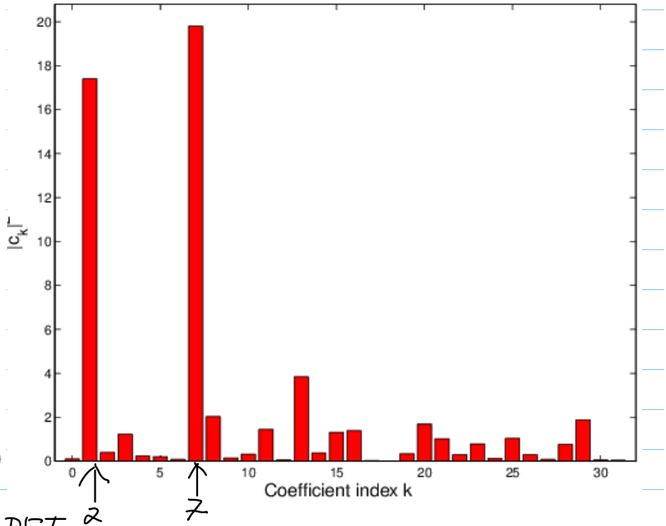
▷ (y_j) as superposition of oscillations

$$ny_j = c_0 + \sum_{k=1}^{m} c_k \omega_n^{-kj} + \sum_{k=m+1}^{2m} c_k \omega_n^{-kj} = c_0 + \sum_{k=1}^{m} c_k \omega_n^{-kj} + c_{n-k} \omega_n^{(k-n)j}$$

$$= c_0 + 2\sum_{k=1}^{m} \mathrm{Re}(c_k)\cos(2\pi kj/n) + \mathrm{Im}(c_k)\sin(2\pi kj/n), \quad j = 0,\dots,n-1,$$

$|c_k| \leftrightarrow$ strength of frequency (mode) k in signal





time-domain signal ⟷ (DFT) Frequency-domain signal

Example 4.2.3.3: Frequency identification with DFT

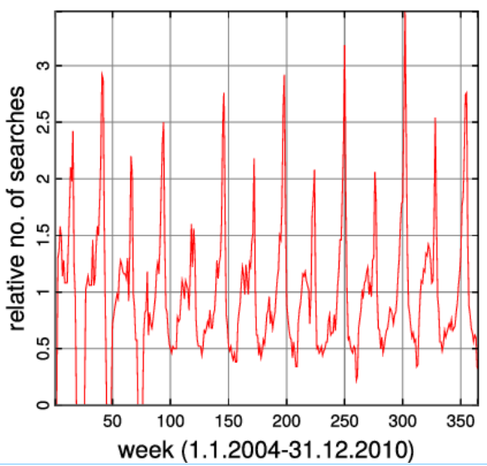
Signal = two modes + white noise

C++ code 4.2.3.4: Generation of noisy sinusoidal signal ⇢ GITLAB

```
VectorXd signalgen() {
  const int N = 64;
  const ArrayXd t = ArrayXd::LinSpaced(N, 0, N);
  const VectorXd x = ((2*M_PI/N*t).sin() + (14*M_PI/N*t).sin()).matrix();
  return x + VectorXd::Random(N);
}
```

Example 4.2.3.5 (Detecting periodicity in data)

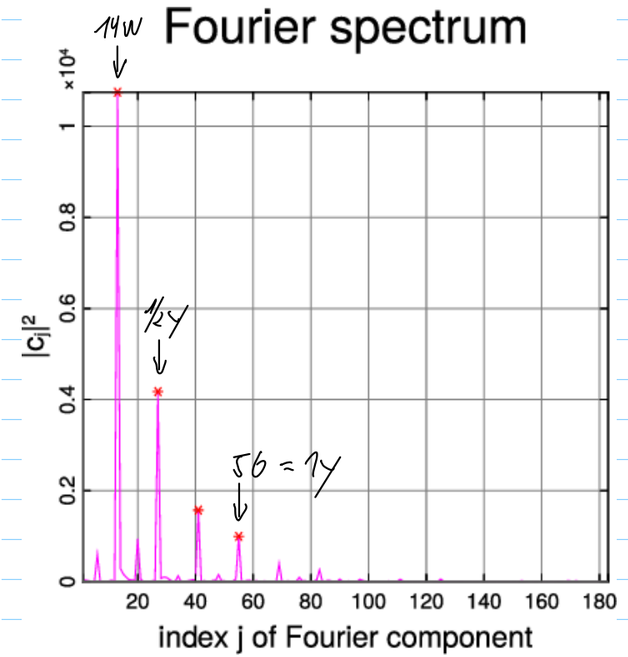
Google: 'Vorlesungsverzeichnis'



△ # weekly searches

[320 weeks]

③



◁ Power spectrum of data

$j \mapsto |c_j|^2$

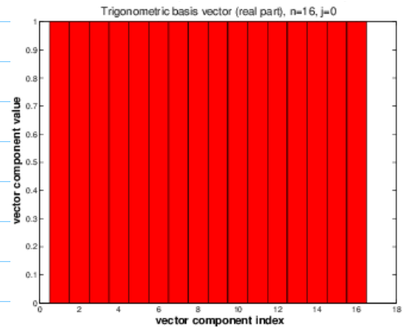
$c = \mathrm{DFT}_n y$

$\longrightarrow$ up to $\lceil n/2 \rceil$

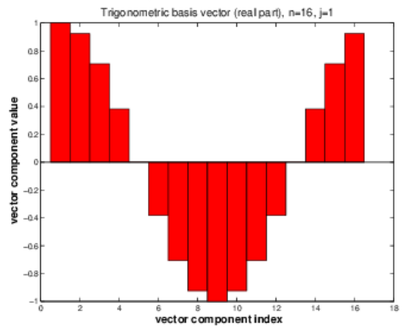
DFT is a computer's eye for periodic patterns in data

§ 4.2.3.6 ("Low" and "high" frequencies)

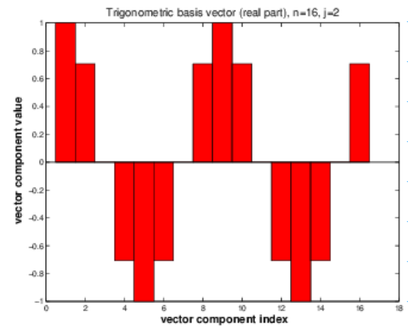
Real parts of cols. of F_n, $n = 16$



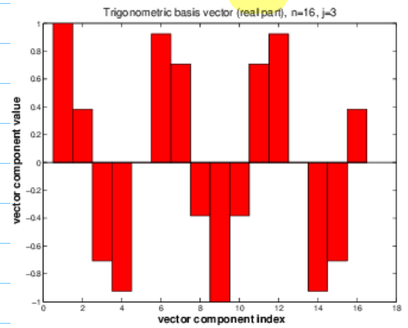
$\mathrm{Re}(\mathbf{F}_{16})_{:,0}$



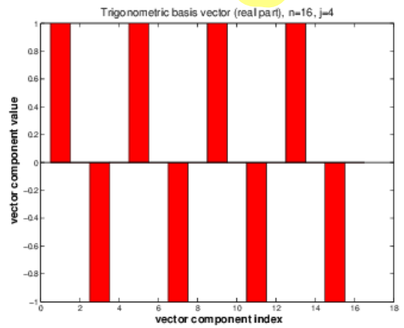
$\mathrm{Re}(\mathbf{F}_{16})_{:,1}$



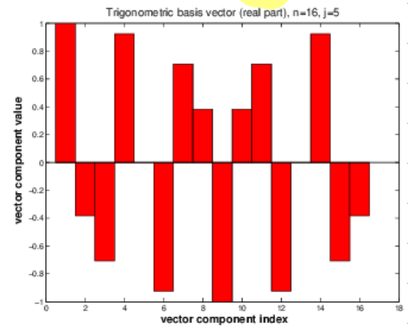
$\mathrm{Re}(\mathbf{F}_{16})_{:,2}$



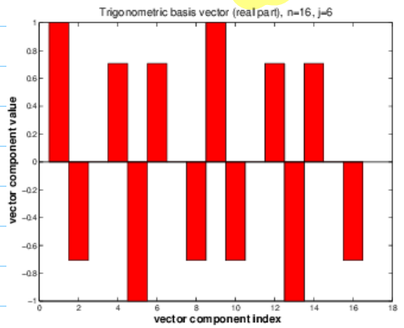
$\mathrm{Re}(\mathbf{F}_{16})_{:,3}$



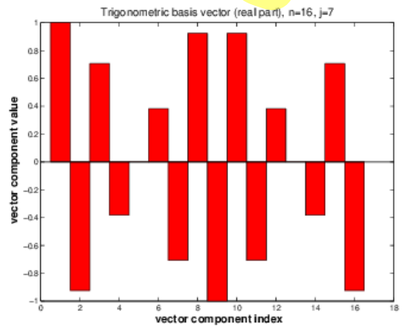
$\mathrm{Re}(\mathbf{F}_{16})_{:,4}$



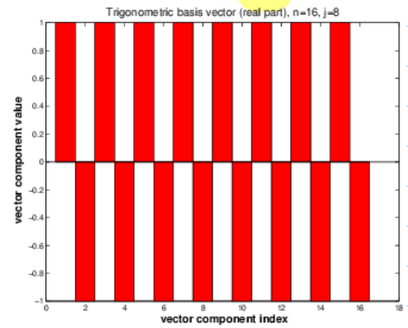
$\mathrm{Re}(\mathbf{F}_{16})_{:,5}$



$\mathrm{Re}(\mathbf{F}_{16})_{:,6}$



$\mathrm{Re}(\mathbf{F}_{16})_{:,7}$

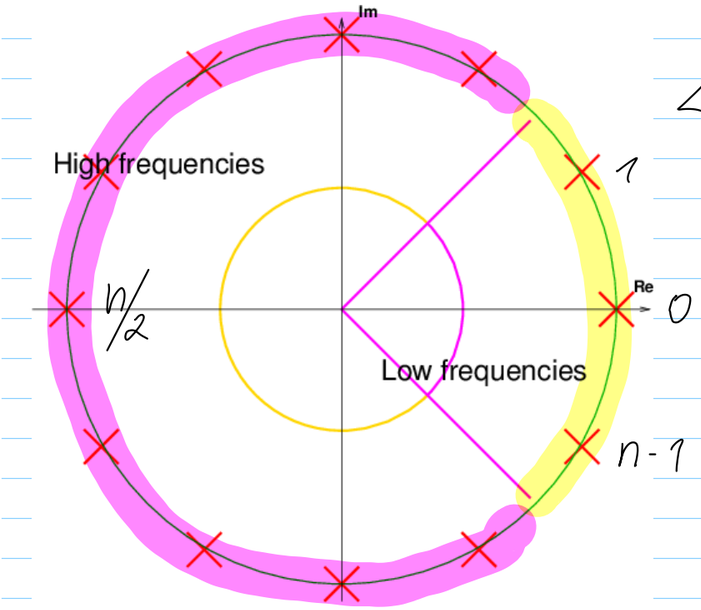


$\mathrm{Re}(\mathbf{F}_{16})_{:,8}$

(4)

$$(F_n)_{:,n-k} = \left[\omega_n^{j(n-k)}\right]_{j=0}^{n-1} = \left[\omega_n^{-jk}\right] = (\overline{F_n})_{:,k}$$

$\uparrow$ $\omega_n^n = 1$ ["wrapped around"]



△ Frequency circle

▷ Frequency filtering:

time-domain signal $\underline{y} \xrightarrow{\text{DFT}} \underline{c} \xrightarrow{\text{"do something"}} \underline{\tilde{c}} \xrightarrow{\text{DFT}^{-1}} \underline{\tilde{y}}$

"do something": low-pass: set $c_j := 0$ for $j : |j - n/2| < k$

high-pass: set $c_j := 0$ for $j : |j - n/2| \geq k$

C++11-code 4.2.3.7: DFT-based frequency filtering → GITLAB

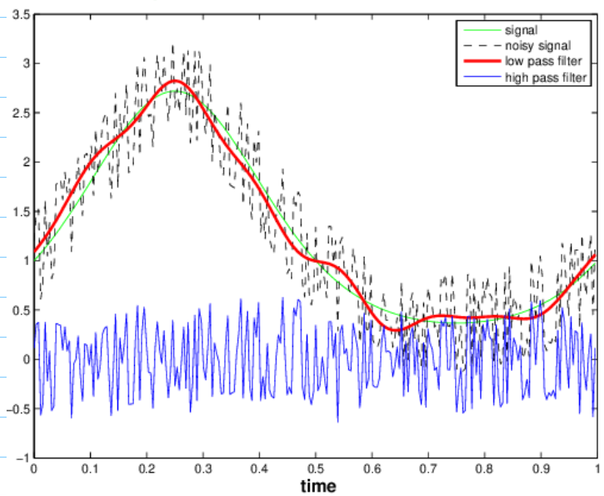
```
void freqfilter(const VectorXd &y, int k, VectorXd &low, VectorXd &high) {
  const VectorXd::Index n = y.size();
  if (n % 2 != 0)
    throw std::runtime_error("Even vector length required!");
  const VectorXd::Index m = y.size() / 2;

  Eigen::FFT<double> fft;  // DFT helper object
  VectorXcd c = fft.fwd(y); // Perform DFT of input vector

  VectorXcd clow = c;
  // Set high frequency coefficients to zero, Fig. 125
  for (int j = -k; j <= +k; ++j) clow(m + j) = 0;
  // (Complementary) vector of high frequency coefficients
  VectorXcd chigh = c - clow;

  // Recover filtered time-domain signals
  low = fft.inv(clow).real();
  high = fft.inv(chigh).real();
}
```

↑ threshold

→ low-pass

→ high-pass

Example 4.2.3.8 (Denoising by frequency filtering)

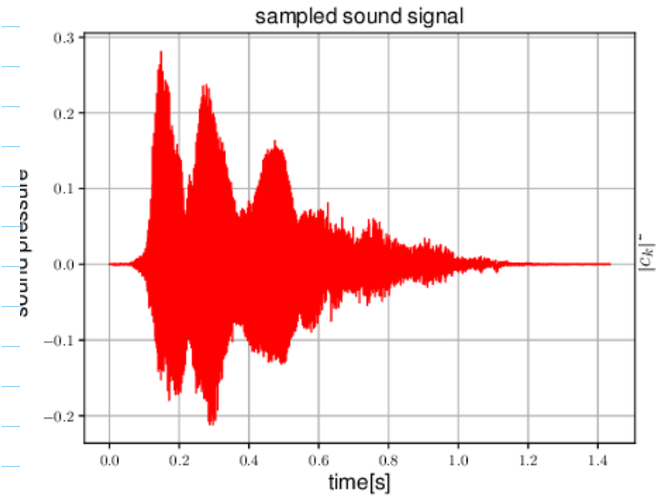


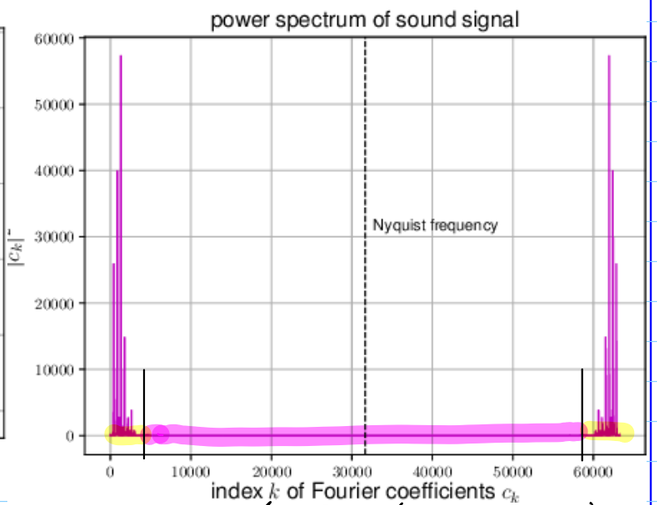
△ Signal + high-frequency noise [$n = 64k$]

→ Low-pass filtering removes noise

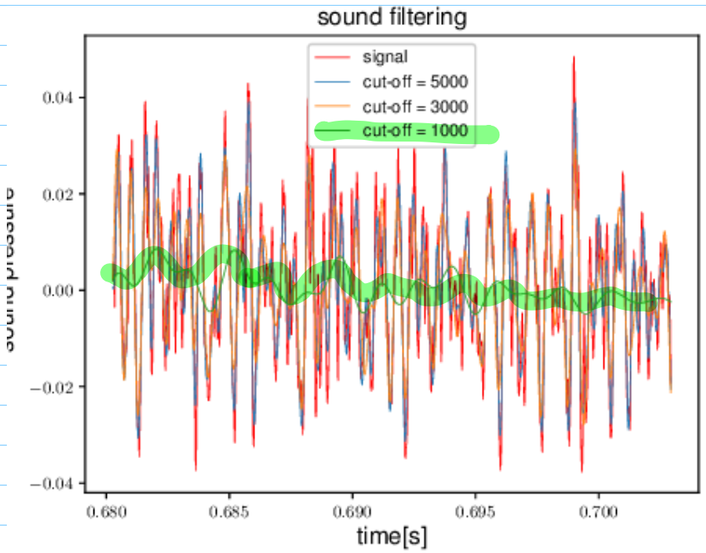
NumCSE@ETHZ

Example 4.2.3.9. (Sound filtering by DFT)





Time domain: human voice → signal information mainly in low frequencies



▷ Compression by low-pass filtering

Remark 4.2.3.10: Linear filtering

Signal $\not\sim$ (in "pulse basis") $\xrightarrow[\text{basis trf.}]{} \subseteq \xrightarrow[\text{operation}]{\text{linear}} \simeq \xrightarrow[\text{trf.}]{\text{inv. basis}} \not\sim$

Review questions 4.2.3.11/4.2.4.6

(Tackle the questions without relying on tablet or lecture notes)

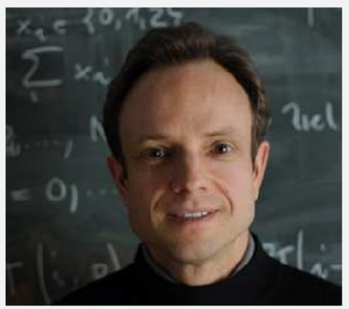
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 4.2.5: Two-Dimensional DFT

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 17, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 4.2.2

Duration: minutes

IV. Filtering Algorithms

4.2. Discrete Fourier Transform (DFT)

4.2.5. Two-Dimensional DFT

2D signals $\longleftrightarrow$ images $\longleftrightarrow$ matrices

§ 4.2.5.1 Matrix Fourier modes

Recall: DFT $\longleftrightarrow$ Interpretation into Fourier-coefficients of modes

basis vectors: $(F_n)_{:,k} = [\omega_n^{kj}]_{j=0}^{n-1}$

Idea: 2D Fourier basis matrices by tensor product

$$\{(F_m)_{:,j}(F_n)_{:,\ell}^T\}_{\substack{j=0,\dots,m-1\\ \ell=0,\dots,n-1}} \subset \mathbb{C}^{m,n}$$

2D DFT of $Y \in \mathbb{C}^{m,n}$: $\underbrace{\mathrm{DFT}_{m,n}(Y)}_{=:C} = \sum_{j=0}^{m-1}\sum_{\ell=0}^{n-1} (Y)_{j,\ell}(F_m)_{:,j}(F_n)_{:,\ell}^T$

[C++-indexing]

(2)

$$C = \sum_{j=0}^{m} \sum_{\ell=0}^{n} (Y)_{j,\ell} (F_m)_{:,j} (F_m)_{:,\ell}^T$$

$$(\mathbf{C})_{k_1,k_2} = \sum_{j_1=0}^{m-1} \sum_{j_2=0}^{n-1} y_{j_1,j_2} \omega_m^{j_1 k_1} \omega_n^{j_2 k_2} = \sum_{j_1=0}^{m-1} \omega_m^{j_1 k_1} \underbrace{\left(\sum_{j_2=0}^{n-1} \omega_n^{j_2 k_2} y_{j_1,j_2} \right)}_{\text{m 1D DFTs of rows of Y}}, \quad 0 \le k_1 < m, 0 \le k_2 < n .$$

n 1D DFT of cols of $DFT_{m,n}(Y)$

$$(\mathbf{C})_{k_1,k_2} = \sum_{j_1=0}^{m-1} (\mathbf{F}_n (\mathbf{Y})_{j_1,:})_{k_2} \omega_m^{j_1 k_1} \quad \blacktriangleright \quad \boxed{\mathbf{C} = \mathbf{F}_m (\mathbf{F}_n \mathbf{Y}^\top)^\top = \mathbf{F}_m \mathbf{Y} \mathbf{F}_n} . \tag{4.2.5.4}$$

Lemma 4.2.1.14. Properties of Fourier matrix

The scaled Fourier-matrix $\frac{1}{\sqrt{n}}\mathbf{F}_n$ is unitary (→ Def. 6.2.1.2) : $\mathbf{F}_n^{-1} = \frac{1}{n}\mathbf{F}_n^{\mathrm{H}} = \frac{1}{n}\overline{\mathbf{F}}_n$.

$$\mathbf{C} = \sum_{j_1=0}^{m-1} \sum_{j_2=0}^{n-1} y_{j_1,j_2} (\mathbf{F}_m)_{:,j_1} (\mathbf{F}_n)_{:,j_2}^\top \quad \Rightarrow \quad \boxed{\mathbf{Y} = \mathbf{F}_m^{-1} \mathbf{C} \mathbf{F}_n^{-1} = \frac{1}{mn} \overline{\mathbf{F}}_m \mathbf{C} \overline{\mathbf{F}}_n} . \tag{4.2.5.5}$$

C++ code 4.2.5.6: Two-dimensional discrete Fourier transform ↪ GITLAB

```
template <typename Scalar>
void fft2(Eigen::MatrixXcd &C, const Eigen::MatrixBase<Scalar> &Y) {
  using idx_t = Eigen::MatrixXcd::Index;
  const idx_t m = Y.rows(),n=Y.cols();
  C.resize(m,n);
  Eigen::MatrixXcd tmp(m,n);

  Eigen::FFT<double> fft; // Helper class for DFT
  // Transform rows of matrix Y
  for (idx_t k=0;k<m;k++) {
    Eigen::VectorXcd tv(Y.row(k));
    tmp.row(k) = fft.fwd(tv).transpose();
  }

  // Transform columns of temporary matrix
  for (idx_t k=0;k<n;k++) {
    Eigen::VectorXcd tv(tmp.col(k));
    C.col(k) = fft.fwd(tv);
  }
}
```

C++ code 4.2.5.7: Inverse two-dimensional discrete Fourier transform ↪ GITLAB

```
template <typename Scalar>
void ifft2(Eigen::MatrixXcd &C, const Eigen::MatrixBase<Scalar> &Y) {
  using idx_t = Eigen::MatrixXcd::Index;
  const idx_t m = Y.rows(),n=Y.cols();
  fft2(C,Y.conjugate()); C = C.conjugate()/(m*n);
}
```

(4)

§ 4.2.5.9. Periodic convolution of matrices

1D:

$$\mathbf{y} = \mathbf{p} *_n \mathbf{x} := \left[\sum_{j=0}^{n-1} x_j \, p_{(k-j) \bmod n} \right]_{k=0}^{n-1}, \quad \mathbf{p}, \mathbf{x} \in \mathbb{C}^n, \tag{4.2.5.10}$$

2D:

$$\mathbf{X} *_{m,n} \mathbf{Y} := \mathsf{B}(\mathbf{X}, \mathbf{Y}) = \left[\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (\mathbf{X})_{i,j} (\mathbf{Y})_{(k-i) \bmod m, (\ell-j) \bmod n} \right]_{\substack{k=0,\ldots,m-1 \\ \ell=0,\ldots,n-1}}, \quad \mathbf{X}, \mathbf{Y} \in \mathbb{C}^{m,n}.$$

→ Diagonal in basis $\{ (F_m)_{:,j} (F_n)_{:,\ell}^T \}_{\substack{j=0,\ldots,m-1 \\ \ell=0,\ldots,n-1}}$

$$\begin{aligned}
(\mathsf{B}(\mathbf{X}, \mathbf{Y}))_{k,\ell} &= \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (\mathbf{X})_{i,j} (\mathbf{Y})_{(k-i) \bmod m, (\ell-j) \bmod n} \\
&= \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (\mathbf{X})_{i,j} \omega_m^{r(k-i)} \omega_n^{s(\ell-j)} \\
&= \left(\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (\mathbf{X})_{i,j} \overline{\omega_m^{ri} \omega_n^{sj}} \right) \cdot \omega_m^{rk} \omega_n^{s\ell}.
\end{aligned}$$

$$\mathsf{B}(\mathbf{X}, \underbrace{(\mathbf{F}_m)_{:,r} (\mathbf{F}_n)_{s,:}}_{\text{"eigenvector"}}) = \underbrace{\left(\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (\mathbf{X})_{i,j} \overline{\omega_m^{ri} \omega_n^{sj}} \right)}_{\text{"eigenvalue", see Eq. (4.2.5.3)}} (\mathbf{F}_m)_{:,r} (\mathbf{F}_n)_{s,:}. \tag{4.2.5.13}$$

≙ an entry of $\mathrm{DFT}_{m,n}(X)$!

1D result:

Theorem 4.2.2.2. Convolution theorem

The discrete periodic convolution $*_n$ between n-dimensional vectors $\mathbf{u}$ and $\mathbf{x}$ is equal to the inverse DFT of the component-wise product between the DFTs of $\mathbf{u}$ and $\mathbf{u}$; i.e.:

$$(\mathbf{u}) *_n (\mathbf{x}) := \left[\sum_{j=0}^{n-1} u_{k-j} x_j \right]_{k=0}^{n-1} = \mathbf{F}_n^{-1} \left[(\mathbf{F}_n \mathbf{u})_j (\mathbf{F}_n \mathbf{x})_j \right]_{j=1}^{n}. \tag{4.2.2.3}$$

Theorem 4.2.5.14. 2D convolution theorem

For any $\mathbf{X}, \mathbf{Y} \in \mathbb{C}^{m,n}$, we have

$$\mathbf{X} *_{m,n} \mathbf{Y} = \mathrm{DFT}_{m,n}^{-1} \left(\mathrm{DFT}_{m,n}(\mathbf{X}) \odot \mathrm{DFT}_{m,n}(\mathbf{Y}) \right),$$

where $\odot$ stands for the entrywise multiplication of matrices of equal size.

C++ code 4.2.5.15: DFT-based 2D discrete periodic convolution → GITLAB

```
// DFT based implementation of 2D periodic convolution
template <typename Scalar1, typename Scalar2, class EigenMatrix>
void pmconv(const Eigen::MatrixBase<Scalar1> &X,
            const Eigen::MatrixBase<Scalar2> &Y, EigenMatrix &Z) {
  using Comp = std::complex<double>;
  using idx_t = typename EigenMatrix::Index;
  using val_t = typename EigenMatrix::Scalar;
  const idx_t n = X.cols(), m = X.rows();
  if ((m != Y.rows()) || (n != Y.cols()))
    throw std::runtime_error("pmconv: size mismatch");
  Z.resize(m, n);
  Eigen::MatrixXcd Xh(m, n), Yh(m, n);
  // Step ❶: 2D DFT of Y
  fft2(Yh, (Y.template cast<Comp>()));
  // Step ❷: 2D DFT of X
  fft2(Xh, (X.template cast<Comp>()));
  // Steps ❸, ❹: inverse DFT of component-wise product
  ifft2(Z, Xh.cwiseProduct(Yh));
}
```

Ex. 4.2.5.16 (Deblurring by DFT)

image $\longleftrightarrow$ $P \in \mathbb{R}^{m,n}$ (pixel matrix)

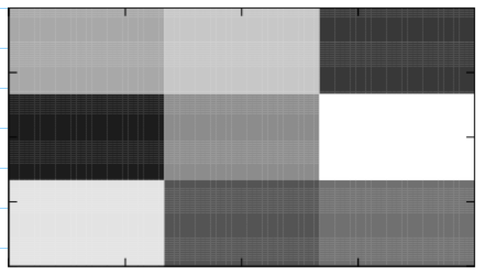
Blurring operator ($L \ll m,n$)

$$\left(C := \mathcal{B}(P)\right)_{l,j} := \sum_{k,q=-L}^{L} s_{k,q} \, (P)_{(l+k) \bmod m,\,(j+q) \bmod n}$$

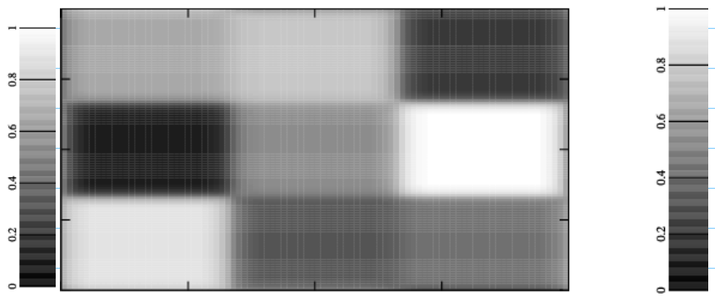
$s_{k,q}$ ↑ PSF

Here: $s_{k,q} = \dfrac{1}{7 + k^2 + q^2}$ & normalized, $L = 5$

Original



Blurred image



$\mathcal{B} \hat{=}$ a 2D discrete periodic convolution

NumCSE@ETHZ

$$(C)_{l,j} = c_{l,j} = \sum_{k=-L}^{L} \sum_{q=-L}^{L} s_{k,q} (\mathbf{P})_{(l+k) \bmod m,\,(j+q) \bmod n}$$

$$= \sum_{k=-L}^{L} \sum_{q=-L}^{L} s_{-k,-q} (\mathbf{P})_{(l-k) \bmod m,\,(j-q) \bmod n}$$

Split in 4 sums:

$$= \sum_{k=0}^{L} \sum_{q=0}^{L} s_{-k,-q} (\mathbf{P})_{(l-k) \bmod m,\,(j-q) \bmod n} +$$

$$\sum_{k=0}^{L} \sum_{q=-L}^{-1} s_{-k,-q} (\mathbf{P})_{(l-k) \bmod m,\,(j-(q+n)) \bmod n} +$$

$$\sum_{k=-L}^{-1} \sum_{q=0}^{L} s_{-k,-q} (\mathbf{P})_{(l-(k+m)) \bmod m,\,(j-q) \bmod n} +$$

$$\mathbf{X} *_{m,n} \mathbf{Y} := \mathsf{B}(\mathbf{X},\mathbf{Y}) = \left[\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (\mathbf{X})_{i,j} (\mathbf{Y})_{(k-i) \bmod m,\,(\ell-j) \bmod n} \right]_{\substack{k=0,\ldots,m-1 \\ \ell=0,\ldots,n-1}}, \quad \mathbf{X},\mathbf{Y} \in \mathbb{C}^{m,n}$$

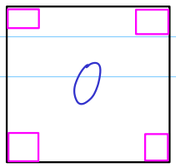
$$= \sum_{k=0}^{L} \sum_{q=0}^{L} s_{-k,-q} (\mathbf{P})_{(l-k) \bmod m,\,(j-q) \bmod n} +$$

$$\sum_{k=0}^{L} \sum_{q=n-L}^{n-1} s_{-k,-q+n} (\mathbf{P})_{(l-k) \bmod m,\,(j-q) \bmod n} +$$

$$\sum_{k=m-L}^{m-1} \sum_{q=0}^{L} s_{-k+m,-q} (\mathbf{P})_{(l-k) \bmod m,\,(j-q) \bmod n} +$$

$$\sum_{k=m-L}^{m-1} \sum_{q=n-L}^{n-1} s_{-k+m,-q+n} (\mathbf{P})_{(l-k) \bmod m,\,(j-q) \bmod n} .$$

⑥



$$\mathbf{C} = \mathbf{S} *_{m,n} \mathbf{P} \quad \text{with} \quad (\mathbf{S})_{k,q} = \begin{cases} s_{-k,-q} & \text{, for } 0 \le k,q \le L\,, \\ s_{-k+m,-q} & \text{, for } m-L \le k < m\,, \quad 0 \le q \le L\,, \\ s_{-k,q+n} & \text{, for } 0 \le k \le L\,, \quad n-L \le q < n\,, \\ s_{-k+m,-q+m} & \text{, for } m-L \le k < m\,, \quad n-L \le q < n\,, \\ 0 & \text{else.} \end{cases} \tag{4.2.5.20}$$

blurred image

Theorem 4.2.5.14. 2D convolution theorem

For any $\mathbf{X}, \mathbf{Y} \in \mathbb{C}^{m,n}$, we have

$$\mathbf{X} *_{m,n} \mathbf{Y} = \mathrm{DFT}_{m,n}^{-1}(\mathrm{DFT}_{m,n}(\mathbf{X}) \odot \mathrm{DFT}_{m,n}(\mathbf{Y}))\,,$$

where $\odot$ stands for the entrywise multiplication of matrices of equal size.

▷ $\mathbf{P} = \mathrm{DFT}_{m,n}^{-1}\left(\mathrm{DFT}_{m,n}(\mathbf{C}) \odot \mathrm{DFT}_{m,n}(\mathbf{S})\right)$ ← entrywise /

C++11-code 4.2.5.22: DFT based deblurring → GITLAB

```
MatrixXd deblur(const MatrixXd &C, const MatrixXd &S, const double tol = 1e-3) {
  const long m = C.rows(), n = C.cols(), M = S.rows(), N = S.cols();
  const long L = (M - 1) / 2;
  if (M != N) {
    throw std::runtime_error("Error: S not quadratic!");
  }
  MatrixXd Spad = MatrixXd::Zero(m, n);
  // Zero padding, see (4.2.5.20)
  Spad.block(0, 0, L + 1, L + 1) = S.block(L, L, L + 1, L + 1);
  Spad.block(m - L, n - L, L, L) = S.block(0, 0, L, L);
  Spad.block(0, n - L, L + 1, L) = S.block(L, 0, L + 1, L);
  Spad.block(m - L, 0, L, L + 1) = S.block(0, L, L, L + 1);
  // Inverse of blurring operator (fft2 expects a complex matrix)
  MatrixXcd SF = fft2(Spad.cast<complex>());
  // Test for invertibility
  if (SF.cwiseAbs().minCoeff() < tol * SF.cwiseAbs().maxCoeff()) {
    std::cerr << "Error: Deblurring impossible!\n";
  }
  // DFT based deblurring
  return fft2(ifft2(C.cast<complex>()).cwiseQuotient(SF)).real();
}
```

} build $S \in \mathbb{C}^{m,n}$

What could be a problem

Check for $(\mathrm{DFT}_{m,n}(S))_{l,k} \neq 0$

[in a numerically clean way]

Review questions 4.2.5.23

(Try to answer them without looking up formulas in the lecture/tablet notes)

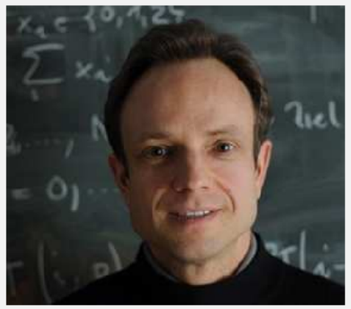
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 4.3: Fast Fourier Transform (FFT)

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 18, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Def. 4.2.1.18

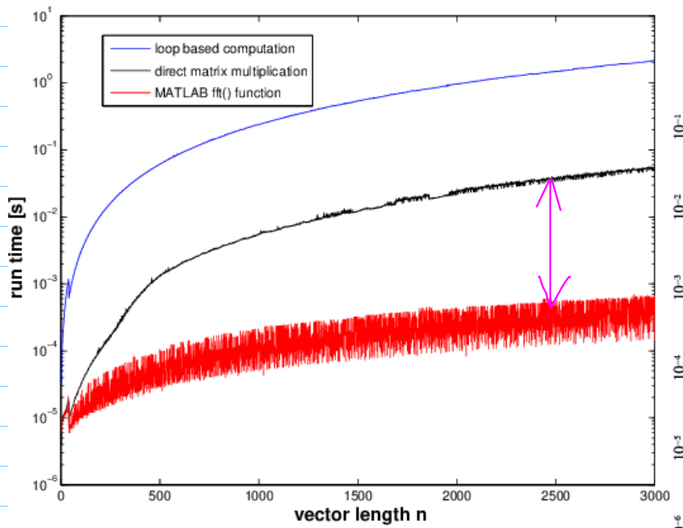
Duration: 30 minutes

IV. Filtering Algorithms

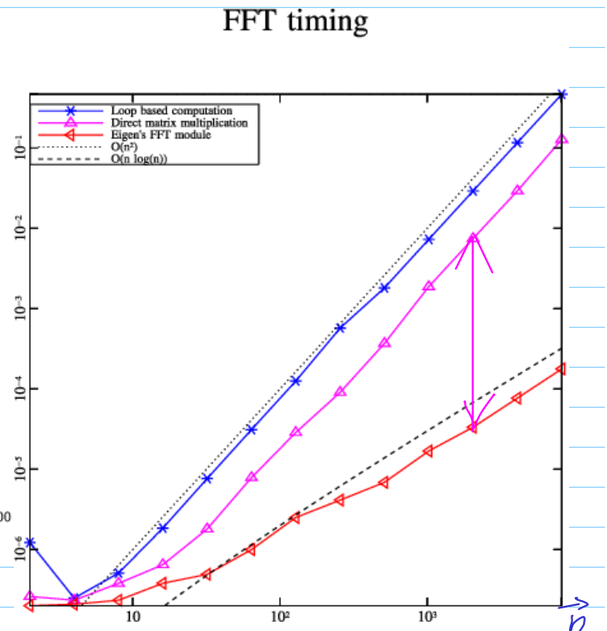
4.3. Fast Fourier Transform (FFT)

DFT : $c = F_n y$: cost $= O(n^2)$ for $n \to \infty$

Exp 4.3.0.2 : Runtimes of DFT implementations



MATLAB



Eigen (Default)

(2)

The secret of MATLAB's/EIGEN's/PYTHON's `fft()`:
the Fast Fourier Transform (FFT) algorithm

(discovered by C.F. Gauss in 1805, rediscovered by Cooley & Tuckey in 1965,
one of the "top ten algorithms of the century").

§ 4.3.0.3 : FFT : Derivation & Complexity

Consider $n = 2m$:

even/odd splitting

$$c_k = \sum_{j=0}^{n-1} y_j e^{-\frac{2\pi i}{n}jk} = \sum_{j=0}^{m-1} y_{2j} e^{-\frac{2\pi i}{n}2jk} + \sum_{j=0}^{m-1} y_{2j+1} e^{-\frac{2\pi i}{n}(2j+1)k}$$

$$= \underbrace{\sum_{j=0}^{m-1} y_{2j} \underbrace{e^{-\frac{2\pi i}{m}jk}}_{=\omega_m^{jk}}}_{=:\widetilde{c}_k^{\text{even}}} + e^{-\frac{2\pi i}{n}k} \cdot \underbrace{\sum_{j=0}^{m-1} y_{2k+1} \underbrace{e^{-\frac{2\pi i}{m}jk}}_{=\omega_m^{jk}}}_{=:\widetilde{c}_k^{\text{odd}}} \quad k \in \mathbb{Z} . \qquad (4.3.0.4)$$

$\widetilde{c}^{\text{even}}$ ↑

▷ $\underline{\widetilde{c}}^{\text{even}} = \mathrm{DFT}_m [y_{2j}]_{j=0}^{m-1}$

$\underline{\widetilde{c}}^{\text{odd}} = \mathrm{DFT}_m [y_{2j+1}]_{j=0}^{m-1}$

Idea for $n = 2^L$: "Divide & conquer" recursion

C++ code 4.3.0.5: Recursive FFT ⇝ GITLAB

```
2  // Recursive DFT for vectors of length n = 2^L
3  VectorXcd fftrec(const VectorXcd& y) {
4    using idx_t = VectorXcd::Index;
5    using comp = std::complex<double>;
6    // Nothing to do for DFT of length 1
7    const idx_t n = y.size();
8    if (n == 1) return y;
9    if (n % 2 != 0) throw std::runtime_error("size(y) must be even!");
10   const Eigen::Map<const
         Eigen::Matrix<comp, Eigen::Dynamic, Eigen::Dynamic, Eigen::RowMajor>>
11     Y(y.data(),n/2,2); // for selecting even and off compoents
12   const VectorXcd c1 = fftrec(Y.col(0)),c2 = fftrec(Y.col(1));
13   const comp omega = std::exp(-2*M_PI/n*comp(0,1)); // root of unity ω_n
14   comp s(1.0,0.0);
15   VectorXcd c(n);
16   // Scaling of DFT of odd components plus periodic copying
17   for (long k = 0; k < n; ++k) {
18     c(k) = c1(k%(n/2)) + c2(k%(n/2))*s;
19     s *= omega;
20   }
21   return c;
22 }
```

cost = O(n)

$c1 \leftrightarrow \widetilde{c}^{\text{even}}$

$c2 \leftrightarrow \widetilde{c}^{\text{odd}}$

Computational cost ?

Visualization of computational cost of `fftrec()` from Code 4.3.0.5:

NumCSE@ETHZ

▷ $O(2^L)$ ops. on every level

▷ cost $= O(2^L \cdot L) =$ $O(n \cdot \log n)$ for $n \to \infty$

"$\ll O(n^2)$"

Beyond $n = 2^L$:

Rem. 4.3.0.8 : FFT for $n = pq$

Re-indexing: generalizes even/odd splitting $\omega_q^q = 1$

$$c_k = \sum_{j=0}^{n-1} y_j \omega_n^{jk} \overset{[j=:lp+m]}{=} \sum_{m=0}^{p-1} \sum_{l=0}^{q-1} y_{lp+m} e^{-\frac{2\pi i}{pq}(lp+m)k} = \sum_{m=0}^{p-1} \omega_n^{mk} \underbrace{\sum_{l=0}^{q-1} y_{lp+m} \omega_q^{l(k \bmod q)}}_{\leftrightarrow \mathrm{DFT}_q}. \quad (4.3.0.9)$$

I : $z_m := \mathrm{DFT}_q [y_{lp+m}]_{l=0}^{q-1}$: p-times

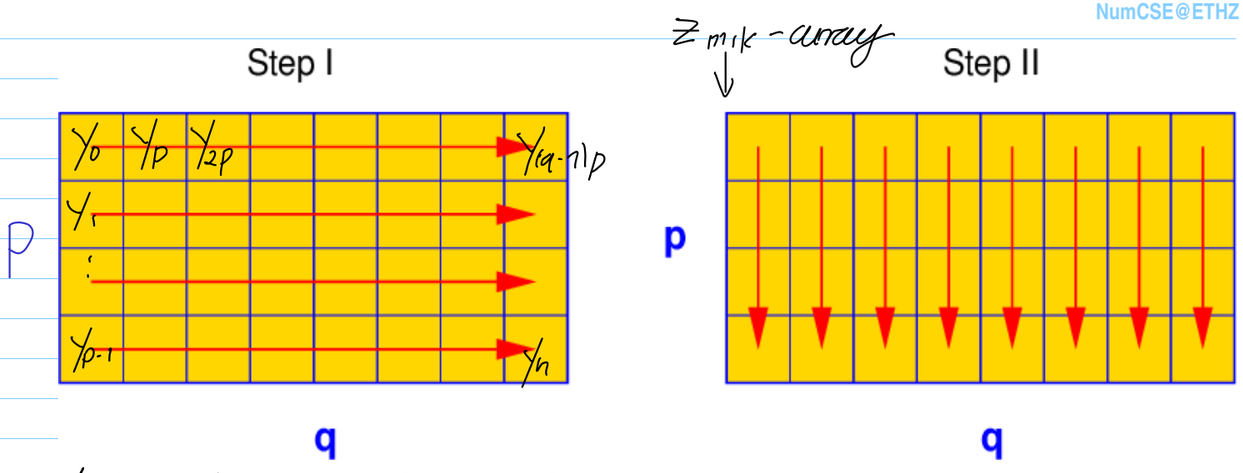
$$(\mathbf{z}_m)_k = z_{m,k} := \sum_{l=0}^{q-1} y_{lp+m} \omega_q^{lk}, \quad 0 \le m < p, \quad 0 \le k < q.$$

II

n multiplications

$$c_{rq+s} = \sum_{m=0}^{p-1} e^{-\frac{2\pi i}{pq}(rq+s)m} z_{m,s} = \underbrace{\sum_{m=0}^{p-1} (\omega_n^{ms} z_{m,s}) \omega_p^{mr}}_{\leftrightarrow \mathrm{DFT}_p},$$

$k = rq+s$: $0 \le r < p$, $0 \le s < q$



Related to reduction of 2D DFT of $C^{m,n}$ to $m \times \mathrm{DFT}_n + n \times \mathrm{DFT}_m$!

Special treatment of prime n :

The execution time for fft depends on the length of the transform. It is fastest for powers of two. It is almost as fast for lengths that have only small prime factors. It is typically several times slower for lengths that are prime or which have large prime factors → Ex. 4.3.0.12.

Never implement DFT/FFT by yourself!

Under all circumstances use high-quality numerical libraries!

Review questions 4.3.0.13

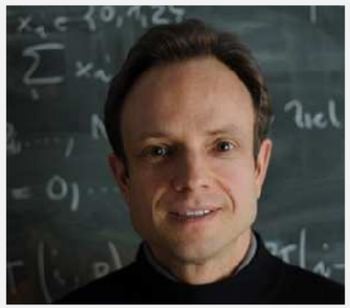
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 4.5: Toeplitz Matrix Techniques

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 18, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

-

Dependencies. Section 3.1.2, , Section 4.1.4, Section 4.2.1

Duration: 35 minutes

IV. Filtering Algorithms

4.5. Toeplitz Matrix Techniques

4.5.1. Matrices with Constant Diagonals

Definition 4.5.1.7. Toeplitz matrix

$\mathbf{T} = (t_{ij})_{i,j=1}^{n} \in \mathbb{K}^{m,n}$ is a Toeplitz matrix, if there is a vector $\mathbf{u} = [u_{-m+1}, \ldots, u_{n-1}] \in \mathbb{K}^{m+n-1}$ such that $t_{ij} = u_{j-i}$, $1 \le i \le m$, $1 \le j \le n$.

$$\mathbf{T} = \begin{bmatrix} u_0 & u_1 & \cdots & \cdots & u_{n-1} \\ u_{-1} & u_0 & u_1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & u_1 \\ \vdots & & \ddots & \ddots & u_0 \\ u_{1-m} & \cdots & \cdots & u_{-1} & u_0 \end{bmatrix}$$

▷ $m+n-1$ numbers to describe an $m \times n$ Toeplitz matrix: data-sparse

Recall: LT-FIR channels
(linear, time-invariant, finite, causal)

If $(h_0, \dots, h_{n-1})$ = impulse response

▷ matrix model: $(m > n)$

$$\mathbf{y} := \begin{bmatrix} y_0 \\ \vdots \\ y_{m+n-2} \end{bmatrix} = \begin{bmatrix} h_0 & 0 & \cdots & 0 \\ h_1 & & & \\ & \ddots & & \\ h_{n-1} & & & h_0 \\ 0 & & & h_1 \\ & \ddots & & \\ 0 & \cdots & 0 & h_{n-1} \end{bmatrix} \begin{bmatrix} x_0 \\ \vdots \\ x_{m-1} \end{bmatrix} =: \mathbf{C}\mathbf{x} . \qquad (4.1.3.2)$$

output signal

Toeplitz matrix

input signal

NumCSE@ETHZ

Ex. 4.5.1.1: Parameter identification for LT-FIR channels

Given: • m-periodic input signal (x_k)
• m-periodic output signal (y_k)

(both measured ⇒ affected by measurement errors)

$m > n$

cf. (4.1.2.4)

$$\exists \mathbf{h} = [h_0, \dots, h_{n-1}]^\top \in \mathbb{R}^n, \quad n \le m : \quad y_k = \sum_{j=0}^{n-1} h_j x_{k-j} . \qquad (4.5.1.2)$$

$k = 0, \dots, m-1$

impulse response: to be estimated

↳ overdetermined LSE: least squares estimator

$$\|\mathbf{A}\mathbf{h} - \mathbf{y}\|_2 = \left\| \begin{bmatrix} x_0 & x_{-1} & \cdots & \cdots & x_{1-n} \\ x_1 & x_0 & x_{-1} & & \vdots \\ \vdots & x_1 & x_0 & \ddots & \\ & & \ddots & \ddots & x_{-1} \\ x_{n-1} & & & x_1 & x_0 \\ x_n & x_{n-1} & & & x_1 \\ \vdots & & & & \vdots \\ x_{m-1} & \cdots & & \cdots & x_{m-n} \end{bmatrix} \begin{bmatrix} h_0 \\ \vdots \\ h_{n-1} \end{bmatrix} - \begin{bmatrix} y_0 \\ \vdots \\ y_{m-1} \end{bmatrix} \right\|_2 \to \min .$$

③ Normal equations: $A^TA\underline{h} = A^T\underline{y}$

$$(A^TA)_{i,j} = \sum_{k=0}^{m-1} x_{k-i}\,x_{k-j} \overset{(x_k)\ m\text{-periodic}}{=} \sum_{k=0}^{m-1} x_k\,x_{k-(j-i)} = z_{i-j}$$

$(A^TA)_{i,j}$ is a Toeplitz matrix

4.5.2. Toeplitz Matrix Arithmetic

Compute $T\underline{x}$, $\underline{x}\in\mathbb{R}^n$, $T\in\mathbb{R}^{m,n}$ a Toeplitz matrix

$$T = [u_{j-i}] \in \mathbb{R}^{m,n}$$

$(u_{-m+1},\dots,u_{n-1}) \triangleq$ defining sequence

Idea: Embedding into a circulant matrix

T as sub-block of $(m+n)\times(m+n)$ circulant matrix

For $m = n$:

$$\mathbf{C} = \left[\begin{array}{ccccc|ccccc}
u_0 & u_1 & \cdots & \cdots & u_{n-1} & 0 & u_{1-n} & \cdots & \cdots & u_{-1}\\
u_{-1} & u_0 & u_1 & \ddots & \vdots & u_{n-1} & 0 & \ddots & \ddots & \vdots\\
\vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & & \\
\vdots & & \ddots & \ddots & u_1 & \vdots & & \ddots & \ddots & u_{1-n}\\
u_{1-n} & \cdots & \cdots & u_{-1} & u_0 & u_1 & & & u_{n-1} & 0\\
\hline
0 & u_{1-n} & \cdots & \cdots & u_{-1} & u_0 & u_1 & \cdots & \cdots & u_{n-1}\\
u_{n-1} & 0 & \ddots & & \vdots & u_{-1} & u_0 & u_1 & & \vdots\\
\vdots & \ddots & \ddots & & & \vdots & \ddots & \ddots & \ddots & \vdots\\
 & & & \ddots & & \vdots & & \ddots & \ddots & \vdots\\
\vdots & & \ddots & \ddots & u_{1-n} & \vdots & & & \ddots & u_1\\
u_1 & & & u_{n-1} & 0 & u_{1-n} & \cdots & \cdots & u_{-1} & u_0
\end{array}\right]$$

spanned by $(m+n)$-periodic seq. $(u_0,\dots,u_{1-n},0,u_{n-1},\dots,u_1)$

$$\triangleright\quad \underbrace{C\begin{bmatrix}\underline{x}\\0\end{bmatrix}}_{\triangleq\ \text{discrete periodic convolution; DFT via FFT}} = \begin{bmatrix}Tx\\*\end{bmatrix}$$

$\triangleright$ Cost $= O((m+n)\log(m+n))$ for $m,n\to\infty$

4.5.3. The Levinson Algorithm

$T = [a_{j-i}] \in \mathbb{R}^{n,n}$ Toeplitz, s.p.d., $a_0 = 1$

Solve LSE $T\underline{x} = \underline{b} = [b_1, \dots, b_n]^T \in \mathbb{R}^n$

Idea: Recursion based on block partitioning

$T_k = [a_{j-i}]_{i,j=0}^{k-1} \in \mathbb{R}^{k,k}$, s.p.d. Toeplitz

$\underline{a}^k = [a_1, \dots, a_k]^T \in \mathbb{R}^k$ $\quad 1 \le k \le n$

$\underline{b}^k = [b_1, \dots, b_k]^T \in \mathbb{R}^k$

$\underline{x}^k \in \mathbb{R}^k$: $\quad T_k \underline{x}^k = \underline{b}^k$

$(k,1)$-partitioned LSE ↗

$$\mathbf{T}_{k+1}\mathbf{x}^{k+1} = \left[\begin{array}{c|c} \mathbf{T}_k & \begin{matrix} a_k \\ \vdots \\ a_1 \end{matrix} \\ \hline \begin{matrix} a_k & \cdots & a_1 \end{matrix} & 1 \end{array}\right] \left[\begin{array}{c} \tilde{\mathbf{x}}^{k+1} \\ \hline x_{k+1}^{k+1} \end{array}\right] = \left[\begin{array}{c} \begin{matrix} b_1 \\ \vdots \\ b_k \end{matrix} \\ \hline b_{k+1} \end{array}\right] = \left[\begin{array}{c} \mathbf{b}^k \\ b_{k+1} \end{array}\right] = \underline{b}^{k+1} \qquad (4.5.3.1)$$

T^{k+1} $\quad \underline{x}^{k+1}$

Next: block elimination

NumCSE@ETHZ

reverse()-permutation ↓

$$\begin{cases} \tilde{\mathbf{x}}_{k+1} = \mathbf{T}_k^{-1}(\mathbf{b}^k - x_{k+1}^{k+1}\mathsf{P}_k\mathbf{a}^k) = \mathbf{x}^k - x_{k+1}^{k+1}\mathbf{T}_k^{-1}\mathsf{P}_k\mathbf{a}^k, \\ x_{k+1}^{k+1} = b_{k+1} - \mathsf{P}_k\mathbf{a}^k \cdot \tilde{\mathbf{x}}^{k+1} = b_{k+1} - \left(\mathsf{P}_k\mathbf{a}^k\right)^\top \mathbf{x}^k + x_{k+1}^{k+1}\left(\mathsf{P}_k\mathbf{a}^k\right)^\top \underbrace{\mathbf{T}_k^{-1}\mathsf{P}_k\mathbf{a}^k}_{\mathbf{y}^k}. \end{cases} \qquad (4.5.3.3)$$

$$\mathbf{x}^{k+1} = \begin{bmatrix} \tilde{\mathbf{x}}^{k+1} \\ x_{k+1}^{k+1} \end{bmatrix} \text{ with } \begin{matrix} x_{k+1}^{k+1} = (b_{k+1} - \mathsf{P}_k\mathbf{a}^k)/\sigma_k \\ \tilde{\mathbf{x}}^{k+1} = \mathbf{x}^k - x_{k+1}^{k+1}\mathbf{y}^k \end{matrix}, \quad \sigma_k := 1 - \mathsf{P}_k\mathbf{a}^k \cdot \mathbf{y}^k. \qquad (4.5.3.4)$$

$\neq 0$, if T_k s.p.d.

$T_k \mathbf{y}^k = \mathsf{P}_k \mathbf{a}^k$

Two recursive calls? → Too expensive

Idea: If $\underline{b} = \mathsf{P}_n \mathbf{a}^n \Rightarrow \underline{x}^k = \mathbf{y}^k$

→ Simple linear recursion: $\mathbf{y}^k$ with cost $= O(k^2)$

Then compute $\underline{x}^k$: another simple linear rec.

cost $= O(k^2)$, for $k \to \infty$

→ Review questions 4.5.3.8

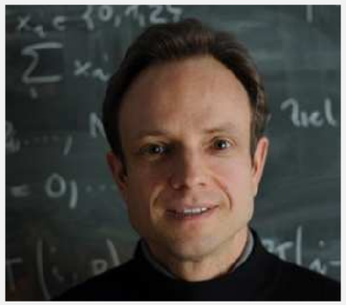
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 5.1: Abstract Interpolation

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 27, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- Linear systems of equations
- Notion of continuous functions

Duration: 26 minutes

Note: Possible minor *mismatch of video and tablet notes*!

[Corrections and updates can be incorporated into tablet notes only]

V. Machine Learning of One-Dimensional Data (Data Interpolation and Data Fitting in 1D)

5.1. Abstract Interpolation (AI)

One-dimensional interpolation

Given: data points (t_i, y_i), $i = 0, \ldots, n$, $n \in \mathbb{N}$, $t_i \in I \subset \mathbb{R}$, $y_i \in \mathbb{R}$

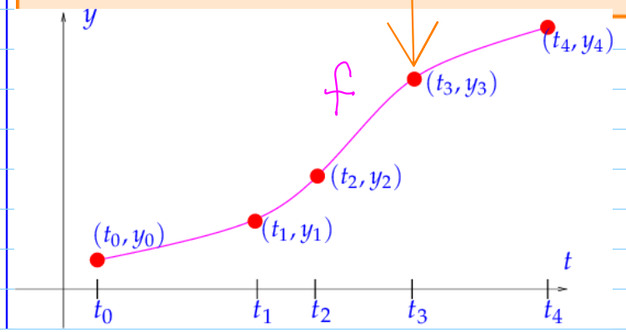
Objective: *Reconstruction of a function* $f : I \mapsto \mathbb{R}$

- satisfying the $n+1$ **interpolation conditions** (IC) → defines smoothness requirements etc.

$$f(t_i) = y_i, \quad i = 0, \ldots, n \,. \tag{5.1.0.2}$$

- and belonging to a set V of eligible functions.

The function f we find is called the **interpolant** of the given data set $\{(t_i, y_i)\}_{i=0}^{n}$.



$t_i \hat{=}$ nodes

$y_i \hat{=}$ values

(2)

Ex 5.1.0.8 : Constitutive relations from measurements

t	y
voltage U	current I
pressure p	density ρ
magnetic field H	magnetic flux B
...	...

Examples: t and y could be

accurate ⇕ IC

Functional relation $y = f(t)$ needed for numerical

Rem. 5.1.0.9 / 5.1.0.11 : Functions in a code

→ OO-approach : Functor object

C++-code 5.1.0.12: C++ class representing an interpolant in 1D

```
class Interpolant {
  private:
    // Various internal data describing f
    // Can be the coefficients of a basis representation (5.1.0.14)
  public:
    // Constructor:  computation of coefficients c_j of representation
       (5.1.0.14)
    Interpolant(const vector<double>& t, const vector<double>& y);
    // Evaluation operator for interpolant f
    double operator() (double t) const;
};
```

→ f(t)

↳ Issue : Efficient implementation

build an internal representation of f

Special case

$V = \mathrm{Span}\underbrace{\{\{t \mapsto b_0(t)\}, \ldots, \{t \mapsto b_m(t)\}\}}_{\text{a basis}} = \mathrm{Span}\{b_0, \ldots, b_m\}$ ⇒ $\dim V = m+1$

$\forall f \in V \Leftrightarrow \exists \alpha_j \in \mathbb{R} : f(t) = \sum_{j=0}^{n} \alpha_j b_j(t)$ (5.1.0.14)

↳ basis expansion coefficients

IC : $f(t_i) = y_i$

(5.1.0.2) & (5.1.0.14) ⇒ $f(t_i) = \sum_{j=0}^{m} \alpha_j b_j(t_i) = y_i\,, \quad i = 0, \ldots, n\,,$ (5.1.0.22)

⇕

$$\mathbf{Ac} := \begin{bmatrix} b_0(t_0) & \ldots & b_m(t_0) \\ \vdots & & \vdots \\ b_0(t_n) & \ldots & b_m(t_n) \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \vdots \\ \alpha_m \end{bmatrix} = \begin{bmatrix} y_0 \\ \vdots \\ y_n \end{bmatrix} =: \mathbf{y}\,. \tag{5.1.0.23}$$

This is an $(m+1) \times (n+1)$ linear system of equations !

Unique solution $\forall y$ only if $m = n$

$m = n$, A regular ⇒ $f(t) = \sum_{j=0}^{n} (A^{-1}y)_j b_j(t)$

▷ Linear interpolation operator

↳ as a mapping

$$\begin{cases} \mathbb{R}^{n+1} \longrightarrow V \\ y \longmapsto f \end{cases}$$

continuous function $[t_0, t_m] \to \mathbb{R}$

Definition 5.1.0.25. Linear interpolation operator

An interpolation operator $\mathsf{I} : \mathbb{R}^{n+1} \mapsto C^0([t_0, t_m])$ for the given nodes $t_0 < t_1 < \cdots < t_n$ is called linear, if

$$\mathsf{I}(\alpha \mathbf{y} + \beta \mathbf{z}) = \alpha \mathsf{I}(\mathbf{y}) + \beta \mathsf{I}(\mathbf{z}) \quad \forall \mathbf{y}, \mathbf{z} \in \mathbb{R}^{n+1},\ \alpha, \beta \in \mathbb{R}\,. \tag{5.1.0.26}$$

When will $A = I$?

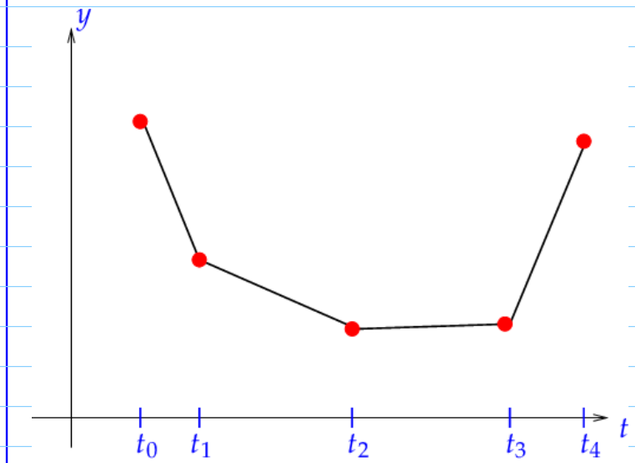
$[(A)_{ij} = b_j(t_i)]$ ⇕

$b_i(t_j) = \delta_{ij}$: a cardinal basis

↳ $f = \sum_{j=0}^{n} y_j b_j$

NumCSE@ETHZ

Ex. 5.1.0.15 : Piecewise linear interpolation

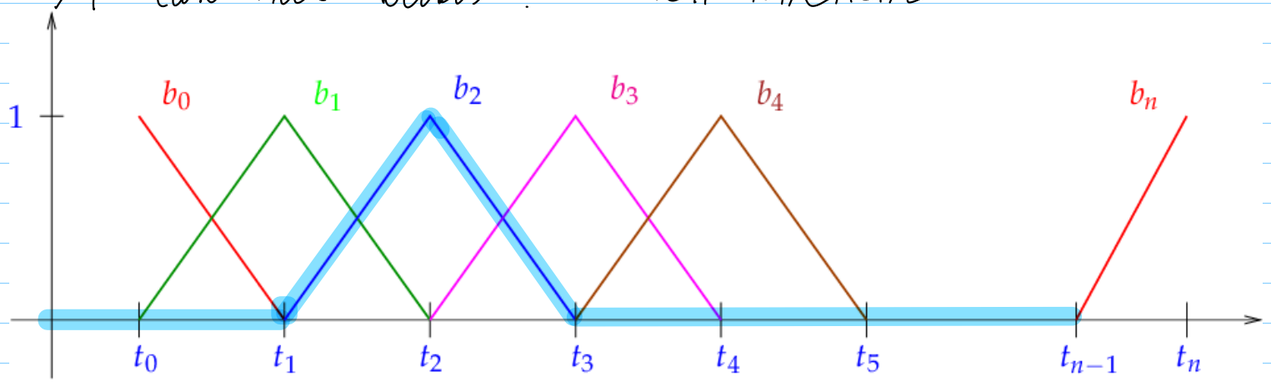


Graph of interpolant ⇕ polygon through data points

$V = \{ f \in C^0([t_0, t_m]) : \underbrace{f(t) = \alpha_i + \beta_i t}_{\text{on } [t_i, t_{i+1}],\ i = 0, \ldots, n-1},\ \alpha_i, \beta_i \in \mathbb{R} \}$ (5.1.0.16)

↑ dim = m+1

A cardinal basis : "tent functions"



$b_2(t_2) = 1$

NumCSE@ETHZ

▷ Review questions 5.1.0.27

(Try to answer without consulting notes)

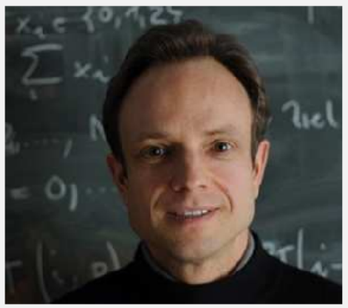
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 5.2.1: Uni-Variate Polynomials

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 27, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- Concept of a basis from linear algebra

Duration: 10 minutes

V. Data Interpolation and Data Fitting in 1D

5.2. Global Polynomial Interpolation

5.2.1. Uni-Variate Polynomials

Notation: Vector space of the **(uni-variate) polynomials** of **degree** $\leq k$, $k \in \mathbb{N}$:

Now $V := \mathcal{P}_k := \{t \mapsto \alpha_k t^k + \alpha_{k-1} t^{k-1} + \cdots + \alpha_1 t + \alpha_0 \cdot 1, \alpha_j \in \mathbb{R}\}$. (5.2.1.1)

leading coefficient

↓ monomial representation

$\{t \mapsto t^\ell\}_{\ell=0}^{k}$ ≙ monomial basis of $\mathcal{P}_k$

⇒ $\dim \mathcal{P}_k = k+1$

(2)

§ 5.2.1.3 : The charms of polynomials

- Vector space & pol. * pol. → pol.
- easy to derive & integrate
- Taylor pols. ⇒ approximation of fcts.
- easy to evaluate

↓

§ 5.2.1.5. Horner scheme

→ $p \in \mathcal{P}_n$ in monomial representation

$p(t)$ with cost = $O(n)$

$$p(t) = t(\cdots t(t(\alpha_n t + \alpha_{n-1}) + \alpha_{n-2}) + \cdots + \alpha_1) + \alpha_0 . \qquad (5.2.1.6)$$

C++-code 5.2.1.7: Horner scheme (vectorized version)

```
// Efficient evaluation of a polynomial in monomial representation
// using the Horner scheme (5.2.1.6)
// IN: p = vector of monomial coefficients, length = degree + 1
// (leading coefficient in p(0), PYTHON convention Rem. 5.2.1.4)
// t = vector of evaluation points t_i
// OUT: vector of values: polynomial evaluated at t_i
Eigen::VectorXd horner(const Eigen::VectorXd &p, const Eigen::VectorXd &t) {
  const VectorXd::Index n = t.size();
  Eigen::VectorXd y{p[0] * VectorXd::Ones(n)};
  for (unsigned i = 1; i < p.size(); ++i)
    y = t.cwiseProduct(y) + p[i] * VectorXd::Ones(n);
  return y;
}
```

Convention: $p = [\alpha_n \; \alpha_{n-1} \; \ldots \; \alpha_0]^T \in \mathbb{R}^{n+1}$

▷ Review questions 5.2.1.8

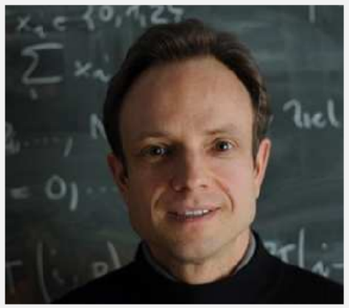
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 5.2.2: Polynomial Interpolation: Theory

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 27, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.
- Theory of linear systems of equations from linear algebra

Dependencies. Section 5.2.1

Duration: 10 minutes

Note: Possible minor *mismatch of video and tablet notes*!

[Corrections and updates can be incorporated into tablet notes only]

V. Data Interpolation and Data Fitting in 1D

5.2. Global Polynomial Interpolation

5.2.2. Polynomial Interpolation: Theory

One-dimensional interpolation

Given: data points (t_i, y_i), $i = 0, \ldots, n$, $n \in \mathbb{N}$, $t_i \in I \subset \mathbb{R}$, $y_i \in \mathbb{R}$

Objective: *Reconstruction of a function* $f : I \mapsto \mathbb{R}$
- satisfying the $n+1$ **interpolation conditions**

$$f(t_i) = y_i, \quad i = 0, \ldots, n \,. \tag{5.1.0.2}$$

- and belonging to a set V of eligible functions.

The function f we find is called the **interpolant** of the given data set $\{(t_i, y_i)\}_{i=0}^{n}$.

$V = \mathcal{P}_n$:

Lagrange polynomial interpolation problem

Given the set of interpolation nodes $\{t_0, \ldots, t_n\} \subset \mathbb{R}$, $n \in \mathbb{N}$, and the values $y_0, \ldots, y_n \in \mathbb{R}$ compute $p \in \mathcal{P}_n$ such that it satisfies the interpolation conditions (IC)

$$p(t_j) = y_j \quad \text{for} \quad j = 0, \ldots, n \,. \tag{5.2.2.2}$$

(2)

Tool: A cardinal basis $\{L_i\}_{i=0}^n$ of $\mathcal{P}_n$

$$L_i(t_j) = \delta_{ij} \quad , \; i,j \in \{0,\dots,n\}$$

$$L_i(t) = \frac{(t-t_0)\cdot\dots\cdot(t-t_{i-1})(t-t_{i+1})\cdot\dots\cdot(t-t_n)}{(t_i-t_0)\cdot\dots\cdot(t_i-t_{i-1})(t_i-t_{i+1})\cdot\dots\cdot(t_i-t_n)}$$

↳ Lagrange polynomial $\in \mathcal{P}_n$

$$\Rightarrow \text{interpolant} : \quad p(t) = \sum_{i=0}^{n} y_i L_i(t)$$

$$\Rightarrow \begin{cases} \mathcal{P}_n \longrightarrow \mathbb{R}^{n+1} \\ p \to (p(t_j))_{j=0}^n \end{cases} \quad \text{is surjective} \quad (\dim = 0)$$

$\Downarrow$

Uniqueness

Theorem 5.2.2.7. Existence & uniqueness of Lagrange interpolation polynomial →,

The general Lagrange polynomial interpolation problem admits a unique solution $p \in \mathcal{P}_n$ for any (t_i, y_i).

▷ Review questions 5.2.2.19

(Please answer without aids)

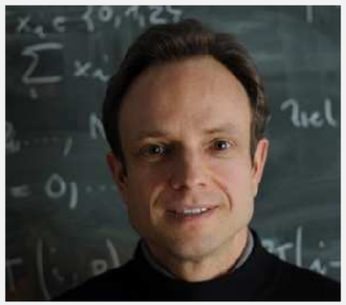
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 5.2.3: Polynomial Interpolation: Algorithms

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 27, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.
- Knowledge of C++

Dependencies. Section 5.1, Section 5.2.1, Section 5.2.2

Duration: 30 minutes

V. Data Interpolation and Data Fitting in 1D

5.2. Global Polynomial Interpolation

5.2.3. Polynomial Interpolation: Algorithms

Goal : Functor class

C++-code 5.1.0.12: C++ class representing an interpolant in 1D

```
1
2  class Interpolant {
3    private:
4      // Various internal data describing f
5      // Can be the coefficients of a basis representation (5.1.0.14)
6    public:
7      // Constructor:  computation of coefficients c_j of representation
           (5.1.0.14)
8      Interpolant(const vector<double>& t, const vector<double>& y);
9      // Evaluation operator for interpolant f
10     double operator() (double t) const;
11 };
```

→ Adapt to different use scenarios

(2)

5.2.3.1. Multiple Evaluations

Task: For ❶ a *fixed* set $\{t_0, \ldots, t_n\}$ of nodes,
and ❷ *many* different given data values y_i, $i = 0, \ldots, n$
and ❸ *many* arguments x_k, $k = 1, \ldots, N$, $N \gg 1$,
efficiently compute all $p(x_k)$ for $p \in \mathcal{P}_n$ interpolating in (t_i, y_i), $i = 0, \ldots, n$.

C++-code 5.2.3.1: Polynomial Interpolation

```
class PolyInterp {
  private:
    // various internal data describing p
    Eigen::VectorXd t;
  public:
    // Constructors taking node vector (t_0,...,t_n) as argument
  PolyInterp(const Eigen::VectorXd &_t);
  template <typename SeqContainer>
           PolyInterp(const SeqContainer &v);
  // Evaluation operator for data (y_0,...,y_n); computes
  // p(x_k) for x_k's passed in x
  Eigen::VectorXd eval(const Eigen::VectorXd &y,
                       const Eigen::VectorXd &x) const;
};
```

Straightforward: $p(x_k) = \sum_{i=0}^{n} y_i L_i(x_k)$

$L_i(t) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{(t - t_j)}{(t_i - t_j)}$ ▷ Cost $= O(n^2 N)$ for $n, N \to \infty$

Idea: Precompute parts of L_i in the constructor!

$$p(t) = \sum_{i=0}^{n} L_i(t)\, y_i = \sum_{i=0}^{n} \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{t - t_j}{t_i - t_j} y_i = \sum_{i=0}^{n} \lambda_i \prod_{\substack{j=0 \\ j \neq i}}^{n} (t - t_j)\, y_i = \prod_{j=0}^{n} (t - t_j) \cdot \sum_{i=0}^{n} \frac{\lambda_i}{t - t_i} y_i ,$$

where $\lambda_i = \dfrac{1}{(t_i - t_0) \cdots (t_i - t_{i-1})(t_i - t_{i+1}) \cdots (t_i - t_n)}$, $i = 0, \ldots, n$: independent of y_i!

↳ precompute in constructor, cost $= O(n^2)$

Observe: $y_i = 1 \;\forall i \Rightarrow p(t) = 1$

$$1 = \prod_{j=0}^{n} (t - t_j) \sum_{i=0}^{n} \frac{\lambda_i}{t - t_i} \quad \Rightarrow \quad \prod_{j=0}^{n} (t - t_j) = \frac{1}{\sum_{i=0}^{n} \frac{\lambda_i}{t - t_i}}$$

▷ Barycentric interpolation formula

$$p(t) = \frac{\sum_{i=0}^{n} \frac{\lambda_i}{t - t_i} y_i}{\sum_{i=0}^{n} \frac{\lambda_i}{t - t_i}} . \qquad (5.2.3.3)$$

③

C++-code 5.2.3.7: Interpolation class: multiple point evaluations

```
template <typename NODESCALAR>
template <typename RESVEC, typename DATAVEC>
RESVEC BarycPolyInterp<NODESCALAR>::eval(const DATAVEC &y,
                                         const nodeVec_t &x) const {
  const idx_t N = x.size(); // No.  of evaluation points
  RESVEC p(N);              // Ouput vector
  // Compute quotient of weighted sums of λ_i/(t-t_i),
  // effort O(n)
  for (int i = 0; i < N; ++i) {
    nodeVec_t z = (x[i] * nodeVec_t::Ones(n) - t);
    // Check if we want to evaluate close to a node
    const double tref{z.cwiseAbs().maxCoeff()}; // reference size
    idx_t k;
    if (z.cwiseAbs().minCoeff(&k) <
        tref * std::abs(std::numeric_limits<NODESCALAR>::epsilon())) {
      // evaluation at node t_k
      p[i] = y[k];
    } else {
      const nodeVec_t mu = lambda.cwiseQuotient(z);
      p[i] = (mu.cwiseProduct(y)).sum() / mu.sum();
    }
  } // end for
  return p;
}
```

→ $[x_k - t_i]_{i=0}^{n}$

→ avoid "/0"

→ comb. weight (5.2.3.3)

→ Loop over x_k - values

Total cost $= O(n^2 + N \cdot n)$

n^2 ↓ constructor

$N \cdot n$ ↓ eval()

5.2.3.2. Single evaluation

```
double eval(const Eigen::VectorXd &t,const Eigen::VectorXd &y,
            double x);
```

$n := (t.\text{size}() = x.\text{size}()) - 1 \Rightarrow 1$

Tool: "partial polynomial interpolants"

$p_{k,\ell} \in \mathcal{P}_{\ell-k}$: $p_{k,\ell}(t_j) = y_j$, $j = k, \ldots, \ell$

Idea: recursion in degree

$p_{k,k}(x) \equiv y_k$ ("constant polynomial"), $k = 0, \ldots, n$, ✓

$$p_{k,\ell}(x) = \frac{(x - t_k)p_{k+1,\ell}(x) - (x - t_\ell)p_{k,\ell-1}(x)}{t_\ell - t_k} = * \in \mathcal{P}_{\ell-k} \qquad (5.2.3.9)$$

$$= p_{k+1,\ell}(x) + \frac{x - t_\ell}{t_\ell - t_k}(p_{k+1,\ell}(x) - p_{k,\ell-1}(x)), \quad 0 \le k \le \ell \le n,$$

$*$ ↓ satisfies IC $= \begin{cases} y_j & \text{for } x = t_j,\ k < j < \ell \\ y_k & \text{for } x = t_k \\ y_\ell & \text{for } x = t_\ell \end{cases}$

Finally: $p = p_{0,n}$

④

Ex: $n = 3$

$n =$	0	1	2	3
t_0	$y_0 =: p_{0,0}(x)$	$\to p_{0,1}(x)$	$\to p_{0,2}(x)$	$\to p_{0,3}(x)$
t_1	$y_1 =: p_{1,1}(x)$	$\to p_{1,2}(x)$	$\to p_{1,3}(x)$	
t_2	$y_2 =: p_{2,2}(x)$	$\to p_{2,3}(x)$		
t_3	$y_3 =: p_{3,3}(x)$			

→ i loop

↓ k loop

(ANS)

Aitken-Neville scheme

C++-code 5.2.3.10: Aitken-Neville algorithm

```
// Aitken-Neville algorithm for evaluation of interpolating polynomial
// IN: t, y:  (vectors of) interpolation data points
// x:  (single) evaluation point
// OUT: value of interpolant in x
double ANipoleval(const VectorXd& t, VectorXd y, const double x) {
  for (int i = 0; i < y.size(); ++i) {
    for (int k = i - 1; k >= 0; --k) {
      // Recursion (5.2.3.9)
      y[k] = y[k + 1] + (y[k + 1] - y[k])*(x - t[i])/(t[i] - t[k]);
    }
  }
  return y[0];
}
```

cost = $O(n^2)$

Memory efficient: $O(n)$

pol. interpolant never computed!

A.N. useful for simultaneous evaluations & updates

C++-code 5.2.3.12: Single point evaluation with data updates

```
class PolyEval {
    private:
     // evaluation point and various internal data describing the
        polynomials
    public:
     // Constructor taking the evaluation point as argument
     PolyEval(double x);
     // Add another data point and update internal information
     void addPoint(t,y);
     // Value of current interpolating polynomial at x
     double eval(void) const;
};
```

→ data update

Ex: $n = 3$

$n =$	0	1	2	3
t_0	$y_0 =: p_{0,0}(x)$	$\to p_{0,1}(x)$	$\to p_{0,2}(x)$	$\to p_{0,3}(x)$
t_1	$y_1 =: p_{1,1}(x)$	$\to p_{1,2}(x)$	$\to p_{1,3}(x)$	
t_2	$y_2 =: p_{2,2}(x)$	$\to p_{2,3}(x)$		
t_3	$y_3 =: p_{3,3}(x)$			

cost(single update) = $O(k)$, k = current #data

Memory efficient, cost(eval) = $O(1)$

▷ Review questions 5.2.3.14

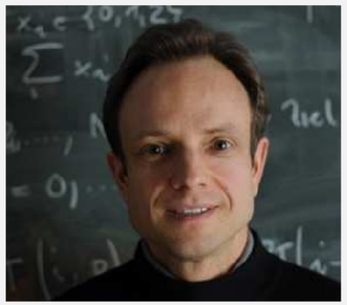
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 5.2.3.3: Extrapolation to Zero

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 27, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.
- Taylor expansion from calculus

Dependencies. Section 5.2.3, Ex. 1.5.4.7

Duration: 23 minutes

Note: Possible minor *mismatch of video and tablet notes*!

[Corrections and updates can be incorporated into tablet notes only]

V. Data Interpolation and Data Fitting in 1D

5.2. Global Polynomial Interpolation

5.2.3. Polynomial Interpolation: Algorithms

5.2.3.3. Extrapolation to Zero

Compute $\Psi(0)$ for "nice" function $\Psi : [-1,1] \to \mathbb{R}$
↳ available as `double Psi(double h)`
not reliable for $|h| \ll 1$

Example 5.2.3.16 (Numerical differention through extrapolation)
$(f \in C^2)$

$$\Psi(h) := \frac{f(x+h) - f(x-h)}{2h} \overset{!}{=} \text{cancellation for } h \approx 0$$

↳ $\Psi(0) = f'(x)$

② Example:

$f(x) = \arctan(x)$

h	Relative error
2^{-1}	0.20786640808609
2^{-6}	0.00773341103991
2^{-11}	0.00024299312415
2^{-16}	0.00000759482296
2^{-21}	0.00000023712637
2^{-26}	0.00000001020730
2^{-31}	0.00000005960464
2^{-36}	0.00000679016113

$f(x) = \sqrt{x}$

h	Relative error
2^{-1}	0.09340033543136
2^{-6}	0.00352613693103
2^{-11}	0.00011094838842
2^{-16}	0.00000346787667
2^{-21}	0.00000010812198
2^{-26}	0.00000001923506
2^{-31}	0.00000001202188
2^{-36}	0.00000198842224

$f(x) = \exp(x)$

h	Relative error
2^{-1}	0.29744254140026
2^{-6}	0.00785334954789
2^{-11}	0.00024418036620
2^{-16}	0.00000762943394
2^{-21}	0.00000023835113
2^{-26}	0.00000000429331
2^{-31}	0.00000012467100
2^{-36}	0.00000495453865

Taylor expansion, $f \in C^{L+1}$

Small remainder for $|h| \ll 1$

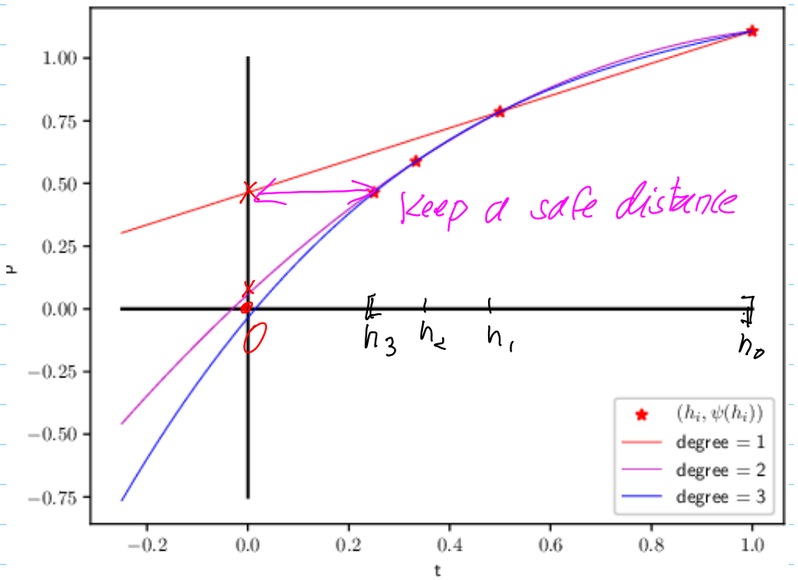
$$\psi(h) := \frac{f(x+h) - f(x-h)}{2h} = \underbrace{f'(x) + \sum_{\ell=1}^{L} c_\ell h^\ell}_{\text{a polynomial } p \text{ in } h:\ p(0) = \psi(0)} + \left[O(h^{L+1}) \right]$$

Idea: computing inaccessible limit by extrapolation to zero

① Pick $h_0, \ldots, h_n$ for which ψ can be evaluated "safely".
② evaluation of $\psi(h_i)$ for different h_i, $i = 0, \ldots, n$, $|t_i| > 0$. } $(h_i, \psi(h_i))$
③ $\psi(0) \approx p(0)$ with interpolating polynomial $p \in \mathcal{P}_n$, $p(h_i) = \psi(h_i)$.

"Extrapolation"

0 & interval covered by nodes



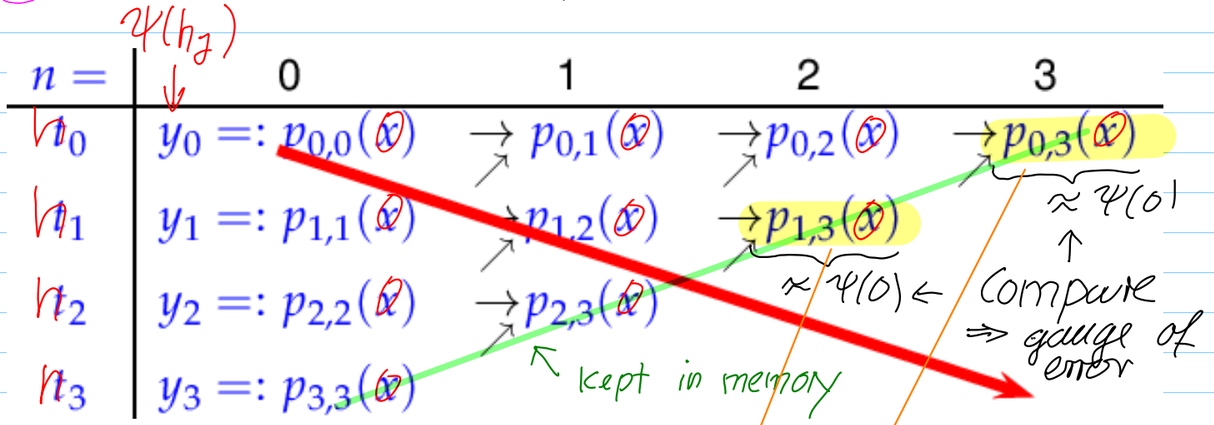
Algorithm: Compute $p(0)$ by Aitken-Neville

Recall: A.N. "update-friendly", for single evaluation

C++code 5.2.3.21: Polynomial evaluation

```
class PolyEval {
  private:
    // evaluation point and various internal data describing the
       polynomials
  public:
    // Idle Constructor
    PolyEval();
    // Add another data point and update internal information
    void addPoint(t,y);        → add (h_j, ψ(h_j))
    // Evaluation of current interpolating polynomial at x
    Eigen::VectorXd operator () (const Eigen::VectorXd &x) const;
};
```

③ A.N. recursive ($n = 3$)



C++-code 5.2.3.19: Numerical differentiation by adaptive extrapolation to zero

```
// Extrapolation based numerical differentation
// with a posteriori error control
// f:  handle of a function defined in a neighbourhood of x ∈ ℝ
// x:  point at which approximate derivative is desired
// h0:  initial distance from x
// rtol:  relative target tolerance, atol:  absolute tolerance
template <class Function>
double diffex(Function& f, const double x, const double h0,
              const double rtol, const double atol) {
  const unsigned nit = 10; // Maximum depth of extrapolation
  VectorXd h(nit); h[0] = h0; // Widths of difference quotients
  VectorXd y(nit); // Approximations returned by difference quotients
  y[0] = (f(x + h0) - f(x - h0))/(2*h0); // Widest difference quotients

  // using Aitken-Neville scheme with x = 0, see Code 5.2.3.10
  for (unsigned i = 1; i < nit; ++i) {
    // create data points for extrapolation
    h[i] = h[i-1]/2; // Next width half a big
    y[i] = ( f(x + h[i]) - f(x - h[i]) )/(2.0*h[i - 1]); // } Next data point (h_j, Ψ(h_j))
    // Aitken-Neville update
    for (int k = i - 1; k >= 0; --k)
      y[k] = y[k+1] - (y[k+1]-y[k])*h[i]/(h[i]-h[k]); // → next diagonal
    // termination of extrapolation when desired tolerance is reached
    const double errest = std::abs(y[1]-y[0]); // error indicator
    if ( errest < rtol*std::abs(y[0]) || errest < atol ) // } stop, when estimated (relative) error ≤ tolerance
      break;
  }
  return y[0]; // Return value extrapolated from largest number of
    difference quotients
}
```

`auto f = [](double x) {return std::atan(x);}` `diffex(f,1.1,0.5)`

Degree	Relative error
0	0.04262829970946
1	0.02044767428982
2	0.00051308519253
3	0.00004087236665
4	0.00000048930018
5	0.00000000746031
6	0.00000000001224

`auto g = [](double x) {return std::sqrt(x);}` `diffex(g,1.1,0.5)`

Degree	Relative error
0	0.02849215135713
1	0.01527790811946
2	0.00061205284652
3	0.00004936258481
4	0.00000067201034
5	0.00000001253250
6	0.00000000004816
7	0.00000000000021

`auto h = [](double x) {return std::exp(x);}` `diffex(h,1.1,0.5)`

Degree	Relative error
0	0.04219061098749
1	0.02129207652215
2	0.00011487434095
3	0.00000825582406
4	0.00000000589624
5	0.00000000009546
6	0.00000000000002

Enhanced accuracy + error control

▷ Review questions 5.2.3.20

ETH Lecture 401-0663-00L Numerical Methods for CSE

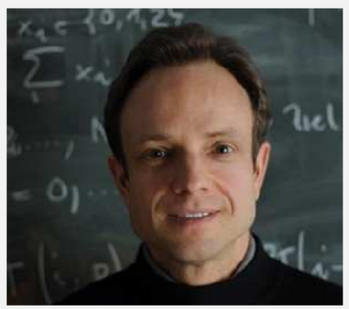
Course Video

Section 5.2.3.4: Newton Basis and Divided Differences

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 28, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 5.1, Section 5.2.2, Section 5.2.3.1, Section 5.2.3.2

Duration: 30 minutes

V. Data Interpolation and Data Fitting in 1D

5.2. Global Polynomial Interpolation

5.2.3. Polynomial Interpolation: Algorithms

5.2.3.4. Newton Basis and Divided Differences

C++code 5.2.3.21: Polynomial evaluation

```
1  class PolyEval {
2    private:
3      // evaluation point and various internal data describing the
           polynomials
4    public:
5      // Idle Constructor
6      PolyEval();
7      // Add another data point and update internal information
8      void addPoint(t,y);
9      // Evaluation of current interpolating polynomial at x
10     Eigen::VectorXd operator () (const Eigen::VectorXd &x) const;
11 };
```

$\Longrightarrow$ add data point $(t_j, y_j) \in \mathbb{R}^2$

evaluation in $N \gg 1$ points

(2)

Idea: Use of an "update-friendly" basis of $\mathcal{P}_n$

$\longrightarrow$ Newton basis

$N_0(t) \equiv 1, \ldots, N_k(t) = \prod_{i=0}^{k-1}(t-t_i)$, $k = 1, \ldots, n$

$\{N_0, \ldots, N_n\}$ is a basis of $\mathcal{P}_n$

↑ basis of $\mathcal{P}_n$

$\in \mathcal{P}_k$ w/ leading coefficient = 1

▷ $N_k(t_j) = 0$, $j = 0, \ldots, k-1$ (*)

▷ a_k depends only on (t_j, y_j): $0 \le j \le k$

No change of $a_0, \ldots, a_n$ when (t_{n+1}, y_{n+1}) added

§ 5.2.3.33: Efficient evaluation of a polynomial in Newton form

$$p(t) = a_0 N_0(t) + a_1 N_1(t) + \cdots + a_n N_n(t)$$

M.B.

$$= a_0 + a_1(t-t_0) + a_2(t-t_0)(t-t_1) + \cdots + a_n \prod_{j=0}^{n-1}(t-t_j), \quad t \in \mathbb{R},$$

$$= a_0 + (\cdots((a_n(t-t_n) + a_{n-1})(t-t_{n-1}) + a_{n-2})\cdots + a_1)(t-t_1)$$

$\longrightarrow$ Horner-like scheme

IC: $a_j \in \mathbb{R}$: $a_0 N_0(t_j) + a_1 N_1(t_j) + \cdots + a_n N_n(t_j) = y_j$, $j = 0, \ldots, n$. (5.2.3.24)

$\Leftrightarrow$ *triangular* linear system

LSE:

$$\begin{bmatrix} N_0(t_0) & N_1(t_0) & \cdots & N_n(t_0) \\ N_0(t_1) & N_1(t_1) & \cdots & N_n(t_1) \\ \vdots & \vdots & & \vdots \\ N_0(t_n) & N_1(t_n) & \cdots & N_n(t_n) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

$\Updownarrow$

triangular LSE

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & (t_1-t_0) & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 1 & (t_n-t_0) & \cdots & \prod_{i=0}^{n-1}(t_n-t_i) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}. \quad (5.2.3.25)$$

C++-code 5.2.3.34: Divided differences evaluation by modified Horner scheme

```
// Evaluation of a polynomial in Newton form, that is, represented through the
// vector of its basis expansion coefficients with respect to the Newton basis
// (5.2.3.23).
Eigen::VectorXd evalNewtonForm(const Eigen::VectorXd &t,   <- nodes
                               const Eigen::VectorXd &a,   <- basis exp. coeffs
                               const Eigen::VectorXd &x) {  <- eval pts.
  const unsigned int n = a.size() - 1;
  const Eigen::VectorXd ones = VectorXd::Ones(x.size());
  Eigen::VectorXd p{a[n] * ones};
  for (int j = n - 1; j >= 0; --j) {
    p = (x - t[j] * ones).cwiseProduct(p) + a[j] * ones;
  }
  return p;
}
```

▷ Cost = $O(n)$

④

§ 5.2.3.26 : Divided differences [→ compute a_j]

$$p(t) = \sum_{j=0}^{n} a_j N_j(t)$$

The Newton basis polynomial $N_j(t)$ has degree j and leading coefficient 1

► a_j is the leading coefficient of the interpolating polynomial $p_{0,j}$.

"Partial polynomial interpolants"

$p_{\ell,m} \in \mathcal{P}_{m-\ell}$: $p(t_j) = y_j$, $j = \ell, \dots, m$

$$p_{0,m}(t) = \sum_{j=0}^{m} a_j N_j(t) \quad \leftarrow$$

Notation : $y[t_\ell, \dots, t_m] \triangleq$ leading coeff. of $p_{\ell,m}$

$$a_j = y[t_0, \dots, t_j] \quad (*)$$

Remember : Aitken-Neville recursion

$p_{k,k}(x) \equiv y_k$ ("constant polynomial"), $k = 0, \dots, n$,

$$p_{k,\ell}(x) = \frac{(x - t_k)p_{k+1,\ell}(x) - (x - t_\ell)p_{k,\ell-1}(x)}{t_\ell - t_k} \tag{5.2.3.9}$$

$$= p_{k+1,\ell}(x) + \frac{x - t_\ell}{t_\ell - t_k}(p_{k+1,\ell}(x) - p_{k,\ell-1}(x)), \quad 0 \le k \le \ell \le n,$$

▷ Recursion for leading coefficients

$$y[t_i] = y_i$$

$$y[t_i, \dots, t_{i+k}] = \frac{y[t_{i+1}, \dots, t_{i+k}] - y[t_i, \dots, t_{i+k-1}]}{t_{i+k} - t_i} \quad \text{(recursion)}. \tag{5.2.3.28}$$

↳ divided differences

A.N.-like scheme :

$$\begin{array}{c|ccccc} t_0 & y[t_0] & & & & \\ & & > y[t_0,t_1] & & & \\ t_1 & y[t_1] & & > y[t_0,t_1,t_2] & & \\ & & > y[t_1,t_2] & & > y[t_0,t_1,t_2,t_3], & \\ t_2 & y[t_2] & & > y[t_1,t_2,t_3] & & \\ & & > y[t_2,t_3] & & & \\ t_3 & y[t_3] & & & & \end{array} \tag{5.2.3.30}$$

$= a_j$ by $(*)$

↳ in place : $y_j \to a_j$

C++ code 5.2.3.31: In-situ computation of divided differences

```
// IN: t = node set (mutually different)
// y = nodal values
// OUT: y = coefficients of polynomial in Newton basis
void divdiff(const VectorXd &t, VectorXd &y) {
  const int n = y.size() - 1;
  // Follow scheme (5.2.3.30), recursion (5.2.3.27)
  for (int l = 1; l < n; ++l)
    for (int j = n - l; j < n; ++j)
      y[j] = (y[j] - y[j - 1]) / (t[j] - t[n - 1 - l]);
}
```

▷ Cost = $O(n^2)$

Example 5.2.3.35 : Class PolyEval

→ Combined : • adding data

• evaluation in many points

depend only on $t_0, \dots, t_{n-1}$

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & (t_1 - t_0) & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 1 & (t_n - t_0) & \cdots & \prod_{i=0}^{n-1}(t_n - t_i) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}. \tag{5.2.3.25}$$

$$\begin{aligned} a_n &= \prod_{i=0}^{n-1}(t_n - t_i)^{-1} \left(y_n - \sum_{k=0}^{n-1} \prod_{i=0}^{k-1} (t_n - t_i)\, a_k \right) \\ &= \prod_{i=0}^{n-1}(t_n - t_i)^{-1} y_n - \sum_{k=0}^{n-1} \prod_{i=k}^{n-1} (t_n - t_i)^{-1}\, a_k \\ &= (\dots(((y_n - a_0)/(t_n - t_0) - a_1)/(t_n - t_1) - a_2)/\cdots - a_{n-1})/(t_n - t_{n-1}) . \end{aligned}$$

→ Horner-like formula

C++-code 5.2.3.37: Implementation of class PolyEval

```
1  PolyEval::PolyEval() {}
2
3  void PolyEval::addPoint(double td, double yd) {
4    t.push_back(td);
5    y.push_back(yd);
6    int n = t.size();
7    for (int j = 0; j < n - 1; j++)
8      y[n - 1] = ((y[n - 1] - y[j]) / (t[n - 1] - t[j]));
9  }
10
11 double PolyEval::operator()(double x) const {
12   double s = y.back();
13   for (int i = y.size() - 2; i >= 0; --i)
14     s = s * (x - t[i]) + y[i];
15   return s;
16 }
```

▷ cost = O(n)

→ a_n

▷ Review questions 5.2.3.39

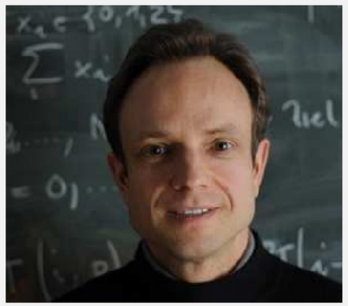
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 5.2.4: Polynomial Interpolation: Sensitivity

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 28, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.
- Norms for functions
- Norms of linear mappings

Dependencies. Section 2.2.2, Section 5.2.2

Duration: 23 minutes

V. Data Interpolation and Data Fitting in 1D

5.2. Global Polynomial Interpolation

5.2.4. Polynomial Interpolation: Sensitivity

Sensitivity (of a problem) $\leftrightarrow$ amplification of perturbations of data

$\hookrightarrow$ A mapping : {Data (space)} $\longrightarrow$ {Result (space)}

For LIP : { Data points : $(\mathbb{R}\times\mathbb{R})^{n+1}$; Data values : $\mathbb{R}^{n+1}$ } $\longrightarrow$ $\mathcal{P}_n$ (a function)

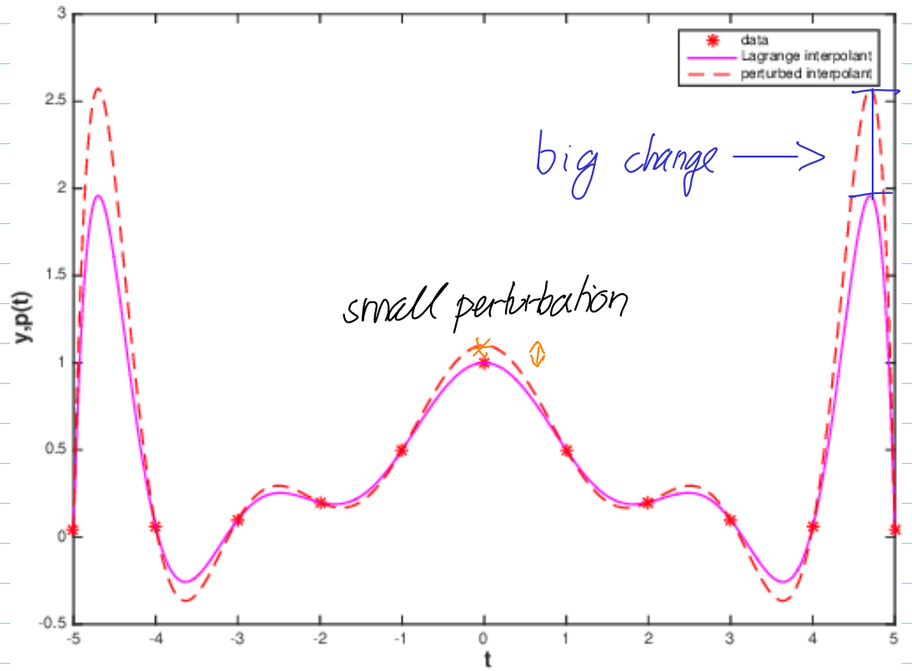
$\mathcal{I} \longleftrightarrow \{p : I \subset \mathbb{R} \to \mathbb{R}\}$

Our focus is on interpolation operator

$$I_{\mathcal{T}} : \begin{cases} \mathbb{R}^{n+1} & \to \mathcal{P}_n , \\ (y_0,\ldots,y_n)^T & \mapsto \text{interpolating polynomial } p . \end{cases} \tag{5.2.2.9}$$

(2)

Ex 5.2.4.1 (Oscillating polynomial interpolant)



- ☞ possible strong oscillations of interpolating polynomials of *high degree* on *uniformly spaced* nodes!
- ☞ Slight perturbations of data values can engender strong variations of a *high-degree* Lagrange interpolant "far away".

Needed for quantifying perturbations: norms

- On data space $\mathbb{R}^{n+1}$: $\|y\|_2$, $\|y\|_\infty$
- On result space $\mathcal{P}_n \hat{=}$ a function space on $I \subset \mathbb{R}$

supremum norm $\|f\|_{L^\infty(I)} := \sup\{|f(t)| : t \in I\}$, (5.2.4.5)

L^2-norm $\|f\|^2_{L^2(I)} := \int_I |f(t)|^2 \,\mathrm{d}t$, $\longleftrightarrow$ $\|y\|_2$ (5.2.4.6)

L^1-norm $\|f\|_{L^1(I)} := \int_I |f(t)| \,\mathrm{d}t$. $\longleftrightarrow$ $\|y\|_1$ (5.2.4.7)

§5.2.4.7: Sensitivity of linear (problem) maps

$$I_\tau(y + \delta y) = I_\tau(y) + I_\tau(\delta y)$$

↳ perturbation: $\delta y \longrightarrow I_\tau(\delta y)$

Sensitivity: $\sup_{\delta y \neq 0} \dfrac{\|I_\tau(\delta y)\|}{\|\delta y\|}$

$\hat{=}$ operator norm of I_τ

③

∞-sensitivity

Lagrange polynomials

$$\|I_{\mathcal{T}}(y)\|_{L^\infty(I)} = \sup_{t\in I} \left| \sum_{j=0}^{n} y_j L_j(t) \right|$$

Δ-ineq.

$$\leq \sum_{j=0}^{n} |y_j| \, \|L_j\|_{L^\infty(I)}$$

$$\leq \|y\|_\infty \sum_{j=0}^{n} \|L_j\|_{L^\infty(I)}$$

Lemma 5.2.4.10. Absolute conditioning of polynomial interpolation

Given a mesh $\mathcal{T} \subset \mathbb{R}$ with generalized Lagrange polynomials L_i, $i = 0,\ldots,n$, and fixed $I \subset \mathbb{R}$, the norm of the interpolation operator satisfies

$$\|I_{\mathcal{T}}\|_{\infty\to\infty} := \sup_{\mathbf{y}\in\mathbb{R}^{n+1}\setminus\{0\}} \frac{\|I_{\mathcal{T}}(\mathbf{y})\|_{L^\infty(I)}}{\|\mathbf{y}\|_\infty} = \left\| \sum_{i=0}^{n} |L_i| \right\|_{L^\infty(I)}, \tag{5.2.4.11}$$

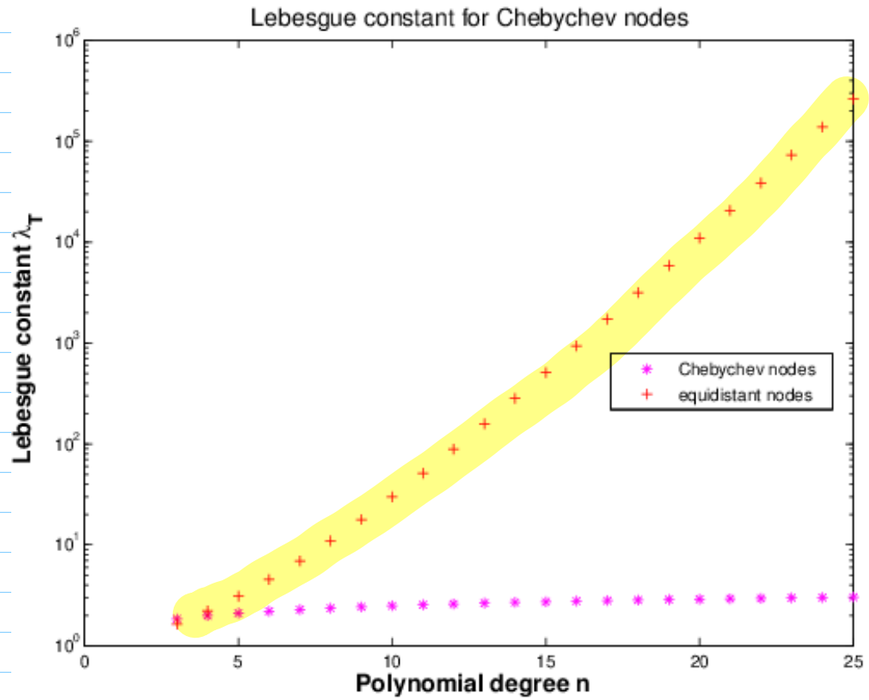
$$\|I_{\mathcal{T}}\|_{2\to 2} := \sup_{\mathbf{y}\in\mathbb{R}^{n+1}\setminus\{0\}} \frac{\|I_{\mathcal{T}}(\mathbf{y})\|_{L^2(I)}}{\|\mathbf{y}\|_2} \leq \left(\sum_{i=0}^{n} \|L_i\|^2_{L^2(I)} \right)^{\frac{1}{2}}. \tag{5.2.4.12}$$

depends on $\mathcal{T}$

Lebesgue constant of $\mathcal{T}$: $\lambda_{\mathcal{T}} := \left\| \sum_{i=0}^{n} |L_i| \right\|_{L^\infty(I)} = \|I_{\mathcal{T}}\|_{\infty\to\infty}$

Rem. 5.2.4.12: Lebesgue constant of equidistant nodes

node set $\mathcal{T}_n = \{-1 + 2\frac{k}{n}\}_{k=0}^{n} \subset I := [-1,1]$



Δ lin-log plot

$\lambda_{\mathcal{T}_n} \geq Ce^{n/2}$

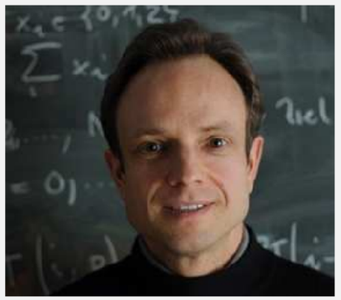
exp. growth

Due to potentially "high sensitivity" interpolation with global polynomials of high degree is not suitable for data interpolation.

▷ Review questions 5.2.4.16

ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 5.3: Shape-Preserving Interpolation



Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 30, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich

Prerequisites.
- Notion of monotonicity and convexity of functions

Dependencies. Section 5.1, Section 5.2.1

Duration: 47 minutes

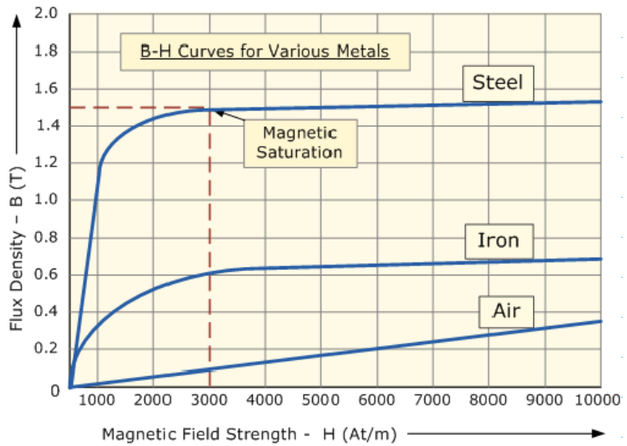
V. Data Interpolation and Data Fitting in 1D

5.3. Shape-Preserving Interpolation

Ex 5.3.0.1. (Magnetization curves)

- $H \mapsto B(H)$ smooth (at least C^1),
- $H \mapsto B(H)$ monotonic (increasing),
- $H \mapsto B(H)$ concave [curvature]

↑ shape properties



(2)

5.3.1. Shape properties of Functions & Data

§ 5.3.1.1. The "shape" of data

[Sorted nodes: $t_0 < t_1 < \dots < t_n$]

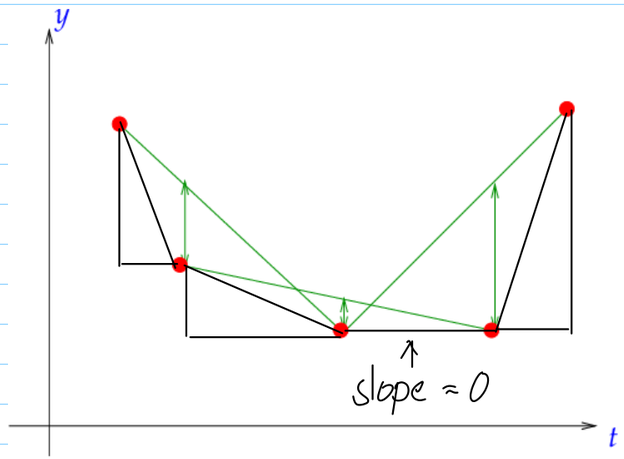
Definition 5.3.1.2. monotonic data

The data (t_i, y_i) are called **monotonic** when $y_i \geq y_{i-1}$ or $y_i \leq y_{i-1}$, $i = 1, \dots, n$.

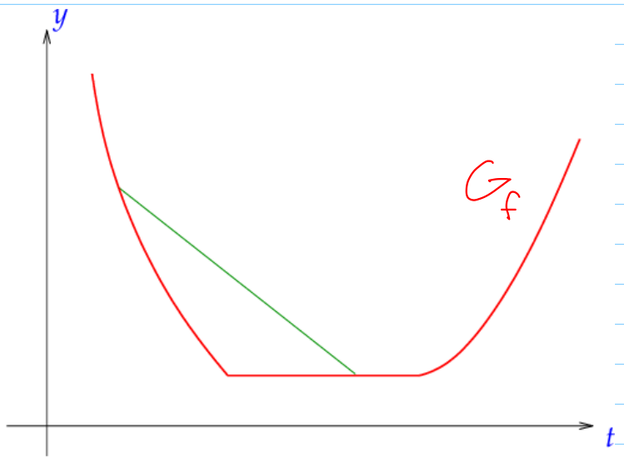
"curvature of data"

Definition 5.3.1.4. Convex/concave function →

$$f : I \subset \mathbb{R} \mapsto \mathbb{R} \quad \begin{matrix}\textbf{convex}\\ \textbf{concave}\end{matrix} \quad :\Leftrightarrow \quad \begin{matrix} f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y) \\ f(\lambda x + (1-\lambda)y) \geq \lambda f(x) + (1-\lambda) f(y) \end{matrix} \quad \begin{matrix}\forall\, 0 \leq \lambda \leq 1,\\ \forall\, x, y \in I\,.\end{matrix}$$



Convex data



Convex function

$\Leftarrow$ f' increasing

Definition 5.3.1.3. Convex/concave data

The data $\{(t_i, y_i)\}_{i=0}^n$ are called **convex** (**concave**) if

$$\Delta_j \overset{(\geq)}{\leq} \Delta_{j+1}\,, \quad j = 1, \dots, n-1 \quad , \quad \Delta_j := \frac{y_j - y_{j-1}}{t_j - t_{j-1}}\,, \quad j = 1, \dots, n\,.$$

↑ monotonicity of slopes ↑ slopes

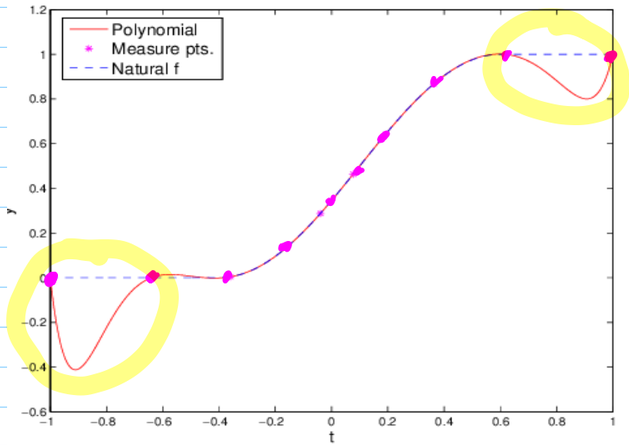
Local shape preservation:

shape property of data ⇒ same shape property of interpolation

↓

hold in every interval $[t_j, t_k]$, $0 \leq j < k \leq n$

Exp.: 5.3.1.6 (Global polynomial interpolation)

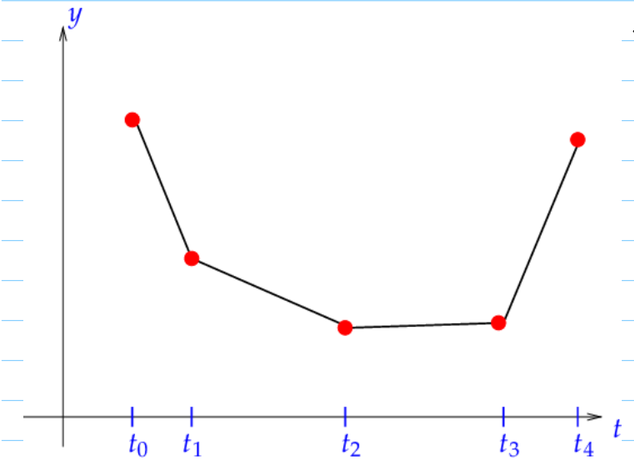


△ Oscillations

← Interpolating polynomial, degree = 10

- no locality,
- no positivity,
- no monotonicity,
- no local conservation of the curvature,

5.3.2. Piecewise Linear Interpolation



◁ polygon
- continuous interpolant
- local
- linear interpolation scheme
- perfectly locally shape-preserving

Theorem 5.3.2.3. Local shape preservation by piecewise linear interpolation

Let $s \in C([t_0,t_n])$ be the piecewise linear interpolant of $(t_i,y_i) \in \mathbb{R}^2$, $i = 0,\ldots,n$, for every subinterval $I = [t_j,t_k] \subset [t_0,t_n]$:

if $(t_i,y_i)|_I$ are positive/negative $\Rightarrow$ *$s|_I$ is positive/negative,*
if $(t_i,y_i)|_I$ are monotonic (increasing/decreasing) $\Rightarrow$ *$s|_I$ is monotonic (increasing/decreasing),*
if $(t_i,y_i)|_I$ are convex/concave $\Rightarrow$ *$s|_I$ is convex/concave.*

↳ perfectly local

A single flaw:
p.w. linear interpolant merely C^0, but not C^1
(lack of smoothness)

5.3.3. Cubic Hermite Interpolation

Goal/policy:
- operates locally
- use p.w. polynomials
- achieve C^1 by raising degree

} like p.w. linear interp.

↳ degree = 3

5.3.3.1 Definition and Algorithms

Definition 5.3.3.1. Cubic Hermite polynomial interpolant

Given data points $(t_j,y_j) \in \mathbb{R}\times\mathbb{R}$, $j = 0,\ldots,n$, with pairwise distinct ordered nodes t_j, and slopes $c_j \in \mathbb{R}$, the piecewise **cubic Hermite interpolant** $s : [t_0,t_n] \to \mathbb{R}$ is defined by the requirements

$$s_{|[t_{i-1},t_i]} \in \mathcal{P}_3\,,\ i = 1,\ldots,n \quad , \quad s(t_i) = y_i \quad , \quad s'(t_i) = c_i\,,\ i = 0,\ldots,n\,.$$

piecewise polynomial — interpolation conditions

$$s \in C^1([t_0,t_n])$$

[because s' continuous by construction]

Compute with local representations

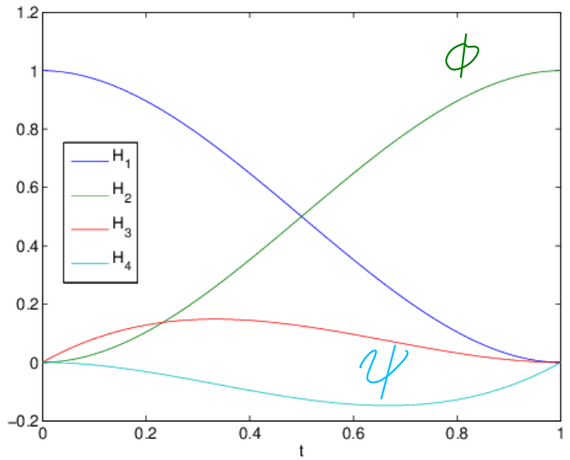
$$s(t) = y_{i-1}H_1(t) + y_iH_2(t) + c_{i-1}H_3(t) + c_iH_4(t) \quad , \quad t \in [t_{i-1}, t_i] \, , \tag{5.3.3.4}$$

$H_i \in \mathcal{P}_3$:

a generalized cardinal basis

	$H(t_{i-1})$	$H(t_i)$	$H'(t_{i-1})$	$H'(t_i)$
H_1	1	0	0	0
H_2	0	1	0	0
H_3	0	0	1	0
H_4	0	0	0	1

↑ by chain rule



Local basis polynomial on $[0,1]$

$$H_1(t) := \phi(\tfrac{t_i - t}{h_i}) \, , \tag{5.3.3.5a}$$
$$H_2(t) := \phi(\tfrac{t - t_{i-1}}{h_i}) \, , \tag{5.3.3.5b}$$
$$H_3(t) := -h_i\psi(\tfrac{t_i - t}{h_i}) \, , \tag{5.3.3.5c}$$
$$H_4(t) := h_i\psi(\tfrac{t - t_{i-1}}{h_i}) \, , \tag{5.3.3.5d}$$
$$h_i := t_i - t_{i-1} \, , \tag{5.3.3.5e}$$
$$\phi(\tau) := 3\tau^2 - 2\tau^3 \, , \tag{5.3.3.5f}$$
$$\psi(\tau) := \tau^3 - \tau^2 \, . \tag{5.3.3.5g}$$

NumCSE@ETHZ

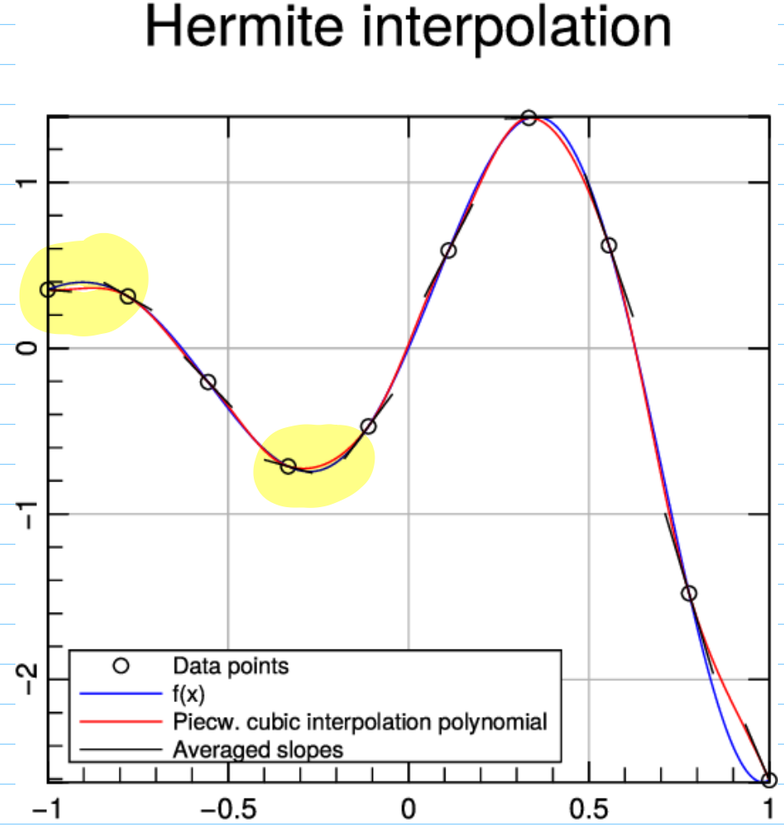
Ex 5.3.3.9 (Average-based p.w. cubic Hermite interpolation)

Needed: Reconstruction of slopes

Natural: use averages of data slopes

$$c_i = \begin{cases} \Delta_1 & \text{, for } i = 0 \, , \\ \Delta_n & \text{, for } i = n \, , \\ \frac{t_{i+1}-t_i}{t_{i+1}-t_{i-1}}\Delta_i + \frac{t_i - t_{i-1}}{t_{i+1}-t_{i-1}}\Delta_{i+1} & \text{, if } 1 \le i < n \, . \end{cases} \quad , \quad \Delta_j := \frac{y_j - y_{j-1}}{t_j - t_{j-1}} \, , j = 1, \ldots, n \, . \tag{5.3.3.8}$$

▷ linear interpolation scheme



Not bad, but not perfect!

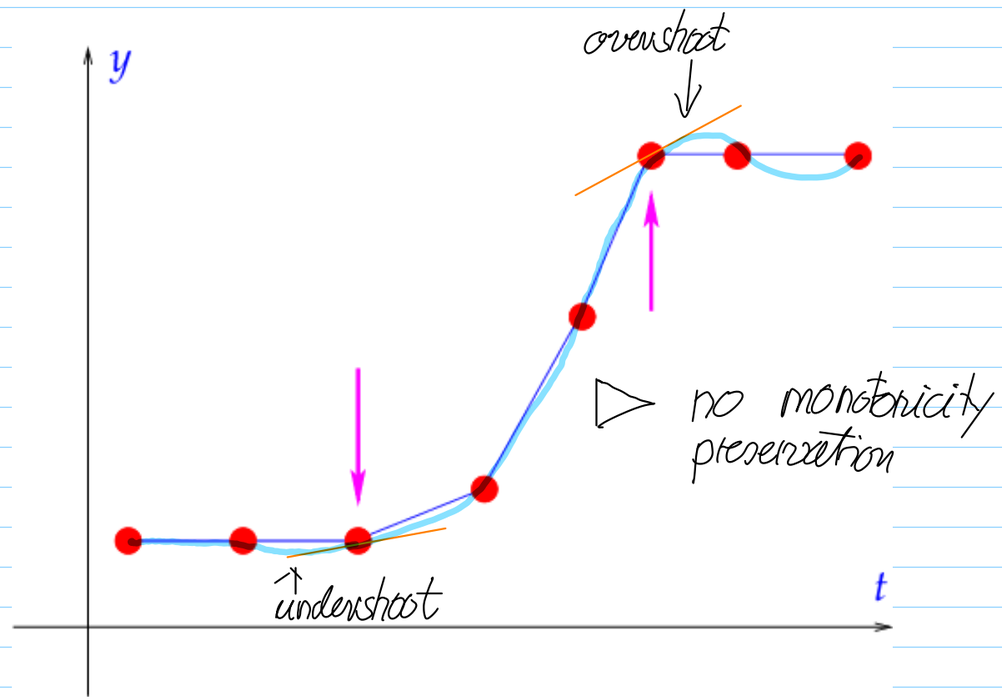
NumCSE@ETHZ

5.3.3.2. Local Monotonicity-Preserving Hermite Interpolation

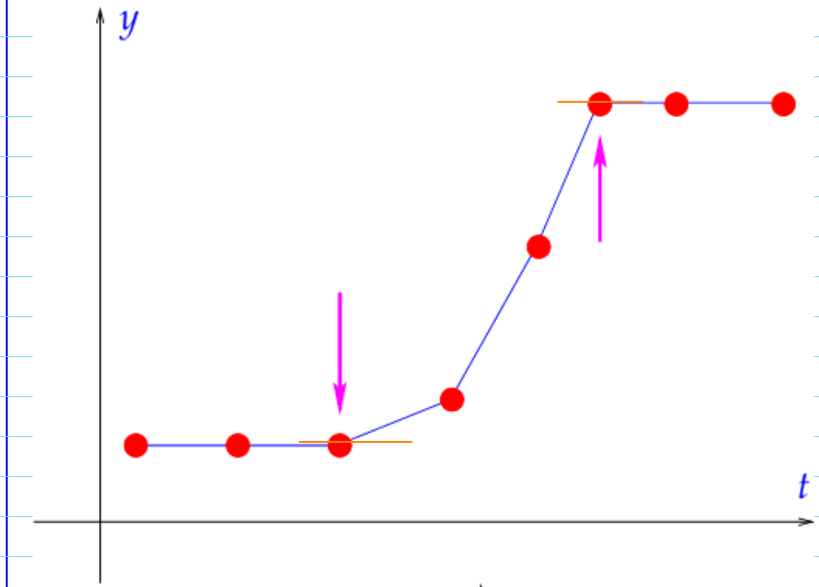
▷ Reconstruct slopes c_i that ensure local monotonicity preservation

The flaw of averaging:

$$c_i = \begin{cases} \Delta_1 & \text{, for } i = 0 \text{,} \\ \Delta_n & \text{, for } i = n \text{,} \\ \frac{t_{i+1}-t_i}{t_{i+1}-t_{i-1}}\Delta_i + \frac{t_i - t_{i-1}}{t_{i+1}-t_{i-1}}\Delta_{i+1} & \text{, if } 1 \le i < n \text{.} \end{cases} \quad , \quad \Delta_j := \frac{y_j - y_{j-1}}{t_j - t_{j-1}} \text{, } j = 1, \ldots, n \text{.} \tag{5.3.3.8}$$



▷ no monotonicity preservation

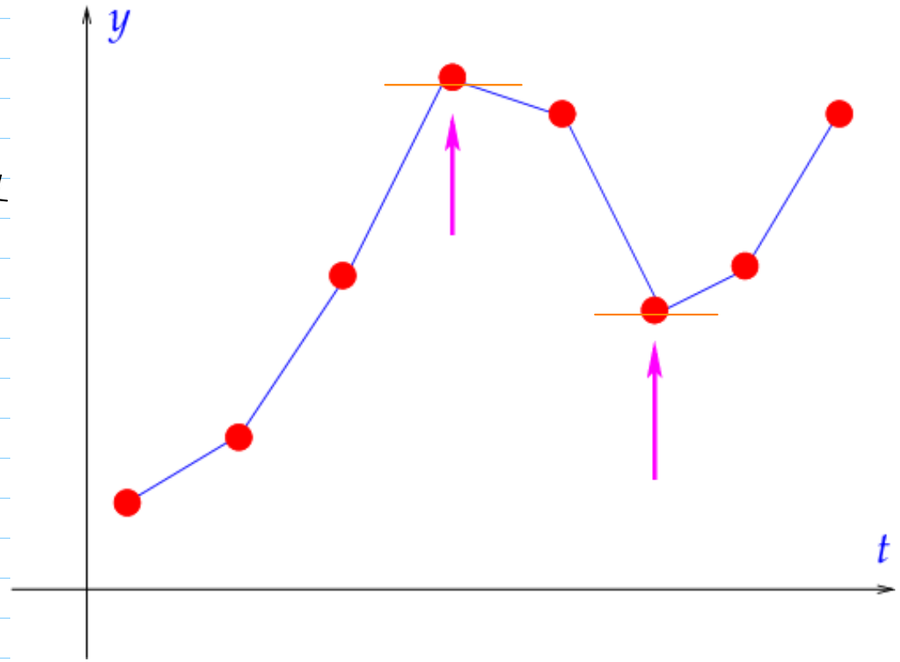


Necessary for monotonicity preservation:

slopes @ ↑-points = 0

For monotonicity preservation:

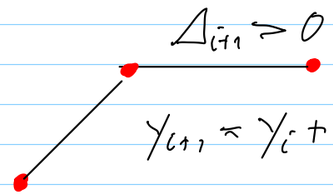
C^1-interpolant must be flat at all interior extremal points!



⑥ Necessary: Slope limiting

$$c_i = \begin{cases} 0 & \text{, if } \operatorname{sgn}(\Delta_i) \neq \operatorname{sgn}(\Delta_{i+1}) \text{,} \\ \text{some "average" of } \Delta_i, \Delta_{i+1} & \text{otherwise} \end{cases} \quad , \quad i = 1, \ldots, n-1 \text{.} \qquad (5.3.3.11)$$

✎ notation: sign function $\operatorname{sgn}(\xi) = \begin{cases} 1 & \text{, if } \xi > 0 \text{,} \\ 0 & \text{, if } \xi = 0 \text{,} \\ -1 & \text{, if } \xi < 0 \text{.} \end{cases}$



$\Rightarrow$ average$(\Delta_i, \Delta_{i+1}) \approx 0$, if $\Delta_i \approx 0$ or $\Delta_{i+1} \approx 0$

Candidates:

(1) Geometric mean: $c_i = \sqrt{\Delta_i \Delta_{i+1}}$

→ Observation: not perfectly monotonicity pres.

(2) [Weighted] harmonic mean

$$c_i = \frac{1}{\frac{w_a}{\Delta_i} + \frac{w_b}{\Delta_{i+1}}} \text{,} \longrightarrow 0 \qquad (5.3.3.12)$$

as $\Delta_i \to 0$ or $\Delta_{i+1} \to 0$

with weights $w_a > 0$, $w_b > 0$, $(w_a + w_b = 1)$.

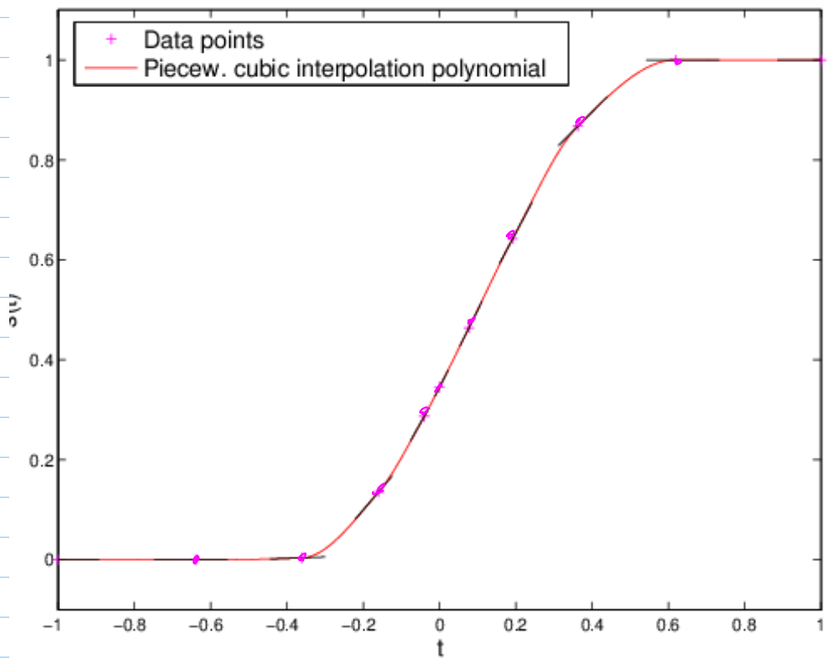
"pchip"

$$\xrightarrow{\operatorname{sgn}(\Delta_1)=\operatorname{sgn}(\Delta_2)} \quad c_i = \begin{cases} \Delta_1 & \text{, if } i = 0 \text{,} \\ \dfrac{3(h_{i+1}+h_i)}{\frac{2h_{i+1}+h_i}{\Delta_i} + \frac{2h_i+h_{i+1}}{\Delta_{i+1}}} & \text{, for } i \in \{1, \ldots, n-1\} \text{,} \\ \Delta_n & \text{, if } i = n \text{,} \end{cases} \quad h_i := t_i - t_{i-1} \text{.} \qquad (5.3.3.13)$$

Theorem 5.3.3.17. Monotonicity preservation of limited cubic Hermite interpolation

The cubic Hermite interpolation polynomial with slopes as in Eq. (5.3.3.13) provides a local monotonicity-preserving C^1-interpolant.

Ex. 5.3.3.14



→ Not a linear interpolation operator

▷ Review questions 5.3.3.19

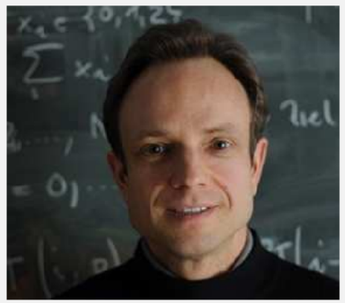
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 5.4.1: Spline Function Spaces

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 28, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 5.2.1

Duration: 15 minutes

V. Data Interpolation and Data Fitting in 1D

5.4 Splines

5.4.1. Spline Function Spaces

Definition 5.4.1.1. Spline space →

Given an interval $I := [a,b] \subset \mathbb{R}$ and a **knot sequence** $\mathcal{M} := \{a = t_0 < t_1 < \ldots < t_{n-1} < t_n = b\}$, $n \in \mathbb{N}$, the vector space $\mathcal{S}_{d,\mathcal{M}}$ of the **spline functions** of degree d (or order $d+1$) is defined by

$$\mathcal{S}_{d,\mathcal{M}} := \{s \in C^{d-1}(I) : s_j := s_{|[t_{j-1},t_j]} \in \mathcal{P}_d \; \forall j = 1,\ldots,n\} .$$

$C^{d-1}(I)$: $d-1$-times continuously differentiable

$\mathcal{P}_d$: *locally* polynomial of degree d

▽ [for interpolation]: knots ≠ node

Examples:

$d = 0$: p.w. constant ("$C^{-1} \hat{=}$ discontinuous")

$d = 1$: C^0 & p.w. linear (Graph $\hat{=}$ polygon)

$d = 2$: C^1 & p.w. quadratic

Issue : $S_{d,\mathcal{M}} \neq \mathcal{P}_d$?

dim $S_{d,\mathcal{M}}$ by a counting argument

$$\dim S_{d,\mathcal{M}} = \underbrace{n}_{\#\text{intervals}} \cdot \underbrace{\#\dim \mathcal{P}_d}_{=d+1} - \underbrace{\#\{\text{continuity constraints}\}}_{=(n-1)d}$$

no. of interior knots

Theorem 5.4.1.2. Dimension of spline space

The space $\mathcal{S}_{d,\mathcal{M}}$ from Def. 5.4.1.1 has dimension

$$\dim \mathcal{S}_{d,\mathcal{M}} = n + d\,.$$

$\dim S_{d,\mathcal{M}} > \dim \mathcal{P}_d$ if $n \geq 2$

NumCSE@ETHZ

Rem 5.4.1.3 : (Differentiating and Integrating Splines)

$s \in S_{d,\mathcal{M}} \Rightarrow s' \in S_{d-1,\mathcal{M}}$

$\Rightarrow \{ t \mapsto \int_a^t s(\tau)\,d\tau \} \in S_{d+1,\mathcal{M}}$

▷ Review questions 5.4.1.3

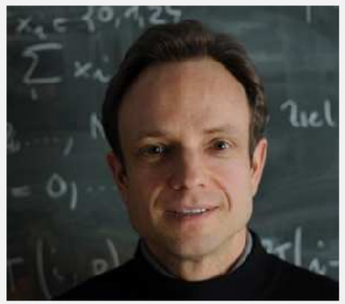
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 5.4.2: Cubic Spline Interpolation

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 7, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

-

Dependencies.

Duration: 26 minutes

V. Data Interpolation and Data Fitting in 1D

5.4 Splines

5.4.2. Cubic-Spline Interpolation (CSI)

One-dimensional interpolation

Given: data points (t_i, y_i), $i = 0, \ldots, n$, $n \in \mathbb{N}$, $t_i \in I \subset \mathbb{R}$, $y_i \in \mathbb{R}$

Objective: *Reconstruction of a function* $f : I \mapsto \mathbb{R}$

- satisfying the $n+1$ **interpolation conditions** (IC)

$$f(t_i) = y_i, \quad i = 0, \ldots, n \,. \tag{5.1.0.2}$$

- and belonging to a set V of eligible functions.

The function f we find is called the **interpolant** of the given data set $\{(t_i, y_i)\}_{i=0}^{n}$.

Now: $V = S_{3,\mathcal{M}} \subset C^2([t_0, t_n])$

with $\mathcal{M} = \{ t_0 < t_1 < \cdots < t_n \}$

node = knots

(2)

▷ Generalizes p.w. linear interpolation : $V = S_{1,\mathcal{M}}$

Definition 5.4.2.3. Cubic-spline interpolant

Given a node set/knot set $\mathcal{M} := \{t_0 < t_1 < \cdots < t_n\}$, $n \in \mathbb{N}$, and data values $y_0, \ldots, y_n \in \mathbb{R}$, an associated **cubic spline interpolant** is a function $s \in \mathcal{S}_{3,\mathcal{M}}$ that complies with the interpolation conditions

$$\text{IC}: \quad s(t_j) = y_j \ , \ j = 0, \ldots, n \, . \tag{5.4.2.4}$$

▽ $\dim \mathcal{S}_{3,\mathcal{M}} = 3 + n \longleftrightarrow n+1$ I.C.

$\longrightarrow$ 2 excess d.o.f.s

§ 5.4.2.5 Computing cubic-spline interpolants

Use local representation ($\leftrightarrow$ CHI)

(5.3.3.4)

$$s_{|[t_{j-1},t_j]}(t) = \begin{array}{ll} s(t_{j-1}) & \cdot \ (1 - 3\tau^2 + 2\tau^3) \ + \\ s(t_j) & \cdot \ (3\tau^2 - 2\tau^3) \ + \\ h_j s'(t_{j-1}) & \cdot \ (\tau - 2\tau^2 + \tau^3) \ + \\ h_j s'(t_j) & \cdot \ (-\tau^2 + \tau^3) \, , \end{array} \tag{5.4.2.4}$$

with $h_j := t_j - t_{j-1}$, $\tau := (t - t_{j-1})/h_j$.

$\widehat{=}$ known values $y_i = s(t_i)$

$\widehat{=}$ slopes $s'(t_i)$: unknown(s) c_i

▷ ensures I.C., s p.w. $\mathcal{P}_3$, $s \in C^1$

To be imposed : C^2-continuity only

$$s''_{|[t_{i-1},t_i]}(t_i) = s''_{|[t_i,t_{i+1}]}(t_i) \ , \ i = 1, \ldots, n-1$$

$\hookrightarrow$ $n-1$ linear equation

$$s''_{|[t_{j-1},t_j]}(t) = s(t_{j-1}) h_j^{-2} 6(-1+2\tau) + s(t_j) h_j^{-2} 6(1-2\tau) \tag{5.4.2.8}$$
$$+ h_j^{-1} s'(t_{j-1})(-4+6\tau) + h_j^{-1} s'(t_j)(-2+6\tau) \, , \ \tau := (t - t_{j-1})/h_j \, ,$$

Eq. (5.4.2.8) $\Rightarrow$

$$s''_{|[t_{j-1},t_j]}(t_{j-1}) = -6 \cdot s(t_{j-1}) h_j^{-2} + 6 \cdot s(t_j) h_j^{-2} - 4 \cdot h_j^{-1} s'(t_{j-1}) - 2 \cdot h_j^{-1} s'(t_j) \, ,$$

$\Leftarrow$

$$s''_{|[t_{j-1},t_j]}(t_j) = 6 \cdot s(t_{j-1}) h_j^{-2} + -6 \cdot s(t_j) h_j^{-2} + 2 \cdot h_j^{-1} s'(t_{j-1}) + 4 \cdot h_j^{-1} s'(t_j) \, .$$

$c_i := s'(t_i)$, $i = 0, \ldots, n$

$$\frac{1}{h_j} c_{j-1} + \left(\frac{2}{h_j} + \frac{2}{h_{j+1}} \right) c_j + \frac{1}{h_{j+1}} c_{j+1} = 3 \left(\frac{y_j - y_{j-1}}{h_j^2} + \frac{y_{j+1} - y_j}{h_{j+1}^2} \right) , \tag{5.4.2.9}$$

for $j = 1, \ldots, n-1$.

▷ $(n-1) \times (n+1)$ underdetermined LSE

2 "missing" equation

NumCSE@ETHZ

③

$$\begin{bmatrix} b_0 & a_1 & b_1 & 0 & \cdots & & \cdots & 0 \\ 0 & b_1 & a_2 & b_2 & & & & \\ & 0 & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & & & \\ & & & \ddots & a_{n-2} & b_{n-2} & 0 & \\ 0 & \cdots & & \cdots & 0 & b_{n-2} & a_{n-1} & b_{n-1} \end{bmatrix} \begin{bmatrix} c_0 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} 3\left(\frac{y_1-y_0}{h_1^2}+\frac{y_2-y_1}{h_2^2}\right) \\ \vdots \\ 3\left(\frac{y_{n-1}-y_{n-2}}{h_{n-1}^2}+\frac{y_n-y_{n-1}}{h_n^2}\right) \end{bmatrix} . \quad (5.4.2.10)$$

with

$$b_i := \frac{1}{h_{i+1}}, \quad a_i := \frac{2}{h_i}+\frac{2}{h_{i+1}}, \quad i = 0,1,\ldots,n-1, \quad [\ b_i, a_i > 0 \ , \ a_i = 2(b_i+b_{i-1})\ ].$$

Additional constraint to ensure uniqueness of CSI

① Complete CSI

Fix $c_0 := s'(t_0)$, $c_n := s'(t_n)$ → remove 2 unknowns → move to r.h.s.

$$\begin{bmatrix} b_0 & a_1 & b_1 & 0 & \cdots & & \cdots & 0 \\ 0 & b_1 & a_2 & b_2 & & & & \\ & 0 & \ddots & \ddots & \ddots & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & \ddots & a_{n-2} & b_{n-2} & 0 & \\ 0 & \cdots & & \cdots & 0 & b_{n-2} & a_{n-1} & b_{n-1} \end{bmatrix} \begin{bmatrix} c_0 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} 3\left(\frac{y_1-y_0}{h_1^2}+\frac{y_2-y_1}{h_2^2}\right) - c_0 b_0 \\ \vdots \\ 3\left(\frac{y_{n-1}-y_{n-2}}{h_{n-1}^2}+\frac{y_n-y_{n-1}}{h_n^2}\right) - c_n b_{n-1} \end{bmatrix} . \quad (5.4.2.10)$$

→ $(n-1)\times(n-1)$ LSE

with

$$b_i := \frac{1}{h_{i+1}}, \quad a_i := \frac{2}{h_i}+\frac{2}{h_{i+1}}, \quad i = 0,1,\ldots,n-1, \quad [\ b_i, a_i > 0 \ , \ a_i = 2(b_i+b_{i-1})\ ].$$

② Natural CSI : $s''(t_0) = s''(t_n) = 0$

→ two extra equations

$$\frac{2}{h_1}c_0 + \frac{1}{h_1}c_1 = 3\frac{y_1-y_0}{h_1^2}, \quad \frac{1}{h_n}c_{n-1} + \frac{2}{h_n}c_n = 3\frac{y_n-y_{n-1}}{h_n^2}.$$

▷ $(n+1)\times(n+1)$ LSE

③ Periodic CSI

$c_0 = c_n$ → −1 unknown

$s''(t_0) = s''(t_n)$ → +1 equation

▷ $n\times n$ LSE

▷ Review questions 5.4.2.16

ETH Lecture 401-0663-00L Numerical Methods for CSE

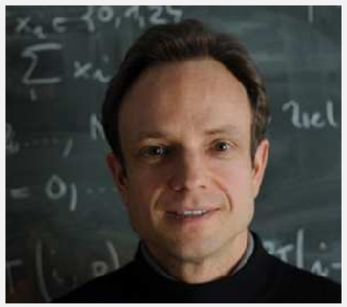
Course Video

Section 5.4.3: Structural Properties of Cubic Spline Interpolants

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 7, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 5.4.2

Duration: 24 minutes

V. Data Interpolation and Data Fitting in 1D

5.4 Splines

5.4.3. Structural Properties of Cubic-Spline Interpolants

Definition 5.4.2.3. Cubic-spline interpolant

Given a node set/knot set $\mathcal{M} := \{t_0 < t_1 < \cdots < t_n\}$, $n \in \mathbb{N}$, and data values $y_0, \ldots, y_n \in \mathbb{R}$, an associated **cubic spline interpolant** is a function $s \in \mathcal{S}_{3,\mathcal{M}}$ that complies with the interpolation conditions

$$s(t_j) = y_j \quad , \quad j = 0, \ldots, n \, . \tag{5.4.2.4}$$

▷ Non-unique

Unique natural CSI : $s''(t_0) = s''(t_n) = 0$

(2)

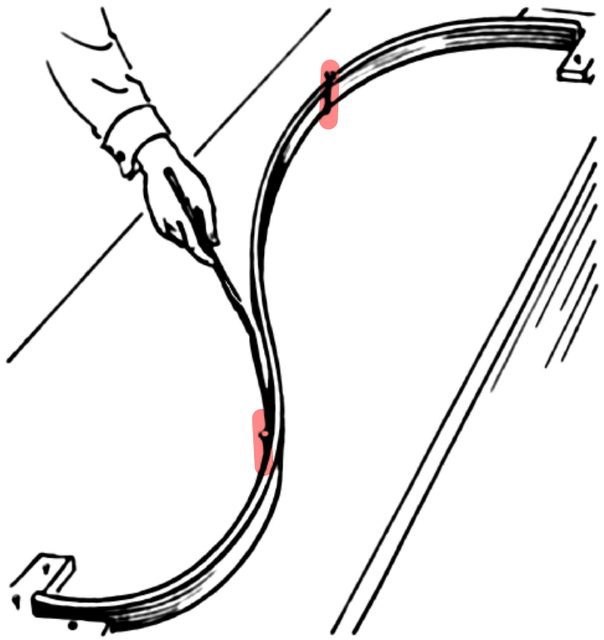
§5.4.3.1 (Minimal bending energy of natural CSI)

$$E_{bd}(f) = \frac{1}{2}\int_{t_0}^{t_n} |f''(t)|^2 \, dt$$

Theorem 5.4.3.3. Optimality of natural cubic spline interpolant

Given a knot/nodes set $\mathcal{M} := \{a = t_0 < t_1 < \cdots < t_n = b\}$ in the interval $[a,b]$, let $s \in \mathcal{S}_{3,\mathcal{M}}$ be the natural cubic spline interpolant of the data points $(t_i, y_i) \in \mathbb{R}^2$, $i = 0,\ldots,n$.
Then s minimizes the elastic bending energy among all interpolating functions in $C^2([a,b])$, that is

$$E_{\mathrm{bd}}(s) \le E_{\mathrm{bd}}(f) \quad \forall f \in C^2([a,b]),\ f(t_i) = y_i,\ i = 0,\ldots,n\,.$$



△ Natural CSI = equilibrium shape of "interpolating" elastic rod

"Calculus of variations" proof

$s \hat{=}$ natural CSI

perturbation: $k \in C^2([a,b])$, $k(t_i) = 0$

▷ $s+k$ yield any C^2-interpolant

$$E_{\mathrm{bd}}(s+k) = \frac{1}{2}\int_a^b |s'' + k''|^2 \, \mathrm{d}t \tag{5.4.3.4}$$

$$= E_{\mathrm{bd}}(s) + \underbrace{\int_a^b s''(t)k''(t)\,\mathrm{d}t}_{:=I=0} + \underbrace{\frac{1}{2}\int_a^b |k''|^2\,\mathrm{d}t}_{\ge 0}\,. \ge E_{bd}(s)$$

$$I = \sum_{j=1}^n \int_{t_{j-1}}^{t_j} s''(t)k''(t)\,\mathrm{d}t$$

[l.b.p.]

$$= \sum_{j=1}^n \left\{ -\int_{t_{j-1}}^{t_j} s'''(t)k'(t)\,\mathrm{d}t + s''(t_j)k'(t_j) - s''(t_{j-1})k'(t_{j-1}) \right\}$$

[i.b.p.]

$$= \sum_{j=1}^n \left\{ \int_{t_{j-1}}^{t_j} \underbrace{s^{(4)}(t)}_{=0 \text{ since } s|_{[t_{j-1},t_j]} \in \mathcal{P}_3} k(t)\,\mathrm{d}t + s''(t_j)k'(t_j) - s''(t_{j-1})k'(t_j) + s'''(t_j^-)k(t_j) - s'''(t_{j-1}^+)k(t_{j-1}) \right\}$$

$$= -\sum_{j=1}^n \left(s'''(t_j^-)\underbrace{k(t_j)}_{=0} - s'''(t_{j-1}^+)\underbrace{k(t_{j-1})}_{=0} \right) + \underbrace{\underbrace{s''(t_n)}_{=0}k'(t_n) - \underbrace{s''(t_0)}_{=0}k'(t_0)}_{\text{by telescopic sum}} = 0\,.$$

§ 5.4.3.7 (Weak locality of CSI)

perturbing y_j affects interpolant only near t_j

Ex: p.w. linear intp.: changing $y_j \Rightarrow$ interpolant changes only in $[t_{j-1}, t_{j+1}]$

(strict locality)*

* not satisfied for CSI

$I : \mathbb{R}^{n+1} \longrightarrow C^0([t_0, t_n]) \hat{=}$ linear interpolation scheme

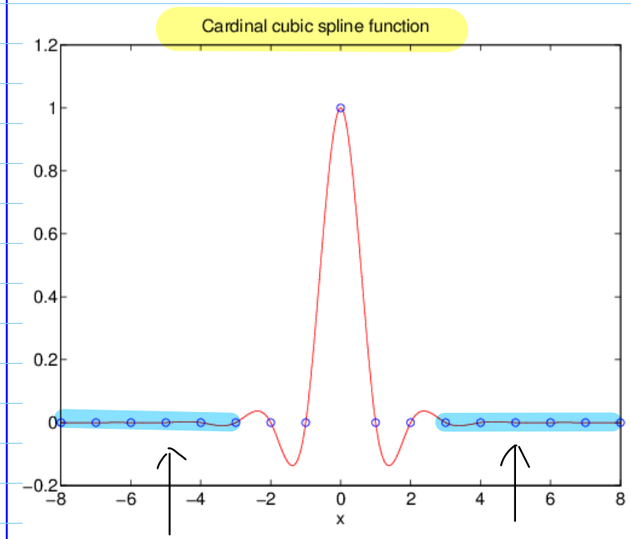
Locality of I can be seen from the cardinal basis $\{b_j\}_{j=0}^n \subset V$

$$I(y) = \sum_{j=0}^{n} y_j b_j$$

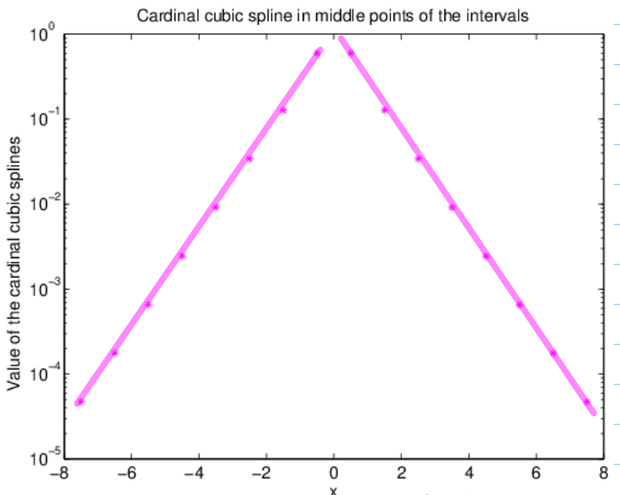
▷ Perturbing y_j impacts $I(y)$ only inside the support of b_j : $\operatorname{supp} b_j := \overline{\{t : b_j(t) \neq 0\}}$

Exp 5.4.3.9 : (Decay of cardinal splines for CSI)

[Equidistant nodes] ↳ cardinal basis fct. of $S_{3,m}$



"virtually zero"



lin - log plot

▷ $|b_j(t)| \leq |b_j(t_j)|\, e^{-\lambda |t - t_j|}$, $\lambda > 0$

$\hat{=}$ exponential decay

▷ Review questions 5.4.3.11

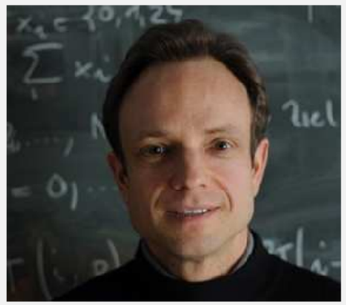
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 5.6: Trigonometric Interpolation

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 8, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.
- Complex arithmetic
- Relationship between trigonometric functions and the complex exponential function

Dependencies. Section 5.2.2, Section 5.2.3

Duration: 30 minutes

V. Data Interpolation and Data Fitting in 1D

5.6. Trigonometric Interpolation

Interpolate data points $(t_k, y_k) \in \mathbb{R}$, $k = 0, \dots, n$, obtained from sampling a T-periodic* function $f : \mathbb{R} \to \mathbb{R}$, $T > 0$ known

* $f(t+T) = f(t) \; \forall t \in \mathbb{R}$

Assume: $0 \le t_k < T$

▷ Want a T-periodic interpolant

Task: Given data points (t_i, y_i), $y_i \in \mathbb{K}$, $t_i \in [0, T[$, find a T-periodic function $f : \mathbb{R} \to \mathbb{K}$ (the interpolant), $f(t+T) = f(t) \; \forall t \in \mathbb{R}$, that satisfies the interpolation conditions

$$f(t_i) = y_i \,, \quad i = 0, \dots, n \,. \tag{5.6.0.2}$$

▷ Now: $T = 1$

(2)

5.6.1 Trigonometric Polynomials

Definition 5.6.1.1. Space of trigonometric polynomials

The vector space of 1-periodic **trigonometric polynomials** of degree $2n$, $n \in \mathbb{N}$, is given by

$$\mathcal{P}_{2n}^T := \operatorname{Span}\{t \mapsto \cos(2\pi jt), t \mapsto \sin(2\pi jt)\}_{j=0}^{n} \subset C^\infty(\mathbb{R}) .$$

→ drop for $j = 0$

Why polynomials:

(*) $\cos x = \frac{1}{2}(e^{ix} + e^{-ix})$, $\sin x = \frac{1}{2i}(e^{ix} - e^{-ix})$

$$q(t) = \alpha_0 + \sum_{j=1}^{n} \alpha_j \cos(2\pi jt) + \beta_j \sin(2\pi jt)$$

$$\begin{aligned} q(t) &= \alpha_0 + \sum_{j=1}^{n} \alpha_j \cos(2\pi jt) + \beta_j \sin(2\pi jt) \\ &= \alpha_0 + \frac{1}{2}\Big\{\sum_{j=1}^{n} (\alpha_j - \imath\beta_j) e^{2\pi\imath jt} + (\alpha_j + \imath\beta_j) e^{-2\pi\imath jt}\Big\} \quad \text{(*)} \\ &= \alpha_0 + \frac{1}{2}\sum_{j=-n}^{-1} (\alpha_{-j} + \imath\beta_{-j}) e^{2\pi\imath jt} + \frac{1}{2}\sum_{j=1}^{n} (\alpha_j - \imath\beta_j) e^{2\pi\imath jt} \\ &= e^{-2\pi\imath nt} \sum_{j=0}^{2n} \gamma_j e^{2\pi\imath jt} , \quad \text{with} \quad \gamma_j := \begin{cases} \frac{1}{2}(\alpha_{n-j} + \imath\beta_{n-j}) & \text{for } j = 0, \ldots, n-1 , \\ \alpha_0 & \text{for } j = n , \\ \frac{1}{2}(\alpha_{j-n} - \imath\beta_{j-n}) & \text{for } j = n+1, \ldots, 2n . \end{cases} \end{aligned} \tag{5.6.1.4}$$

$\gamma_j \in \mathbb{C}$

$q \in \mathcal{P}_{2n}^T \Rightarrow q(t) = e^{-2\pi\imath nt} p(e^{2\pi\imath t})$ with $p(z) = \sum_{j=0}^{2n} \gamma_j z^j$

Isomorphism: $\mathcal{P}_{2n} \longleftrightarrow \mathcal{P}_{2n}^T$

($\gamma_j \in \mathbb{C} \longleftrightarrow \alpha_j, \beta_j \in \mathbb{R}$)

Corollary 5.6.1.6. Dimension of $\mathcal{P}_{2n}^T$

The vector space $\mathcal{P}_{2n}^T$ has dimension $\dim \mathcal{P}_{2n}^T = 2n + 1$.

5.6.2 Reduction to Lagrange Interpolation

$$q \in \mathcal{P}_{2n}^T \;\Leftrightarrow\; q(t) = e^{-2\pi\imath nt} p(e^{2\pi\imath t}) \quad \text{for some} \quad p \in \mathcal{P}_{2n}\,. \tag{5.6.2.2}$$

I.C. : $q(t_k) = y_k \;\Leftrightarrow\; e^{-2\pi\imath n t_k} p(e^{2\pi\imath t_k}) = y_k$

$$\Leftrightarrow \quad p(z_k) = \widehat{y}_k$$

$z_k := e^{2\pi\imath t_k}$, $\widehat{y}_k = e^{2\pi\imath n t_k} y_k$, $k = 0,\dots,2n$

> Trigonometric interpolation through data points (t_k, y_k), $k = 0,\dots,2n$ $\;\Longleftrightarrow\;$ Lagrange polynomial interpolation through data points $(e^{2\pi\imath t_k}, e^{2\pi\imath n t_k} y_k)$, $k = 0,\dots,2n$

$(e^{2\pi\imath t_k}, e^{2\pi\imath n t_k} y_k) \in \mathbb{C}^2$, $(t_k, y_k) \in \mathbb{C}^2$

▷ Existence & uniqueness of trig. interpolant

▷ Runge functions for polynomial interpolation / evaluation.

NumCSE@ETHZ

5.6.3. Equidistant Trigonometric Interpolation

Now: data point $(t_k = \frac{k}{2n+1}, y_k) \in \mathbb{R}^2$, $k = 0,\dots,2n$

§ 5.6.3.1 : Computation of equidistant trigonometric interpolant

$$q \in \mathcal{P}_{2n}^T \;\Rightarrow\; q(t) = e^{-2\pi\imath nt} \cdot p(e^{2\pi\imath t}) \quad \text{with} \quad p(z) = \sum_{j=0}^{2n} \gamma_j z^j \in \mathcal{P}_{2n}\,, \tag{5.6.3}$$

I.C. : $q(t_k) = y_k$

$$\sum_{j=0}^{2n} \gamma_j \exp\left(2\pi\imath \frac{jk}{2n+1}\right) = (\mathbf{b})_k := \exp\left(2\pi\imath \frac{nk}{2n+1}\right) y_k\,, \quad k = 0,\dots,2n\,. \tag{5.6.3.2}$$

($(\mathbf{b})_k = y_k$)

$$\Updownarrow$$

$$\overline{\mathbf{F}}_{2n+1}\mathbf{c} = \mathbf{b}\,, \quad \mathbf{c} = [\gamma_0,\dots,\gamma_{2n}]^\top \quad \overset{\text{Lemma 4.2.1.14}}{\Rightarrow} \quad \mathbf{c} = \frac{1}{2n+1}\mathbf{F}_{2n+1}\mathbf{b}\,. \tag{5.6.3.3}$$

($\overline{\mathbf{F}}_{2n+1} = \frac{1}{2n+1}\mathbf{F}_{2n+1}$)

$(2n+1) \times (2n+1)$ (conjugate) Fourier matrix, see (4.2.1.13)!

$\mathbf{F}_{2n+1} = \left[e^{-\frac{2\pi\imath}{2n+1}jk}\right]_{j,k=0}^{2n}$

γ_j by DFT_{2n+1} !

cost $= O(n \log n)$ for FFT-based implementation

[vs. cost $= O(n^3)$ for Gaussian elimination]

§ 5.6.3.6 Evaluation of equidistant trigonometric interpolant

Task: Find an efficient way to evaluate the trigonometric polynomial

$$q(t) = \alpha_0 + \sum_{j=1} \alpha_j \cos(2\pi jt) + \beta_j \sin(2\pi jt)\,, \quad \alpha_j, \beta_j \in \mathbb{R}\,, \tag{5.6.1.3}$$

at *equidistant* points $\frac{k}{N}$, $N > 2n$. $k = 0, \ldots, N-1$. [many point, $N \gg n$]

α_j, β_j: given

$$q(t) = e^{-2\pi \imath nt} \sum_{j=0}^{2n} \gamma_j e^{2\pi \imath jt}\,, \quad \text{with} \quad \gamma_j = \begin{cases} \frac{1}{2}(\alpha_{n-j} + \imath\beta_{n-j}) & \text{for } j = 0, \ldots, n-1\,, \\ \alpha_0 & \text{for } j = n\,, \\ \frac{1}{2}(\alpha_{j-n} - \imath\beta_{j-n}) & \text{for } j = n+1, \ldots, 2n\,. \end{cases}$$

$$q(k/N) = e^{-2\pi i n k/N} \underbrace{\sum_{j=0}^{2n} \gamma_j e^{2\pi i \frac{jk}{N}}}_{= (\overline{F_N}\, c)_k}\,, \quad k = 0, \ldots, N-1$$

$$(c)_j := \begin{cases} \gamma_j\,, & j \le 2n \\ 0\,, & j > 2n \end{cases}$$

c via DFT_N

▷ Cost $= O(N \log N)$ for FFT-based impl., $N \to \infty$

▷ Review questions 5.6.3.10

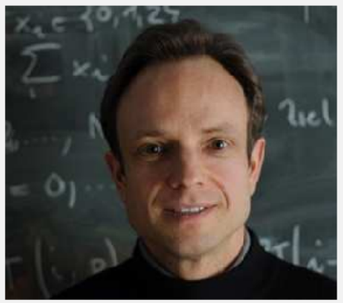
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 5.7: Least Squares Data Fitting

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 8, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 3.1, Section 3.6.1, Section 5.2

Duration: 26 minutes

V. Data Interpolation and Data Fitting in 1D

5.7. Least-Squares Data Fitting

Setting: Data point $(t_i, y_i) \in \mathbb{R}^2$, $i = 0, \ldots, m$, $t_i \in I$, affected measurement errors/uncertain

▷ Interpolation dubious

↓

replace with least-squares fitting

$V \triangleq$ vector space of admissible functions, $V \subset C^0(I)$

$$f \in \operatorname{argmin}_{g \in V} \sum_{i=0}^{m} |g(t_i) - y_i|^2 \qquad (*)$$

Example: polynomial fitting $V = \mathcal{P}_p$, $p < m$

②

Assume: $V = \operatorname{Span}\underbrace{\{b_1, \dots, b_n\}}_{\text{basis}}$, $n \le m$

$$(*) \iff g = \sum_{j=1}^{n} (x)_j b_j$$

$$x \in \operatorname*{argmin}_{z \in \mathbb{R}^n} \underbrace{\sum_{i=0}^{m} \left| \sum_{j=1}^{n} (z)_j b_j(t_i) - y_i \right|^2}_{= \|Az - y\|_2^2}$$

with $A = [b_j(t_i)]_{\substack{i=0,\dots,m \\ j=1,\dots,n}} \in \mathbb{R}^{m+1,n}$

$y = [y_0, \dots, y_m] \in \mathbb{R}^{m+1}$

$\hat{=}$ A std. linear least squares problem

x unique $\iff$ $\operatorname{rank}(A) = n$

Lemma 5.7.0.17. Unique solvability of linear least squares fitting problem

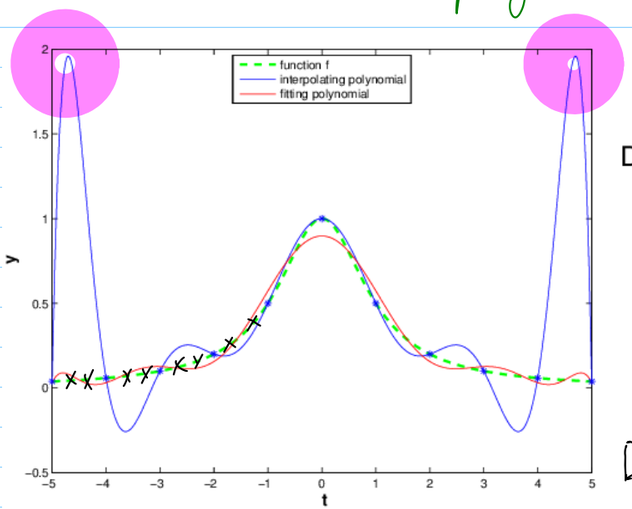
The scalar one-dimensional linear least squares fitting problem (5.7.0.10) *with* $\dim V_n = n$, V_n *the vector space of admissible functions, has a unique solution,* if and only if *there are* $t_{i_1}, \dots, t_{i_n}$ *such that*

$$\begin{bmatrix} b_1(t_{i_1}) & \dots & b_n(t_{i_1}) \\ \vdots & & \vdots \\ b_1(t_{i_n}) & \dots & b_n(t_{i_n}) \end{bmatrix} \in \mathbb{R}^{n,n} \quad \text{is invertible,} \tag{5.7.0.18}$$

which is independent of the choice of basis of V_n.

Existence of an n-subset of nodes for which interpolation into V will always have a unique solution

Exp. 5.7.0.23 (Polynomial interpolation vs. polynomial fitting)



Data from function $f(t) = \frac{1}{1+t^2}$,

- polynomial degree $d = 10$,
- interpolation through data points $(t_j, f(t_j))$, $j = 0, \dots, d$, $t_j = -5 + j$, see Ex. 5.2.4.3,
- fitting to data points $(t_j, f(t_j))$, $j = 0, \dots, 3d$, $t_j = -5 + j/3$.

▷ Better "shape fidelity" by fitting.

Rem 5.7.0.24 (Overfitting)

Polynomial fitting for $n \to m$:

⇒ Fitting becomes interpolation : poor shape fidelity

Overfitting

Fitting data with functions from a large space often produces poorer results (in terms of "shape preservation" and "generalization error") than the use of a smaller subspace for fitting.

NumCSE@ETHZ

§ 5.7.0.26 Data fitting with regularization

Idea : Suppress oscillations by "punishing" large derivatives :

$$f \in \operatorname*{argmin}_{g \in V} \left\{ \sum_{i=0}^{n} |g(t_i) - y_i|^2 + \alpha \int_a^b |g''(t)|^2 \mathrm{d}t \right\}, \quad \alpha > 0 \tag{5.7.0.27}$$

(α ↳ regularization parameter)

$V \hat{=}$ vector space of admissible functions

$V = \operatorname{Span}\{b_1, \dots, b_n\}$ [a basis]

$\left[g = \sum_{j=1}^{n} (x)_j b_j \right] \Rightarrow g'' = \sum_{j=1}^{n} (x)_j b_j''$

$$\phi(x) = \|\mathbf{A}x - \mathbf{y}\|_2^2 + \alpha x^\top \mathbf{M} x \to \min, \quad \phi : \mathbb{R}^n \to \mathbb{R} \tag{5.7.0.28}$$

$$\mathbf{A} := [b_j(t_i)]_{\substack{i=0,\dots,m \\ j=1,\dots,n}} \in \mathbb{R}^{m+1,n}, \quad \mathbf{M} := \left[\int_a^b b_j''(t)\, b_k''(t)\, \mathrm{d}t \right]_{j,k=1}^{n} \in \mathbb{R}^{n,n}. \tag{5.7.0.29}$$

$$\operatorname{grad} \phi(\mathbf{x}) = 2\mathbf{A}^\top \mathbf{A}x - 2\mathbf{A}^\top \mathbf{y} + 2\alpha \mathbf{M}x \overset{!}{=} \mathbf{0},$$

$$\Updownarrow$$

$$(\mathbf{A}^\top \mathbf{A} + \alpha \mathbf{M})\mathbf{x} = \mathbf{A}^\top \mathbf{y}. \quad [\text{generalized normal equation}] \tag{5.7.0.30}$$

▷ Review questions 5.7.0.31

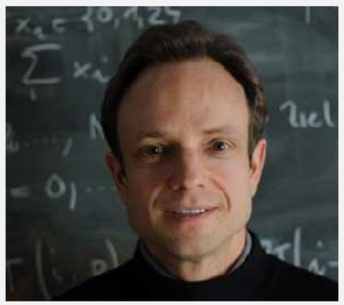
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 6.1: Approximation of Functions in 1D: Introduction

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 9, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.
- Functions and norms
- Object-oriented programming

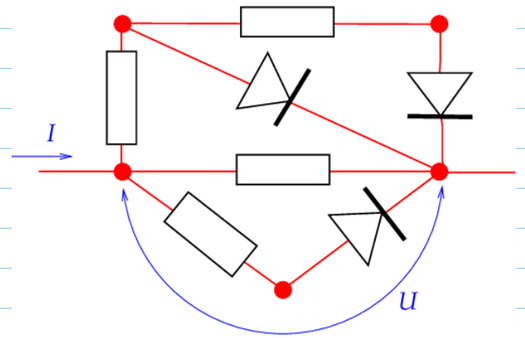
Dependencies. Section 5.1

Duration: 13 minutes

VI. Approximation of Functions in One Dimension

6.1. Introduction

Ex 6.1.0.3 (Model reduction in circuit simulation)



◁ 2-port circuit element described by $I = I(U)$

$I(U)$ is computable $\forall U$, but this is expensive

Idea: replace $U \mapsto I(U)$ with $U \mapsto \widetilde{I}(U)$ that
- allows fast evaluation
- is "close to" $I(U)$

$\widetilde{I} \hat{=}$ surrogate function

[Model reduction]

(2)

Implementation: Functor-type class

C++ code 6.1.0.4: Class describing a 2-port circuit element for circuit simulation

```
class CircuitElement {
  private:
  // internal data describing U ↦ Ĩ(U).
  public:
    // Constructor taking some parameters and building Ĩ
    CircuitElement(const Parameters &P);
    // Point evaluation operators for Ĩ and d/dU Ĩ
    double I(double U) const;
    double dIdU(double U) const;
};
```

Policy: Approximation by interpolation

Interpolation scheme + sampling → approximation scheme

$$f : I \subset \mathbb{R} \to \mathbb{K} \xrightarrow{\text{sampling}} (t_i, y_i := f(t_i))_{i=0}^{m} \xrightarrow{\text{interpolation}} \tilde{f} := \mathrm{I}_{\mathcal{T}}\mathbf{y} \quad (\tilde{f}(t_i) = y_i).$$

free choice of nodes $t_i \in I$: NEW!

Also new: notion of approximation error

§ 6.1.0.1 General Function Approximation Problem

only point evaluation possible

Approximation of functions: Generic view

Given: a **function** $\mathbf{f} : D \subset \mathbb{R}^n \mapsto \mathbb{R}^d$, $n, d \in \mathbb{N}$, often in procedural form, e.g. for $n = d = 1$, as `double f(double)` (→ Rem. 5.1.0.9).

Goal: Find a *"simple"* function $\tilde{\mathbf{f}} : D \mapsto \mathbb{R}^d$ such that the approximation error $\mathbf{f} - \tilde{\mathbf{f}}$ is *"small"*.

a function!

(p.w.) polynomials ‖ Needed: norms on function spaces

- Supremum norm: $\|g\|_\infty := \sup_{t \in D} |g(t)|$
- L^2-norm: $\|g\|_2^2 := \int_D |g(t)|^2 \, dt$

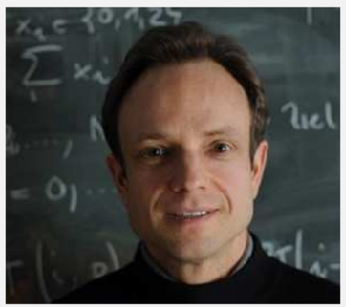
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 6.2: Polynomial Approximation: Theory

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 9, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.
- Elementary calculus of one variable

Dependencies. Section 5.2.1

Duration: 25 minutes

VI. Approximation of Functions in One Dimension

6.2. Approximation by Global Polynomials

Taylor polynomials: polynomials are "natural" for approximation

$$f(t) \approx \underbrace{f(t_0) + f'(t_0)(t-t_0) + \frac{f^{(2)}(t_0)}{2}(t-t_0)^2 + \cdots + \frac{f^{(k)}(t_0)}{k!}(t-t_0)^k}_{=:T_k(t)\in\mathcal{P}_k}, \quad \text{for some } t_0 \in I.$$

for $f \in C^k(I)$ $\hat{=}$ smoothness requirements

$\Rightarrow$ $f \sim T_k$ locally

6.2.1. Polynomial Approximation: Theory

For "large enough" degree : polynomials can approximate anything

Theorem 6.2.1.2. Uniform approximation by polynomials

For $f \in C^0([0,1])$, define the n-th Bernstein approximant as

$$p_n(t) = \sum_{j=0}^{n} f(j/n)\binom{n}{j} t^j (1-t)^{n-j}, \quad p_n \in \mathcal{P}_n. \tag{6.2.1.3}$$

It satisfies $\|f - p_n\|_\infty \to 0$ for $n \to \infty$. If $f \in C^m([0,1])$, then even $\left\|f^{(k)} - p_n^{(k)}\right\|_\infty \to 0$ for $n \to \infty$ and all $0 \le k \le m$.

☹ no further information

More smoothness ⟹ better estimate

Theorem 6.2.1.11. L^∞ polynomial best approximation estimate [Jackson thm.]

If $f \in C^r([-1,1])$ (r times continuously differentiable), $r \in \mathbb{N}$, then, for any polynomial degree $n \ge r$,

$$\inf_{p \in \mathcal{P}_n} \|f - p\|_{L^\infty([-1,1])} \le (1 + \pi^2/2)^r \frac{(n-r)!}{n!} \left\|f^{(r)}\right\|_{L^\infty([-1,1])},$$

where

$$\left\|f^{(r)}\right\|_{L^\infty([-1,1])} := \max_{x \in [-1,1]} |f^{(r)}(x)|.$$

Max-norm of best-approximation error

A simpler bound : reveals asymptotics for $n \to \infty$

$$\inf_{p \in \mathcal{P}_n} \|f - p\|_{L^\infty([-1,1])} \le C(r) n^{-r} \left\|f^{(r)}\right\|_{L^\infty([-1,1])}, \tag{6.2.1.13}$$

$C(r) > 0$, independent of f and n

(6.2.1.13) $\Rightarrow$ $\inf_{p \in \mathcal{P}_n} \|f - p\|_{L^\infty([-1,1])} = O(n^{-r})$ for $n \to \infty$.

Algebraic convergence with rate r for $n \to \infty$.

NumCSE@ETHZ

§§ 6.2.1.14 / 6.2.1.19 (Transformation of approximation error estimates)

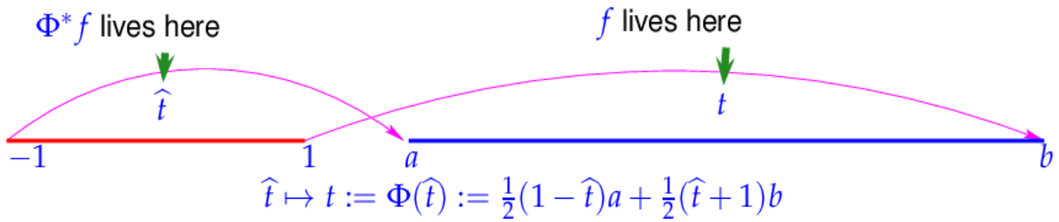
Jackson thm. on $[a,b]$?

Tool: Bijective affine transformation

$$\Phi : [-1,1] \to [a,b] \;,\quad \Phi(\hat{t}) := a + \tfrac{1}{2}(\hat{t}+1)(b-a) \;,\quad -1 \le \hat{t} \le 1 \;, \tag{6.2.1.15}$$

▷ Pullback of functions

$$\Phi^* : C^0([a,b]) \to C^0([-1,1]) \;,\quad \Phi^*(f)(\hat{t}) := (f \circ \Phi)(\hat{t}) := f(\Phi(\hat{t})) \;,\quad -1 \le \hat{t} \le 1 \;. \tag{6.2.1.16}$$



- $\| \Phi^* f \|_{\infty,[-1,1]} = \| f \|_{\infty,[a,b]}$
- $p \in \mathcal{P}_n \implies \Phi^* p \in \mathcal{P}_n$

NumCSE@ETHZ

Pullback & derivatives ($\Phi \leftrightarrow$ dilation)

Chain rule: $$\frac{d}{d\hat{t}}(\Phi^* f)(\hat{t}) \overset{\text{C.R.}}{=} \frac{df}{dt}(\Phi(\hat{t}))\frac{d\Phi}{d\hat{t}} = \frac{df}{dt}(\Phi(\hat{t})) \cdot \tfrac{1}{2}(b-a) \;,$$

$$(\Phi^* f)^{(r)} = \left(\frac{b-a}{2}\right)^r \Phi^*(f^{(r)}) \;. \tag{6.2.1.24}$$

Lemma 6.2.1.20 ▶

$$\left\| (\Phi^* f)^{(r)} \right\|_{L^\infty([-1,1])} = \left(\frac{b-a}{2}\right)^r \left\| f^{(r)} \right\|_{L^\infty([a,b])} \;,\quad f \in C^r([a,b]), r \in \mathbb{N}_0 \;. \tag{6.2.1.25}$$

§ 6.2.1.26 (Polynomial best-approximation error estimates on general intervals)

$$\inf_{p\in\mathcal{P}_n} \|f-p\|_{L^\infty([a,b])} \overset{(*)}{=} \inf_{p\in\mathcal{P}_n} \|\Phi^* f - p\|_{L^\infty([-1,1])} \qquad [\Phi^* \text{ preserves } \mathcal{P}_n \,!]$$

Jackson thm → Thm. 6.2.1.11

$$\le (1+\pi^2/2)^r \frac{(n-r)!}{n!} \left\| (\Phi^* f)^{(r)} \right\|_{L^\infty([-1,1])}$$

$$\overset{(6.2.1.24)}{=} (1+\pi^2/2)^r \frac{(n-r)!}{n!} \left(\frac{b-a}{2}\right)^r \left\| f^{(r)} \right\|_{L^\infty([a,b])} \;.$$

↑ length of interval

NumCSE@ETHZ

§6.2.1.14 (Transformation of polynomial approximation schemes)

Known: Approximation scheme $\hat{A} : C^0([-1,1]) \to \mathcal{P}_n$

Define: $A := (\phi^{-1})^* \circ \hat{A} \circ \phi^* : C^0([a,b]) \to \mathcal{P}_n$

= approximation scheme on $[a,b]$

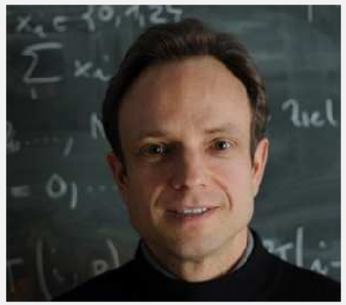
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 6.2.2: Error Estimates for Polynomial Interpolation

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 9, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.
- Calculus in one variable
- Complex analysis: theory of holomorphic functions

Dependencies. Section 5.2.1

Duration: minutes

VI. Approximation of Functions in One Dimension

6.2. Approximation by Global Polynomials

6.2.2. Error Estimates for Polynomial Interpolation

Lagrange interpolation

Interpolation scheme + sampling → approximation scheme

$$f : I \subset \mathbb{R} \to \mathbb{K} \xrightarrow{\text{sampling}} (t_i, y_i := f(t_i))_{i=0}^{m} \xrightarrow{\text{interpolation}} \widetilde{f} := \mathrm{I}_{\mathcal{T}}\mathbf{y} \quad (\widetilde{f}(t_i) = y_i)\,.$$

free choice of nodes $t_i \in I$

Definition 6.2.2.1. Lagrangian (interpolation polynomial) approximation scheme

Given an interval $I \subset \mathbb{R}$, $n \in \mathbb{N}$, a node set $\mathcal{T} = \{t_0, \ldots, t_n\} \subset I$, the Lagrangian (interpolation polynomial) approximation scheme $\mathsf{L}_\mathcal{T} : C^0(I) \to \mathcal{P}_n$ is defined by

$$\mathsf{L}_\mathcal{T}(f) := \mathsf{I}_\mathcal{T}(\mathbf{y}) \in \mathcal{P}_n \quad \text{with} \quad \mathbf{y} := (f(t_0), \ldots, f(t_n))^T \in \mathbb{K}^{n+1},$$
$$\mathsf{I}_\mathcal{T}(\mathbf{y})(t_j) = (\mathbf{y})_j, \quad j = 0, \ldots, n.$$

$f \longmapsto$ interpolating polynomial

6.2.2.1 Convergence of Interpolation Errors

↳ Consider sequencies of node sets

$(\mathcal{T}_n)_{n \in \mathbb{N}}$, $\#\mathcal{T} = n+1 \leftrightarrow$ pol. deg. $\to \infty$

Ex.: $I = [a,b]$: equidistant nodes

$$\mathcal{T}_n := \{t_j^{(n)} := a + (b-a)\frac{j}{n} : j = 0, \ldots, n\} \subset I. \tag{6.2.2.3}$$

▷ Find "sharp" bound $T : \mathbb{N}_0 \to \mathbb{R}^+$

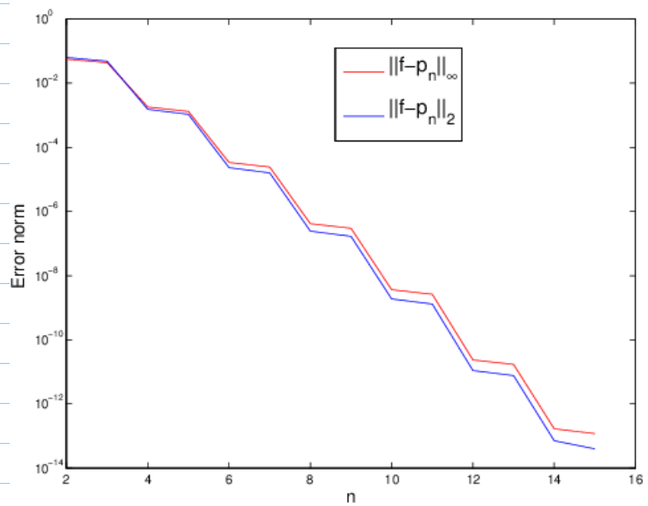
$$\exists C \neq C(n) > 0: \quad \|f - \mathsf{L}_{\mathcal{T}_n} f\| \leq C\, T(n) \quad \text{for } n \to \infty. \tag{6.2.2.6}$$

C: independent of n!

↳ "generic constant", generally unknown

Exp. 6.2.2.4 : (Asymptotic behavior of Lagrange interpolation error)

$f(t) := \sin t$, $I = [0, \pi]$, $\mathcal{T}_n = \{\frac{\pi}{n} j\}_{j=0}^n$



△ $\|f - \mathsf{L}_{\mathcal{T}_n} f\|_\infty$ and $\|f - \mathsf{L}_{\mathcal{T}_n} f\|_2$ ↑ seems to be governed by a law

§ 6.2.2.5 (Classification of asymptotic behavior of the interpolation error)

Definition 6.2.2.7. Types of asymptotic convergence of approximation schemes

Writing $T(n)$ for the bound of the norm of the interpolation error according to (6.2.2.6) we distinguish the following *types of asymptotic behavior*:

$\|f - p_n\| = O(n^{-p})$ for $n \to \infty$

$\exists\, p > 0$: $T(n) \leq n^{-p}$: algebraic convergence, with rate $p > 0$,

$\exists\, 0 < q < 1$: $T(n) \leq q^n$: exponential convergence, $\forall n \in \mathbb{N}$.

$\|f - p_n\| = O(q^n)$ for $n \to \infty$

The bounds are assumed to be sharp in the sense, that no bounds with larger rate p (for algebraic convergence) or smaller q (for exponential convergence) can be found.

§ 6.2.2.9 (Determining type of convergence in numerical experiments)

"Measured": $(n_i, \varepsilon_i) \in \mathbb{N} \times \mathbb{R}^+$ for $i = 1, 2, \ldots$

ε_i ← error norm

❶ Conjectured: algebraic convergence: $\epsilon_i \approx Cn^{-p}$, $C > 0$ unknown

$$\log(\epsilon_i) \approx \log(C) - p \log n_i$$ (affine linear in log-log scale).

$\log(\varepsilon_i / \varepsilon_{i-1}) \approx -p \log(n_i / n_{i-1}) \longrightarrow p$

$n \nearrow$? for $\varepsilon \longrightarrow \varepsilon/2$

❶ Conjectured: exponential convergence: $\epsilon_i \approx C \exp(-\beta n_i)$

$$\log \epsilon_i \approx \log(C) - \beta n_i$$ (affine linear in lin-log scale).

$n \nearrow$? for $\varepsilon \longrightarrow \varepsilon/2$

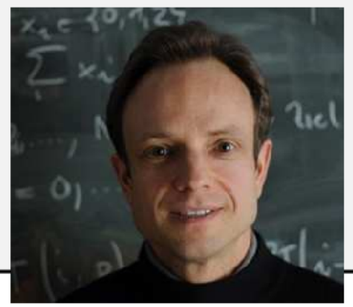
Course Video

Section 6.2.2.2: Error Estimates for Polynomial Interpolation: Interpolands of Finite Smoothness

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 11, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- Calculus in 1 variable
- Mean-value theorem in 1D

Dependencies. Section 6.2.2.1, Section 5.2.4

Duration: 30 minutes

VI. Approximation of Functions in One Dimension

6.2. Approximation by Global Polynomials

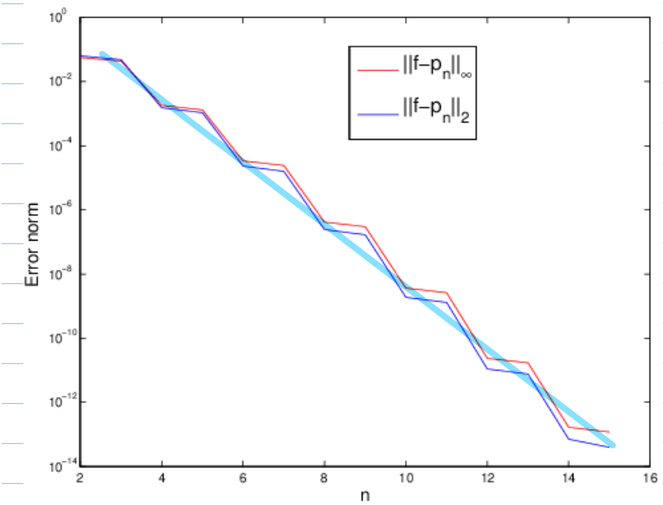
6.2.2. Error Estimates for Polynomial Interpolation

Exp. 6.2.2.4: (Asymptotic behavior of Lagrange interpolation error)

$f(t) := \sin t$, $I = [0, \pi]$, $\mathcal{T}_n = \{ \frac{\pi}{n} j \}_{j=0}^{n}$

↑ equid. nodes

(2)

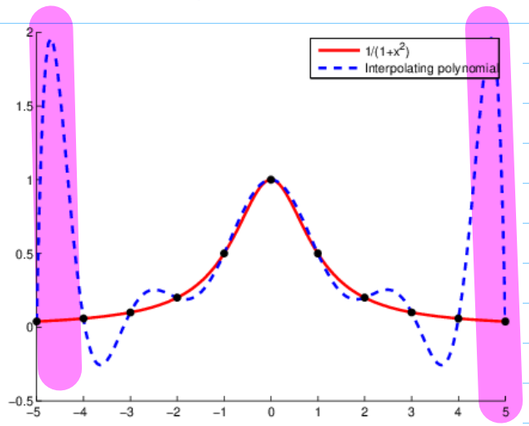


◁ $\|f - L_{\mathcal{T}_n} f\|_\infty$ and $\|f - L_{\mathcal{T}_n} f\|_2$

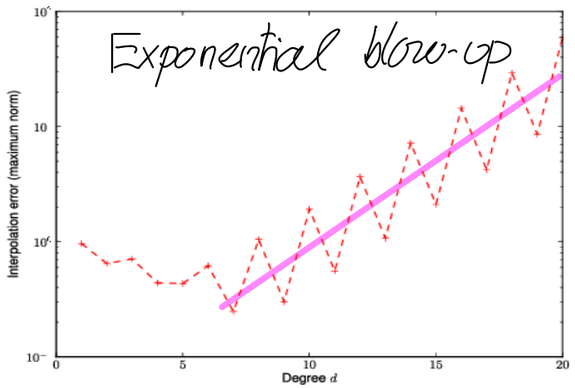
◁ Exponential convergence

Ex. 6.2.2.11 (Runge's counterexample) equid. nodes

$f(t) = \frac{1}{1+t^2}$, $I = [-5,5]$, $\mathcal{T}_n = \{-5 + \frac{10}{n} j\}_{j=0}^{n}$



Interpolating polynomial, $n = 10$



Approximate $\|f - L_{\mathcal{T}_n} f\|_\infty$ on $[-5,5]$

6.2.2.2. Interpolands of Finite Smoothness

↳ class C^k

Main theoretical result :

Theorem 6.2.2.15. Representation of interpolation error ,

We consider $f \in C^{n+1}(I)$ and the Lagrangian interpolation approximation scheme (→ Def. 6.2.2.1) for a node set $\mathcal{T} := \{t_0, \ldots, t_n\} \subset I$. Then, (not known)

for every $t \in I$ there exists a $\tau_t \in]\min\{t, t_0, \ldots, t_n\}, \max\{t, t_0, \ldots, t_n\}[$ such that

$$\underbrace{f(t) - L_{\mathcal{T}}(f)(t)}_{\text{interpolation error}} = \frac{f^{(n+1)}(\tau_t)}{(n+1)!} \cdot \underbrace{\prod_{j=0}^{n}(t - t_j)}_{=: W \,\hat{=}\, \text{nodal polynomial}} . \qquad (6.2.2.16)$$

$W|_{\mathcal{T}} = 0$

Proof: Fix $t \in I \setminus \mathcal{T}$

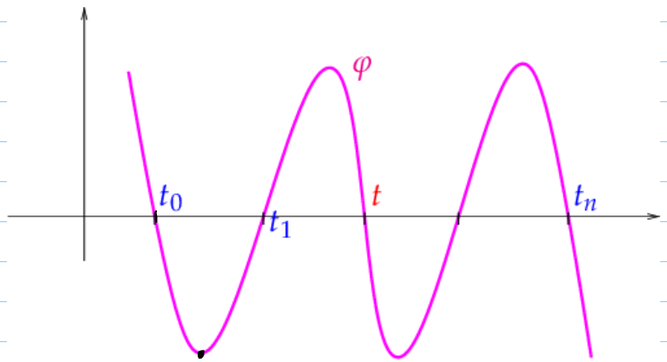
Choose $c \in \mathbb{R}$: $f(t) - L_{\mathcal{T}} f(t) - c\,\overbrace{w(t)}^{\neq 0} = 0$

$\varphi(x) := f(x) - L_{\mathcal{T}} f(x) - c\,w(x) \in C^{n+1}$

$\varphi(t_j) = 0$ by I.C. & $\varphi(t) = 0$ by choice of c

▷ φ has (at least) $n+2$ zeros

③



φ' has at least $n+1$ zeros

$\varphi^{(k)}$ has at least $n+2-k$ distinct zeros

$\varphi^{(n+1)}$ has a zero τ_t

$$\varphi^{(n+1)}(x) = f^{(n+1)}(x) - 0 - c(n+1)!$$
$$\varphi^{(n+1)}(\tau_t) = f^{(n+1)}(\tau_t) - c(n+1)! = 0$$

□

Thm. 6.2.2.15 ⇒ ↑ max on both sides

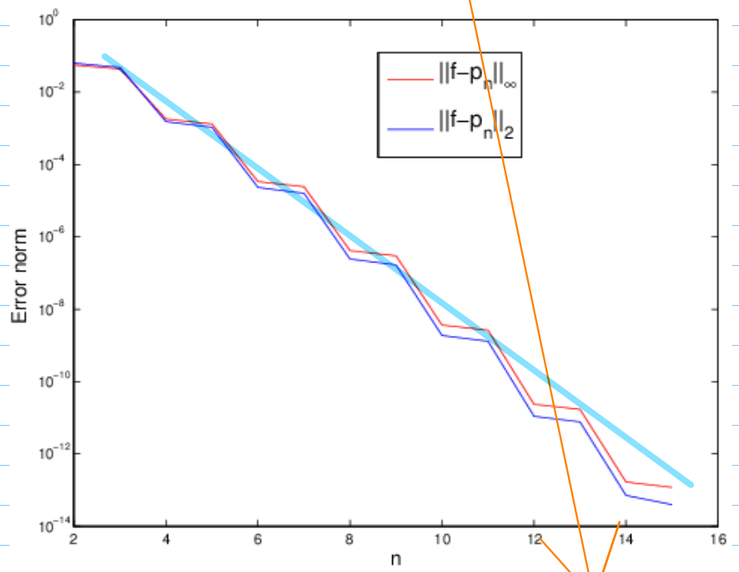
$$\|f - \mathsf{L}_\mathcal{T} f\|_{L^\infty(I)} \leq \frac{\|f^{(n+1)}\|_{L^\infty(I)}}{(n+1)!} \max_{t\in I} |(t-t_0)\cdot\cdots\cdot(t-t_n)| \,. \quad (6.2.2.21)$$

Quantitative interpolation error estimates rely on smoothness!

Ex 6.2.2.23 (→ Exp. 6.2.2.4)

$f(t) := \sin t$, $I = [0,\pi]$, $\mathcal{T}_n = \{\frac{\pi}{n} j\}_{j=0}^{n}$

↳ $\|f^{(k)}\|_\infty \leq 1 \quad \forall k$



"Super" Exponential convergence

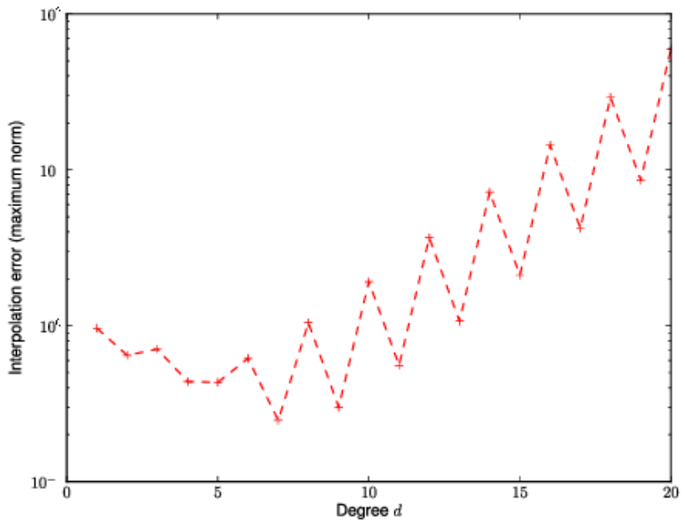
$$\left\|f^{(k)}\right\|_{L^\infty(I)} \leq 1, \quad \forall k \in \mathbb{N}_0 \quad \Rightarrow \quad \|f-p\|_{L^\infty(I)} \leq \frac{1}{(1+n)!} \max_{t\in I} \underbrace{|(t-0)(t-\tfrac{\pi}{n})(t-\tfrac{2\pi}{n})\cdot\cdots\cdot(t-\pi)|}_{\max \text{ at } t=\frac{\pi}{2n}}$$
$$\leq \frac{1}{n+1}\left(\frac{\pi}{n}\right)^{n+1}.$$

↳ → 0 for $n \to \infty$

Ex. 6.2.2.24 (Runge's example, Exp. 6.2.2.11)

$f(t) = \frac{1}{1+t^2}$, $I = [-5,5]$, $\mathcal{T}_n = \{-5 + \frac{10}{n} j\}_{j=0}^{n}$

↑ $|f^{(k)}(t)| = 2^k k!\, O(|t|^{-2-k})$ for $k \to \infty$



Approximate $\|f - \mathsf{L}_{\mathcal{T}_n} f\|_\infty$ on $[-5,5]$

↓

Blow-up of bound (6.2.2.21)

§ 6.2.2.28 (Interpolation error estimates and Lebesgue constants $\lambda_{\mathcal{T}}$)

node set ↓

$$\lambda_{\mathcal{T}} := \|\mathsf{I}_{\mathcal{T}}\|_{\infty\to\infty} := \sup_{\mathbf{y}\in\mathbb{R}^{n+1}\setminus\{0\}} \frac{\|\mathsf{I}_{\mathcal{T}}(\mathbf{y})\|_{L^\infty(I)}}{\|\mathbf{y}\|_\infty},$$

! key observation: $\mathsf{L}_{\mathcal{T}}$ preserves polynomials

$$\mathsf{L}_{\mathcal{T}} p = \mathsf{I}_{\mathcal{T}}([p(t)]_{t\in\mathcal{T}}) = p \quad \forall p \in \mathcal{P}_n. \qquad n := \#\mathcal{T} - 1 \tag{6.2.2.30}$$

$$\begin{aligned} \|f - \mathsf{L}_{\mathcal{T}} f\|_\infty &= \|(f-p) - (\mathsf{L}_{\mathcal{T}} f - \mathsf{L}_{\mathcal{T}} p)\|_\infty \\ &\le \|(\mathrm{Id} - \mathsf{L}_{\mathcal{T}})(f-p)\|_\infty \\ &\le (1 + \lambda_{\mathcal{T}})\|f-p\|_\infty \quad \forall p \in \mathcal{P}_n \end{aligned}$$

$$\|\mathsf{L}_{\mathcal{T}} g\|_\infty \le \lambda_{\mathcal{T}} \max_{t\in\mathcal{T}} |g(t)| \le \lambda_{\mathcal{T}} \|g\|_\infty$$

$$\triangleright \quad \|f - \mathsf{L}_{\mathcal{T}} f\|_\infty \le (1+\lambda_{\mathcal{T}}) \inf_{p\in\mathcal{P}_n} \|f-p\|_\infty$$

interpolation error ↑ — best-approx error

ETH Lecture 401-0663-00L Numerical Methods for CSE

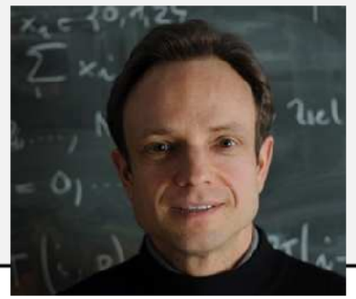
Course Video

Section 6.2.2.3: Error Estimates for Polynomial Interpolation: Analytic Interpolands

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 11, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- Complex analysis: path integrals
- Complex analysis: residue theorem

Dependencies. Section 6.2.2.1

Duration: minutes

VI. Approximation of Functions in One Dimension

6.2. Approximation by Global Polynomials

6.2.2. Error Estimates for Polynomial Interpolation

Ex. 6.2.2.23 / Exp. 6.2.2.4

Lagrange polynomial approximation of $f(t) = \sin t$ on $[0,\pi]$ with equidistant nodes

▷ (super-) exponential convergence

$$\|f - L_{\mathcal{T}_n} f\|_\infty \leq \frac{1}{n+1}\left(\frac{\pi}{n}\right)^{n+1} \quad \forall n \in \mathbb{N}$$

$\in C^\infty(\mathbb{R})$

6.2.2.3. Analytic Interpolands

$$\sin t = \underbrace{\sum_{k=0}^{\infty} (-1)^k \frac{1}{(2k+1)!} t^{2k+1}}_{\hat{=}\ \text{power series} = \text{"polynomial of degree } \infty\text{"}}, \quad \forall t \in \mathbb{R}$$

Definition 6.2.2.35. Real-analytic functions

A function $f \in C^\infty(I)$ defined on an open interval $I \subset \mathbb{R}$ is called **(real-)analytic** on I, if it possesses a convergent Taylor series at every point $t_0 \in I$:

$$\forall t_0 \in I: \ \exists \rho = \rho(t_0) > 0: \ f(t) = \sum_{n=0}^{\infty} \frac{f^{(n)}(t_0)}{n!}(t-t_0)^n \quad \forall t \in I : \ |t-t_0| < \rho(t_0) .$$

$\rho(t_0)$ is called the **radius of convergence** of the Taylor series.

local property

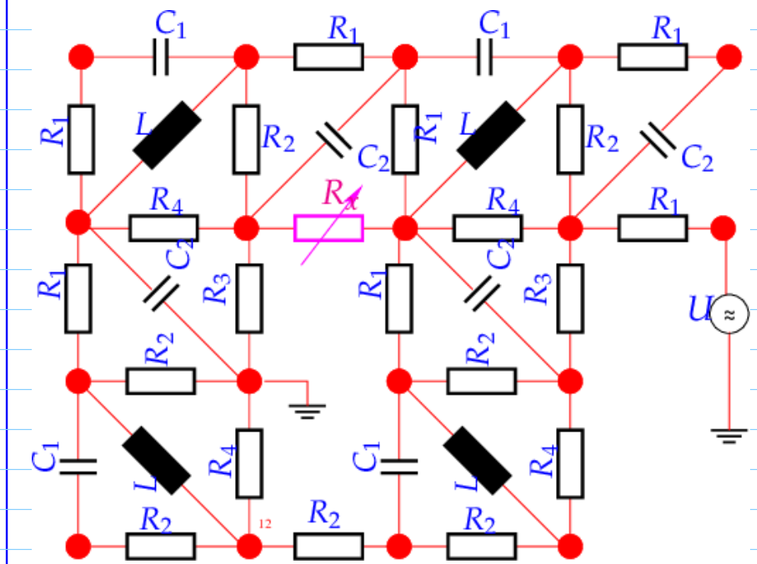
Ex 6.2.2.37 (Hump function)

$$f(t) = \frac{1}{1+t^2} = \sum_{k=0}^{\infty} (-1)^k t^{2k} \quad \text{for } |t| \le 1$$

$\hat{=}$ a local power series representation

NumCSE@ETHZ

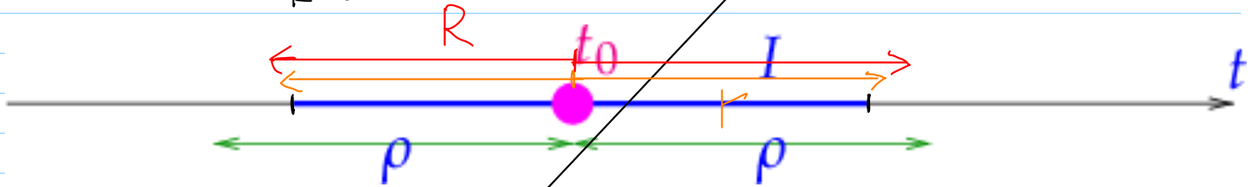
Rem 6.2.2.40 (Analytic functions everywhere)



◁ linear RLC circuit
All currents/voltages are analytic functions of R_x !

§ 6.2.2.41 (Approximation by truncated power series)

$$f(t) = \sum_{k=0}^{\infty} a_k (t-t_0)^k \qquad \forall\, |t-t_0| < \rho$$



$$p_n(t) := \sum_{k=0}^{n} a_k (t-t_0)^k \in \mathcal{P}_n . \qquad (6.2.2.43)$$

③

$r < R < S \hat{=}$ radius of convergence

$$\sum_{k=0}^{\infty} |a_k| R^k =: C < \infty$$

Δ-ineq ↓

$$|f(t) - p_n(t)| = \Big| \sum_{k=n+1}^{\infty} a_k (t-t_0)^k \Big| \le \sum_{k=n+1}^{\infty} |a_k| R^k \left(\frac{t-t_0}{R}\right)^k \le C \left(\frac{r}{R}\right)^{n+1} . \quad \forall t \in I$$

$$\|f - p_n\|_{L^\infty(I)} \le C q^{n+1} \quad \forall n \in \mathbb{N} \quad \text{with} \quad q := r/R < 1 . \tag{6.2.2.46}$$

$\hat{=}$ exponential convergence

Required: globally cvg. power series

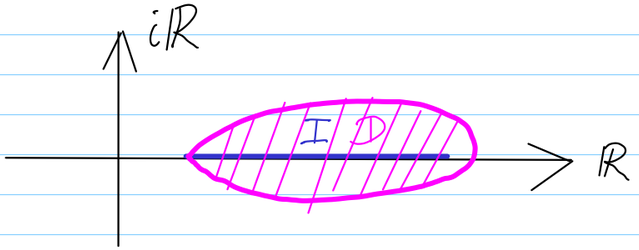
Tool: Complex analysis

$$f(t) = \sum_{n=0}^{\infty} a_n (t-t_0)^n , \quad a_n \in \mathbb{R} , \quad \forall t \in \mathbb{R} : |t-t_0| < \rho(t_0) ,$$

$$f(z) = \sum_{n=0}^{\infty} a_n (z-t_0)^n , \quad \forall z \in \mathbb{C} : |z-t_0| < \rho(t_0) .$$

Real- and complex-analytic functions

Every real-analytic function on $I \subset \mathbb{R}$ can be extended to a complex-analytic function on some open set $D \subset \mathbb{C}$ with $I \subset D$.



Definition 6.2.2.48. Analyticity of a complex valued function

let $D \subset \mathbb{C}$ be an *open* set in the complex plane. A function $f : D \to \mathbb{C}$ is called (complex)-analytic/holomorphic in D, for *every* point $z \in D$ one can find $\rho(z) > 0$ and $a_k \in \mathbb{C}$, $k \in \mathbb{N}_0$, such that

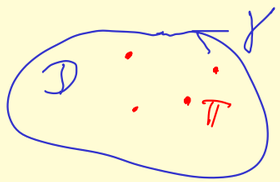
$$f(w) = \sum_{k=0}^{\infty} a_k (w-z)^k \quad \forall w \in D : |z-w| < \rho(z) .$$

Theorem 6.2.2.50. Residue theorem

Let $D \subset \mathbb{C}$ be an open set, $G \subset D$ a closed set contained in D, $\gamma := \partial G$ its (piecewise smooth and oriented) boundary, and Π a finite set contained in the interior of G.

Then for each function f that is analytic in $\overline{D} \setminus \Pi$ holds

$$\frac{1}{2\pi\imath} \int_\gamma f(z)\, dz = \sum_{p \in \Pi} \operatorname{res}_p f ,$$



where $\operatorname{res}_p f$ is the residual of f in $p \in \mathbb{C}$.

complex path integral $\in \mathbb{C}$

$$\left| \int_\gamma f(z) dz \right| \le |\gamma| \cdot \max_{z \in \gamma} |f(z)|$$

↑ length of γ

NumCSE@ETHZ

Lemma 6.2.2.53. Residual formula for quotients

let g and h be complex valued functions that are both analytic in a neighborhood of $p \in \mathbb{C}$, and satisfy $h(p) = 0$, $h'(p) \neq 0$. Then

$$\operatorname{res}_p \frac{g}{h} = \frac{g(p)}{h'(p)} .$$

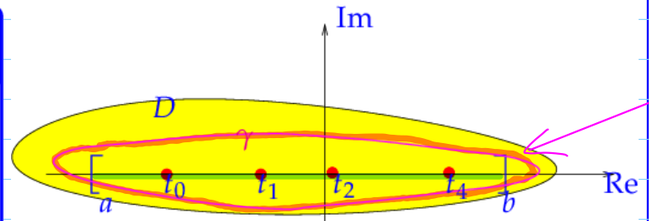
↳ has a pole in p

§ 6.2.2.54 (Residue remainder formula for Lagrange interpolation)

$I = [a,b]$, node set $\mathcal{T} = \{t_0, \dots, t_n\} \subset I$

Assumption 6.2.2.55. Analyticity of interpoland

We assume that the interpoland $f : I \to \mathbb{C}$ *can be extended* to a function $f : D \subset \mathbb{C} \to \mathbb{C}$, which is analytic (→ Def. 6.2.2.48) on the open set $D \subset \mathbb{C}$ with $[a,b] \subset D$.



↳ $[a,b]$ has a positive distance from ∂D

$$L_{\mathcal{T}} f(t) = \sum_{j=0}^{n} f(t_j) L_j(t)$$

Nodal polynomial $w(t) = \prod_{k=0}^{n} (t - t_k)$

$$w'(t_j) = \prod_{\substack{k=0 \\ k \neq j}} (t_j - t_k) \quad \text{[product rule]}$$

$$L_j(t) = \prod_{\substack{k=0 \\ k \neq j}}^{n} \frac{(t - t_k)}{(t_j - t_k)} = \frac{w(t)}{(t - t_j) w'(t_j)}$$

Auxiliary function, fix $t \in I \setminus \mathcal{T}$

$$g_t(z) = \frac{f(z)}{(z - t) w(z)} \qquad z \in D \setminus \Pi, \quad \Pi := \{t_0 \dots, t_n, t\}$$

↓

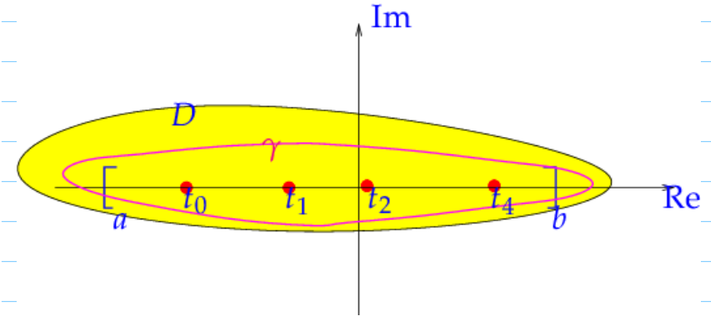
Analytic on $D \setminus \Pi$, denominator has only simple zeros

$$\frac{1}{2\pi i} \int_{\gamma} g_t(z)\, dz \overset{\text{res. Thm}}{=} \operatorname{res}_t g_t + \sum_{j=0}^{n} \operatorname{res}_{t_j} g_t$$

$$= \frac{f(t)}{w(t)} + \sum_{j=0}^{n} \frac{f(t_j)}{(t_j - t) w'(t_j)} \quad | \cdot w$$

$$f(t) = \underbrace{\underbrace{\sum_{j=1}^{n} f(t_j) \frac{w(t)}{(t_j - t) w'(t_j)}}_{\text{Lagrange polynomial !}}}_{\text{polynomial interpolant !}} + \underbrace{\frac{w(t)}{2\pi \imath} \int_{\gamma} g_t(z)\, \mathrm{d}z}_{\text{interpolation error !}} . \qquad (6.2.2.57)$$

$$|f(t) - \mathsf{L}_T f(t)| = \left| \frac{w(t)}{2\pi\imath} \int_\gamma \frac{f(z)}{(z-t)w(z)}\,\mathrm{d}z \right| \le \frac{|\gamma|}{2\pi} \frac{\max_{a\le\tau\le b}|w(\tau)|}{\min_{z\in\gamma}|w(z)|} \cdot \frac{\max_{z\in\gamma}|f(z)|}{\operatorname{dist}([a,b],\gamma)} \,. \quad (6.2.2.58)$$



For studying conv of interp. error we need only existence of D and γ.

Theorem 6.2.2.60. Composition and products of analytic functions

If $f, h : D \subset \mathbb{C} \to \mathbb{C}$ and $g : U \subset \mathbb{C} \to \mathbb{C}$ are analytic in the open sets D and U, respectively, then

(i) the composition $f \circ g$ is analytic in $\{z \in U : g(z) \in D\}$,

(ii) the product $f \cdot h$ is analytic on D.

Ex: $t \longmapsto \sqrt{1+t^2}$ analytic in $\mathbb{C}\setminus\pm[1,\infty]i$

Note: $z \to \sqrt{z}$ is analytic in $\mathbb{C}\setminus]-\infty,0]$

$1+t^2 = -\xi$, $\xi \ge 0$

§ 6.2.2.59 (Determining the domain of analyticity)

Ex: $f(t) = \frac{1}{1+t^2} = \frac{1}{(1+it)(1-it)}$

▷ Analytic in $\mathbb{C}\setminus\{\pm i\}$

ETH Lecture 401-0663-00L Numerical Methods for CSE

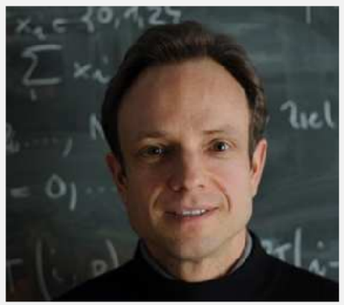
Course Video

Section 6.2.3.1: Chebychev Interpolation: Motivation and Definition

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 11, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 6.2.2.2

Duration: 21 minutes

VI. Approximation of Functions in One Dimension

6.2. Approximation by Global Polynomials

6.2.3. Chebychev Interpolation

Interpolation scheme + sampling → approximation scheme

$$f : I \subset \mathbb{R} \to \mathbb{K} \xrightarrow{\text{sampling}} (t_i, y_i := f(t_i))_{i=0}^{m} \xrightarrow{\text{interpolation}} \widetilde{f} := \mathrm{I}_{\mathcal{T}}\mathbf{y} \quad (\widetilde{f}(t_i) = y_i)\,.$$

free choice of nodes $t_i \in I$

"Bad choice": equidistant nodes, Exp. 6.2.2.11

(2)

6.2.3.1 Motivation and Definition

Goal: "A priori optimal nodes"
↳ independent of f

Thm. 6.2.2.15 $\Rightarrow$ $$\|f - \mathrm{I}_{\mathcal{T}} f\|_{L^\infty(I)} \leq \frac{\|f^{(n+1)}\|_{L^\infty(I)}}{(n+1)!} \underbrace{\max_{t\in I} |(t-t_0)\cdot\cdots\cdot(t-t_n)|}_{= \|w\|_{\infty,I},\ w(t) = \prod_{j=0}^{n}(t-t_j)} . \quad (6.2.2.22)$$

↳ make as small as possible

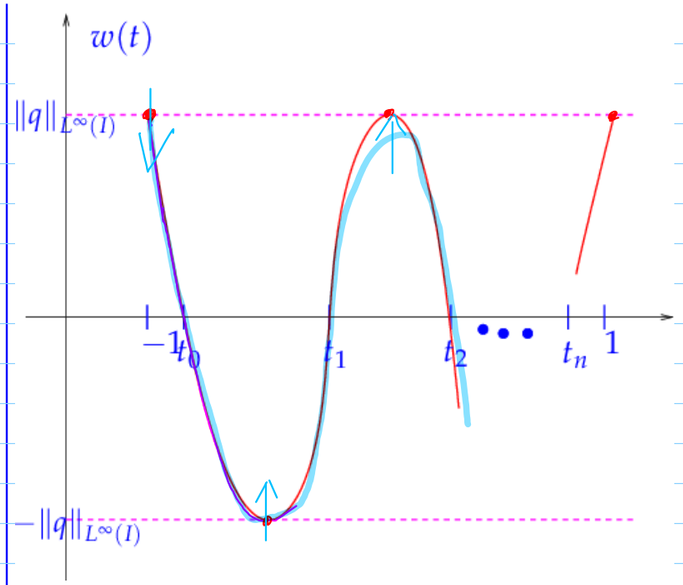
Optimal choice of interpolation nodes independent of interpoland

Idea: choose nodes $t_0, \ldots, t_n$ such that $\|w\|_{L^\infty(I)}$ is minimal!

This is equivalent to finding a polynomial $q \in \mathcal{P}_{n+1}$
- with leading coefficient $= 1$,
- such that it minimizes the norm $\|q\|_{L^\infty(I)}$.

▶ Then choose nodes $t_0, \ldots, t_n$ as zeros of q.
(caution: all t_j must lie in I!)

First $I = [-1, 1]$



Necessary shape requirements:

n+2 equal-|·| extrema with alternating signs

Definition 6.2.3.3. Chebychev polynomials →

The n^{th} Chebychev polynomial is $T_n(t) := \cos(n \arccos t)$, $-1 \leq t \leq 1, n \in \mathbb{N}$.

▷ $\|T_n\|_{\infty,[-1,1]} = 1$

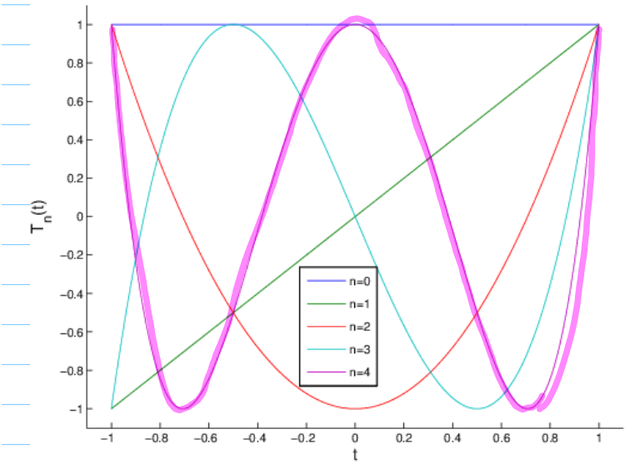
Theorem 6.2.3.4. 3-term recursion for Chebychev polynomials →

The function T_n defined in Def. 6.2.3.3 satisfy the 3-term recursion

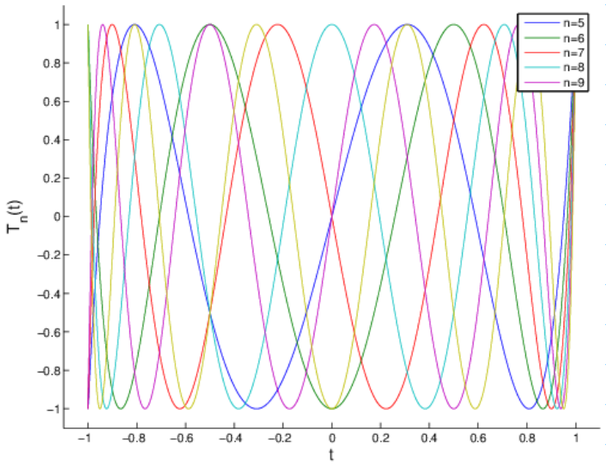
$$T_{n+1}(t) = 2t\,T_n(t) - T_{n-1}(t), \quad T_0 \equiv 1, \quad T_1(t) = t, \quad n \in \mathbb{N}. \quad (6.2.3.5)$$

Proof: $\cos((n+1)x) = 2\cos nx \cdot \cos x - \cos((n-1)x)$

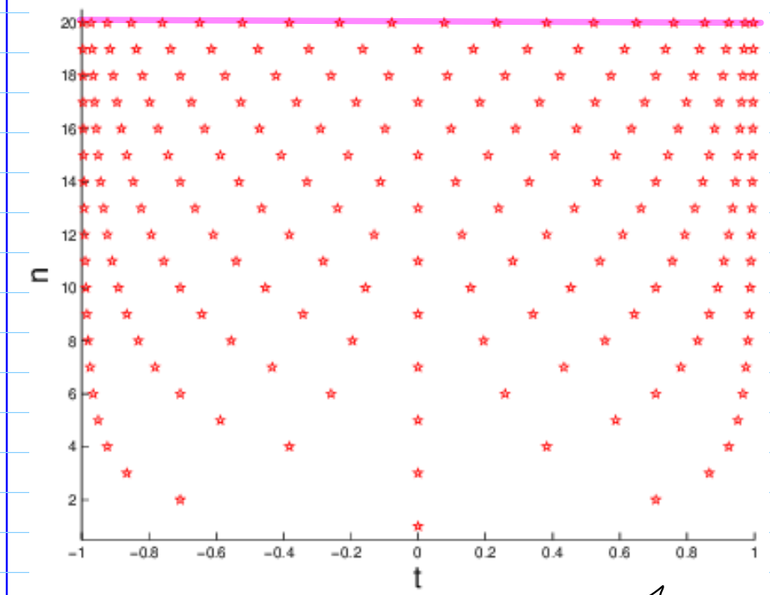
▷ $T_n \in \mathcal{P}_n$, leading coefficient 2^{n-1}



Chebychev polynomials $T_0, \ldots, T_4$



Chebychev polynomials $T_5, \ldots, T_9$



◁ Cheb nodes cluster at ±1

on $[a,b]$: $t_j = \phi(\hat{t}_j) = a + \frac{1}{2}(b-a)(1+\hat{t}_j)$

Theorem 6.2.3.8. Minimax property of the Chebychev polynomials ,

The polynomials T_n from Def. 6.2.3.3 minimize the supremum norm in the following sense:

$$\|T_n\|_{L^\infty([-1,1])} = \inf\{\|p\|_{L^\infty([-1,1])} : p \in \mathcal{P}_n, p(t) = 2^{n-1}t^n + \cdots\}, \quad \forall n \in \mathbb{N}.$$

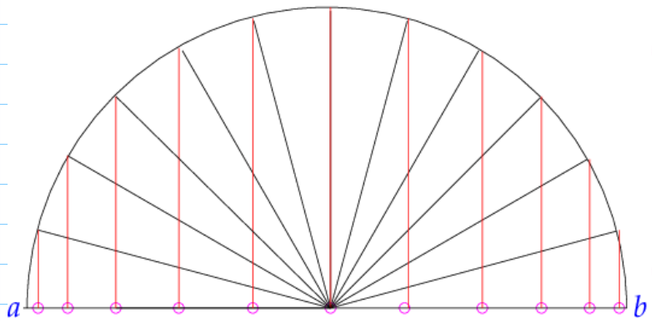
↑ leading coefficient

▷ A priori optimal interpolation nodes $\hat{t}_j$: zeros of T_{n+1}

$$\hat{t}_j = \cos\left(\frac{2j+1}{2n+2}\pi\right), \quad j = 0, \ldots, n$$

↑ Chebychev nodes in $[-1,1]$

Nodal polynomial : $w(t) = 2^{-n} T_{n+1}(t)$



The Chebychev nodes in the interval $I = [a,b]$ are

$$t_k := a + \frac{1}{2}(b-a)\left(\cos\left(\frac{2k+1}{2(n+1)}\pi\right) + 1\right), \quad (6.2.3.13)$$

$k = 0, \ldots, n$.

↓

Chebychev interpolation

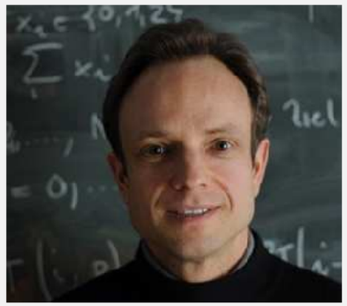
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 6.2.3.2: Chebychev Interpolation Error Estimates

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 13, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 6.2.3.1, Section 6.2.2.3

Duration: 26 minutes

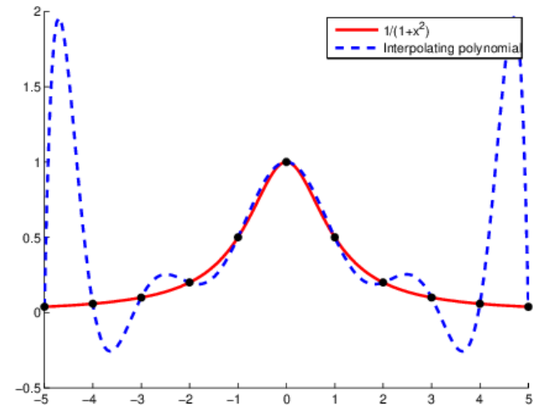
VI. Approximation of Functions in One Dimension

6.2. Approximation by Global Polynomials

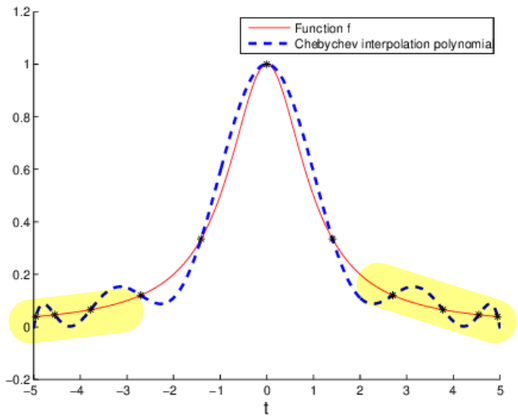
6.2.3. Chebychev Interpolation

Ex. 6.2.3.15 (Chebychev vs. equidistant polynomial intp.)

$f(t) = \frac{1}{1+t^2}$;



Equidistant nodes



Chebychev nodes

(2)

6.2.3.2. Chebychev Interpolation Error Estimates

§ 6.2.3.14 : (Finite-smoothness error estimates for Chebychev interpolation)

Recall Thm. 6.2.2.15:

$I = [-1,1]$

$$\|f - \mathsf{L}_T f\|_{L^\infty(I)} \le \frac{1}{(n+1)!}\left\|f^{(n+1)}\right\|_{L^\infty(I)} \underbrace{\|w\|_{L^\infty(I)}}_{2^{-n}},$$

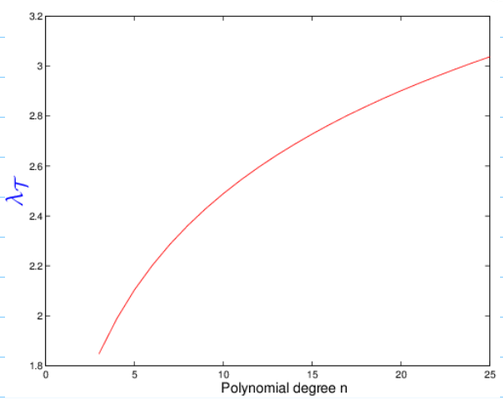
with nodal polynomial $w(t) := (t-t_0)\cdot\cdots\cdot(t-t_n)$; $= 2^{-n}T_{n+1}(t)$

Affine hf to $[a,b]$:

$$\|f - \mathsf{I}_T(f)\|_{L^\infty(I)} = \left\|\hat{f} - \mathsf{I}_{\hat{T}}(\hat{f})\right\|_{L^\infty([-1,1])} \le \frac{2^{-n}}{(n+1)!}\left\|\frac{\mathrm{d}^{n+1}\hat{f}}{\mathrm{d}\hat{t}^{n+1}}\right\|_{L^\infty([-1,1])}$$

$$\le \frac{2^{-2n-1}}{(n+1)!}|I|^{n+1}\left\|f^{(n+1)}\right\|_{L^\infty(I)}. \qquad (6.2.3.17)$$

Rem. 6.2.3.18 (Lebesgue constant for Chebychev nodes)



◁ "Measured"

$\lambda_T \le \frac{2}{\pi}\log(n+1)$

[Equid. nodes : $\lambda_T \ge Ce^{n/2}$]

$$\|f - \mathsf{L}_T f\|_{L^\infty(I)} \le (1+\lambda_T)\inf_{p\in\mathcal{P}_n}\|f-p\|_{L^\infty(I)} \quad \forall f \in C^0(I), \qquad (6.2.2.33)$$

Theorem 6.2.1.11. L^∞ polynomial best approximation estimate

If $f \in C^r([-1,1])$ (r times continuously differentiable), $r \in \mathbb{N}$, then, for any polynomial degree $n \ge r$,

$$\inf_{p\in\mathcal{P}_n}\|f-p\|_{L^\infty([-1,1])} \le (1+\pi^2/2)^r\frac{(n-r)!}{n!}\left\|f^{(r)}\right\|_{L^\infty([-1,1])},$$

where

$$\left\|f^{(r)}\right\|_{L^\infty([-1,1])} := \max_{x\in[-1,1]}|f^{(r)}(x)|.$$

▷ $$\|f - \mathsf{L}_T f\|_{L^\infty([-1,1])} \le (2/\pi\log(1+n)+2)(1+\pi^2/2)^r\frac{(n-r)!}{n!}\left\|f^{(r)}\right\|_{L^\infty([-1,1])}. \qquad (6.2.3.20)$$

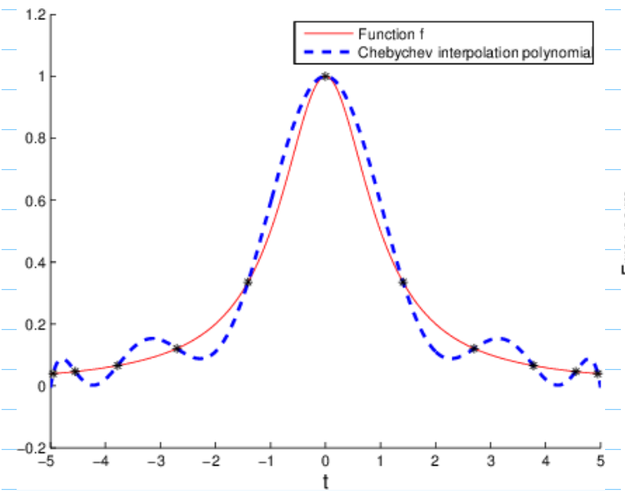
$= O(\log n / n^r)$

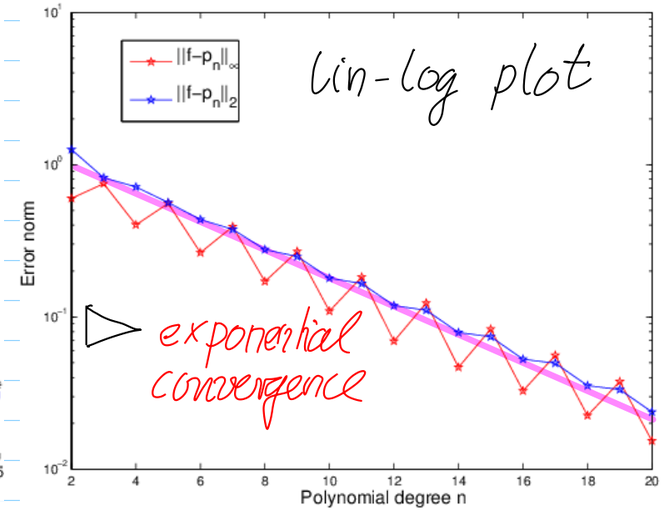
("almost algebraic convergence"

↳ If only $f \in C^1([-1,1]) \Rightarrow$ cvg. guaranteed

Exp. 6.2.3.21 (Chebychev interpolation error)

① $f(t) = \frac{1}{1+t^2}$ on $[-5,5]$

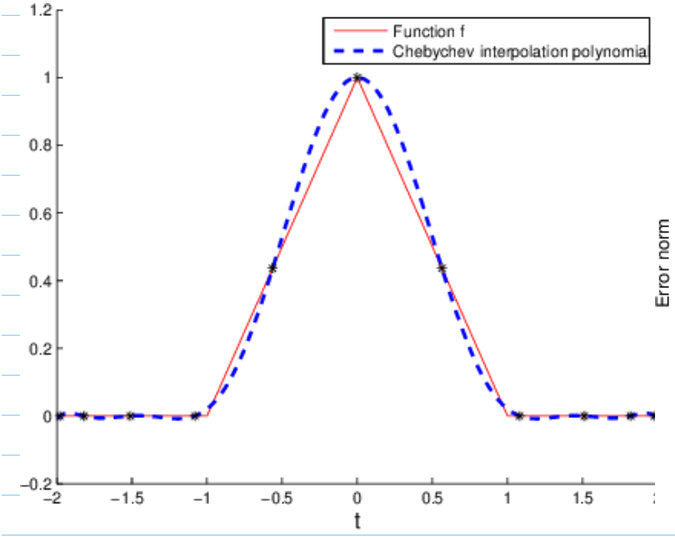


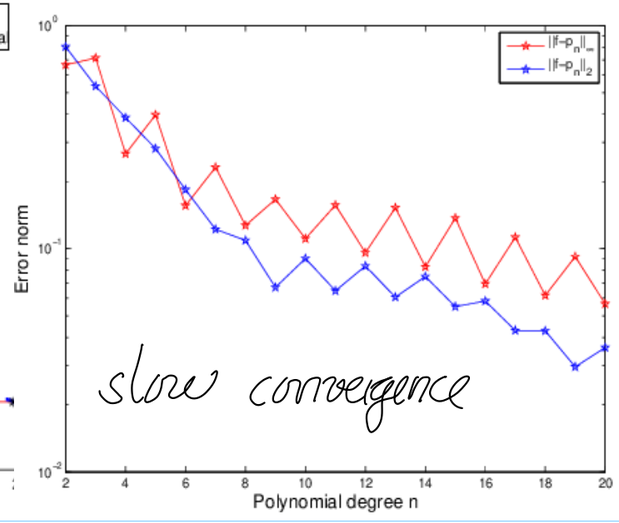


lin-log plot

▷ exponential convergence

② $f(t) = \max\{1-|t|, 0\}$, $f \in C^0$, $f \notin C^1$





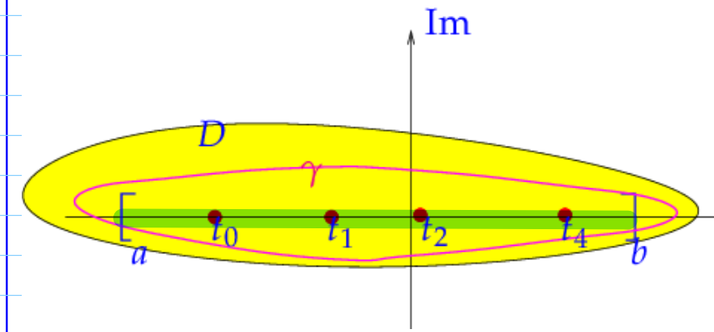
slow convergence

▷ Smoothness of f crucial

Rem. 6.2.3.24: (Chebychev interpolation of analytic functions)

$I = [-1,1]$, $w = 2^{-n}T_{n+1}$: $\leq 2^{-n}$

$$|f(t) - \mathsf{L}_T f(t)| \leq \left| \frac{w(t)}{2\pi\imath} \int_\gamma \frac{f(z)}{(z-t)w(z)} \,\mathrm{d}z \right| \leq \frac{|\gamma|}{2\pi} \frac{\max_{-1\leq\tau\leq 1}|w(\tau)|}{\min_{z\in\gamma}|w(z)|} \cdot \frac{\max_{z\in\gamma}|f(z)|}{\operatorname{dist}([a,b],\gamma)}, \qquad (6.2.2.58)$$



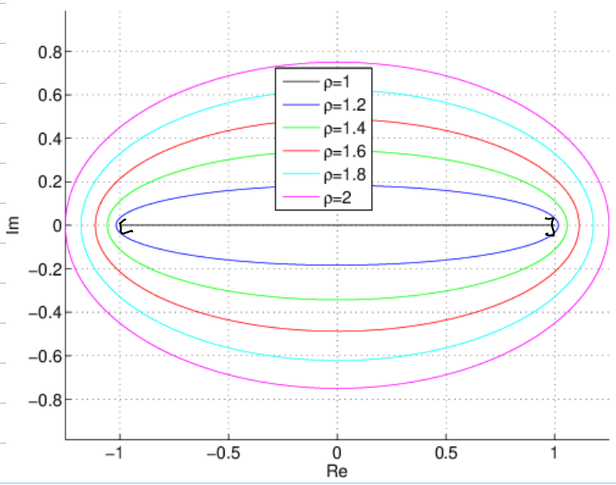
Convenience choice of γ: ($\rho > 1$)

$\gamma(\theta) = \cos(\theta - i\log\rho)$

$0 \leq \theta \leq 2\pi$

to cancel arccos

4 ellipses around $[-1,1]$



(4)

$$
\begin{aligned}
|2^n w(\gamma(\theta))|^2 &= |T_{n+1}(\cos(\theta - \imath \log \rho))|^2 \qquad \arccos \circ \cos = \mathrm{Id}\ !\\
&= |\cos((n+1)(\theta - \imath \log \rho))|^2\\
&= \cos((n+1)(\theta - \imath \log \rho)) \cdot \overline{\cos((n+1)(\theta - \imath \log \rho))}\\
&= \frac{1}{4}(\rho^s \mathrm{e}^{\imath(n+1)\theta} + \rho^{-(n+1)}\mathrm{e}^{-\imath(n+1)\theta})(\rho^{(n+1)}\mathrm{e}^{-\imath(n+1)\theta} + \rho^{-(n+1)}\mathrm{e}^{\imath(n+1)\theta})\\
&= \frac{1}{4}(\rho^{2(n+1)} + \rho^{-2(n+1)} + \mathrm{e}^{2\imath(n+1)\theta} + \mathrm{e}^{-2\imath(n+1)\theta})\\
&= \frac{1}{4}(\rho^{n+1} - \underbrace{\rho^{-(n+1)}}_{<1})^2 + \underbrace{\frac{1}{4}(\mathrm{e}^{\imath(n+1)\theta} + \mathrm{e}^{-\imath(n+1)\theta})^2}_{=\cos^2((n+1)\theta)\geq 0} \geq \frac{1}{4}(\rho^{n+1} - 1)^2 ,
\end{aligned}
$$

$$
\|f - \mathsf{L}_{\mathcal{T}} f\|_{L^\infty([-1,1])} \leq \underbrace{\frac{2|\gamma|}{\pi} \frac{1}{(\rho^{n+1}-1)(\rho + \rho^{-1} - 2)}}_{= O(1/\rho^n) \text{ as } n \to \infty} \cdot \max_{z \in \gamma} |f(z)| . \tag{6.2.3.26}
$$

$\Leftarrow$ exponential cvg.

for any f w/ analytic extension beyond $[-1,1]$

Slower for smaller domains $D \subset \mathbb{C}$ of analyticity

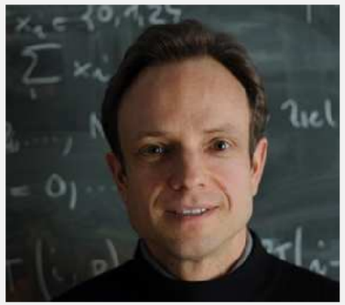
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 6.2.3.3: Chebychev Interpolation: Computational Aspects

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 13, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- C++ and EIGEN
- Arithmetic in $\mathbb{C}$

Dependencies. Section 6.2.3.1, Section 4.3

Duration: 21 minutes

VI. Approximation of Functions in 1D

6.2. Approximation by Global Polynomials

6.2.3. Chebychev Interpolation

6.2.3.3. CI: Computational Aspects

C++ code 6.2.3.31: Definition of class for Chebychev interpolation

```
1  class ChebInterp {
2    private:
3    // various internal data describing Chebychev interpolating polynomial
       p
4    public:
5      // Constructor taking function f and degree n as arguments
6      template <typename Function>
7        PolyInterp(const Function &f,unsigned int n);
8      // Evaluation operator:  y_j = p(x_j), j = 1,...,m  (m "large")
9      template <typename Vector>
10     void eval(const Vector &x, Vector &y) const;
11 };
```

$\hookrightarrow \quad L_J f = \sum_{j=0}^{n} \alpha_j T_j$ [Chebychev expansion]

(2)

§ 6.2.3.32 (Fast evaluation of Chebychev expansion)

Idea: Use 3-term recursion

$$T_j(x) = 2xT_{j-1}(x) - T_{j-2}(x), \quad j \geq 2$$

$$p(x) = \sum_{j=0}^{n-1} \alpha_j T_j(x) + \alpha_n T_n(x) \overset{(6.2.3.5)}{=} \sum_{j=0}^{n-1} \alpha_j T_j(x) + \alpha_n (2xT_{n-1}(x) - T_{n-2}(x))$$

$$= \sum_{j=0}^{n-3} \alpha_j T_j(x) + (\alpha_{n-2} - \alpha_n) T_{n-2}(x) + (\alpha_{n-1} + 2x\alpha_n) T_{n-1}(x) .$$

$\hat{=}$ Another Chebychev expansion, degree $n-1$ ▷ recursion

C++ code 6.2.3.34: Recursive evaluation of Chebychev expansion (6.2.3.3)

```
// Recursive evaluation of a polynomial p = sum_{j=1}^{n+1} a_j T_{j-1} at point x
// based on (6.2.3.33)
// IN : Vector of coefficients a
// evaluation point x
// OUT: Value at point x
double recclenshaw(const VectorXd& a, const double x) {
  const VectorXd::Index n = a.size() - 1;
  if      (n == 0) return a(0);             // Constant polynomial
  else if (n == 1) return (x*a(1) + a(0)); // Value a_1 * x + a_0
  else {
    VectorXd new_a(n);
    new_a << a.head(n - 2), a(n - 2) - a(n), a(n - 1)+ 2*x*a(n);
    return recclenshaw(new_a,x); // recursion
  }
}
```

▷ Cost = $O(n)$,

§ 6.2.3.36 (Computation of Chebychev expansion of interpolant)

I.C: $p(t_k) = f(t_k), \quad k = 0,\ldots,n,$ for $t_k := \cos\left(\frac{2k+1}{2(n+1)}\pi\right).$ (6.2.3.37)

Chebychev nodes

Idea: reduction to DFT by change of variables

$t \in [-1,1] \longmapsto s \in \mathbb{R} : t = \cos 2\pi s$: 1-periodic

$$q(s) := \underbrace{p(\cos 2\pi s)}_{t} = \sum_{j=0}^{n} \alpha_j T_j(\cos 2\pi s) \overset{\text{Def. 6.2.3.3}}{=} \sum_{j=0}^{n} \alpha_j \cos(2\pi j s)$$

$$= \sum_{j=0}^{n} \tfrac{1}{2}\alpha_j (\exp(2\pi \imath j s) + \exp(-2\pi \imath j s)) \quad [\text{by } \cos z = \tfrac{1}{2}(e^z + e^{-z})]$$

(6.2.3.38)

$$= \sum_{j=-n}^{n+1} \beta_j \exp(-2\pi \imath j s), \quad \text{with} \quad \beta_j := \begin{cases} 0 & , \text{for } j = n+1, \\ \frac{1}{2}\alpha_j & , \text{for } j = 1,\ldots,n, \\ \alpha_0 & , \text{for } j = 0, \\ \frac{1}{2}\alpha_{n-j} & , \text{for } j = -n,\ldots,-1. \end{cases}$$

↓

a trigonometric polynomial

I.C. $\Rightarrow$ $q(s_k) = f(t_k), \quad s_k = \frac{2k+1}{4(n+1)}$

$k = 0, \ldots, n$

NumCSE@ETHZ

Chebychev expansion for Chebychev interpolants

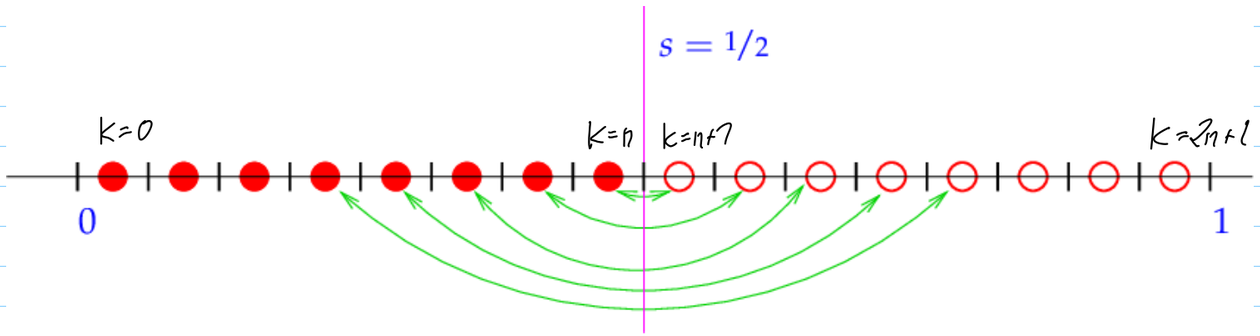
⇕

Trigonometric interpolation at equidistant points → Section 5.6.3 ▷ DFT

$q(s)$ even $\Rightarrow$ $q(s) = q(1-s)$

▷ Equivalent I.C. by reflection:

$$q\left(\frac{k}{2(n+1)}+\frac{1}{4(n+1)}\right) = z_k := \begin{cases} y_k & \text{, for } k = 0,\ldots,n\,, \\ y_{2n+1-k} & \text{, for } k = n+1,\ldots,2n+1\,. \end{cases} \tag{6.2.3.40}$$



$$q(s) = \sum_{j=-n}^{n+1} \beta_j \exp(-2\pi\imath s)\,:$$

$$q\left(\frac{k}{2(n+1)}+\frac{1}{4(n+1)}\right) = \sum_{j=-n}^{n+1} \left(\beta_j \exp\left(-\frac{2\pi\imath j}{4(n+1)}\right)\right) \exp\left(-\frac{2\pi\imath}{2(n+1)}kj\right) = z_k\,.$$

⇕ [reindexing: $j \to j+n$]

$$\sum_{j=0}^{2n+1} \left(\beta_{j-n} \exp\left(-\frac{2\pi\imath(j-n)}{4(n+1)}\right)\right) \underbrace{\exp\left(-\frac{2\pi\imath}{2(n+1)}kj\right)}_{=\omega_{2(n+1)}^{kj}!} = \exp\left(-\pi\imath\frac{nk}{n+1}\right) z_k\,, \quad k = 0,\ldots,2n+1\,.$$

⇕

$$\mathbf{F}_{2(n+1)}\mathbf{c} = \mathbf{b} \quad \text{with} \quad \mathbf{c} = \left[\beta_{j-n}\exp\left(-\frac{2\pi\imath(j-n)}{4(n+1)}\right)\right]_{j=0}^{2n+1}, \quad \mathbf{b} = \left[\exp\left(-\pi\imath\frac{nk}{n+1}\right) z_k\right]_{k=0}^{2n+1}. \tag{6.2.3.41}$$

$(2n+2)\times(2n+2)$ Fourier matrix, see (4.2.1.13)

▷ Solve w/ FFT, cost = $O(n \log n)$

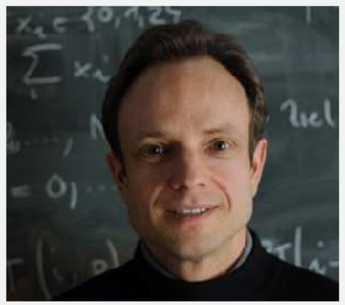
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 6.5.1: Approximation by Trigonometric Interpolation

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 15, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 5.6

Duration: 7 minutes

VI. Approximation of Functions in 1D

6.5 Approximation by Trigonometric Polynomials

$f: \mathbb{R} \to \mathbb{R}$ (~~$\mathbb{R}$~~ $\mathbb{C}$), 1-periodic, continuous : $f(t+1) = f(t)$

↳ also for approximation

↳ use trigonometric polynomial

$$\mathcal{P}_{2n}^{\mathrm{T}} = \mathrm{Span}\{1, \sin(2\pi t), \cos(2\pi t), \sin(4\pi t), \cos(4\pi t), \sin(2\pi n t), \cos(2\pi n t)\}$$

$$= \mathrm{Span}\{t \mapsto \exp(-2\pi i k t)\}_{k=-n}^{n}$$

$$\dim \mathcal{P}_{2n}^{\mathrm{T}} = 2n+1$$

↑ degree $2n$

(2)

6.5.1. Approximation by Trigonometric Interpolation

Interpolation scheme + sampling → approximation scheme

$$f : I \subset \mathbb{R} \to \mathbb{K} \xrightarrow{\text{sampling}} (t_i, y_i := f(t_i))_{i=0}^{m} \xrightarrow[\text{(Trigonometric)}]{\text{interpolation}} \widetilde{f} := \mathrm{I}_{\mathcal{T}}\mathbf{y} \quad (\widetilde{f}(t_i) = y_i) .$$

free choice of nodes $t_i \in I$

Recall: Trigonometric interpolation → Section 5.6

Given nodes $t_0 < t_1 < \cdots < t_{2n}$, $t_k \in [0,1[$, and values $y_k \in \mathbb{R}$, $k = 0, \ldots, 2n$ find

$$q \in \mathcal{P}_{2n}^{T} := \mathrm{Span}\{t \mapsto \cos(2\pi jt), t \mapsto \sin(2\pi jt)\}_{j=0}^{n} , \tag{6.5.1.2}$$

with $q(t_k) = y_k$ for all $k = 0, \ldots, 2n$. (6.5.1.3)

Terminology: $\mathcal{P}_{2n}^{T} \hat{=}$ space of trigonometric polynomials of degree $2n$.

exists / uniqueness

Note: Equidistant node ⇒ FFT-based algorithms

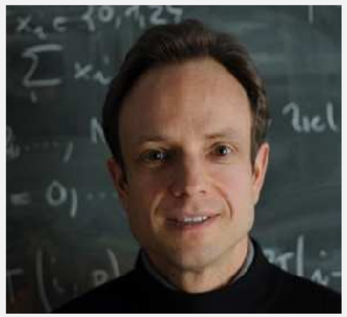
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 6.5.2: Trigonometric Interpolation Error Estimates

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 15, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.
- Fundamentals of Fourier analysis

Dependencies. Section 4.2.6, Section 6.5.1

Duration: 26 minutes

VI. Approximation of Functions in 1D

6.5 Approximation by Trigonometric Polynomials

6.5.2. Trigonometric Interpolation Error Estimates

$$T_n : C^0_{per}([0,1]) \longrightarrow \mathcal{S}^T_{2n}, \qquad n \in \mathbb{N}$$
$$(T_n f)(t_k) = f(t_k), \quad t_k := \frac{k}{2n+1}$$

Want to study $\|f - T_n f\|_*$ for $n \to \infty$, depending on smoothness of f

$$* = L^2([0,1]) \quad (L^\infty([0,1]))$$

(2)

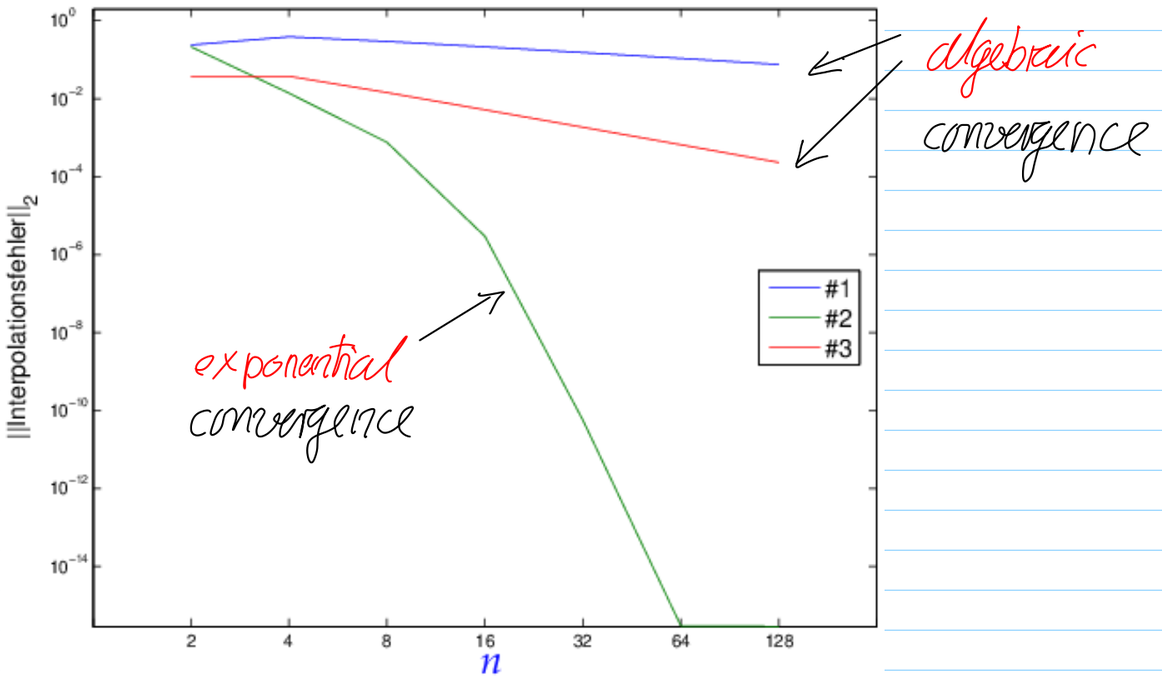
Exp. 6.5.2.1 : (Trigonometric interpolation error)

#1 Step function: $f(t) = 0$ for $|t - \frac{1}{2}| > \frac{1}{4}$, $f(t) = 1$ for $|t - \frac{1}{2}| \leq \frac{1}{4}$ $\rightarrow \notin C^0$

#2 C^∞ periodic function: $f(t) = \dfrac{1}{\sqrt{1 + \frac{1}{2}\sin(2\pi t)}}$.

#3 "wedge function": $f(t) = |t - \frac{1}{2}|$ $\rightarrow \notin C^1$

log-log plot



$L^2([0,1])$-norm of interpolation error

§ 6.5.2.3 (Fourier series → 4.2.6)

$$p \in \mathcal{P}_{2n}^T \Rightarrow p(t) = \sum_{k=-n}^{n} a_k \exp(-2\pi \imath t) \text{ for some } a_k \in \mathbb{C}. \quad (6.5.2.4)$$

Fourier series of $f \in L^2([0,1])$

$$f(t) = \sum_{k=-\infty}^{\infty} \hat{f}_k \exp(-2\pi \imath k t) \text{ in } L^2([0,1]), \quad \hat{f}_k := \int_0^1 f(t) \exp(2\pi \imath k t)\, \mathrm{d}t. \quad (6.5.2.5)$$

↑ Fourier mode μ_k ↳ Fourier coefficient

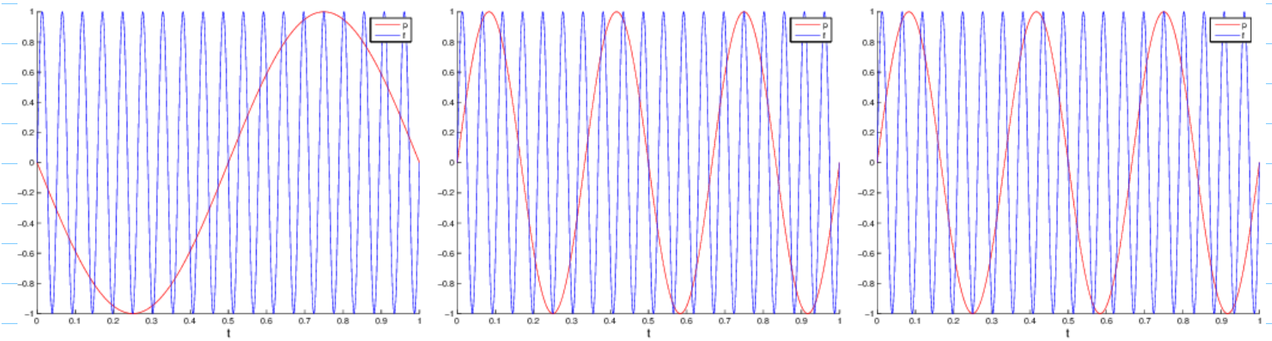
Idea: Study trigonometric interpolation error for interpolands given in Fourier series representation (6.5.2.5)

§ 6.5.2.7 (Aliasing)

$$\mathcal{J}_n := \left\{ \frac{k}{2n+1} \right\}_{k=0}^{2n}, \quad \mu_k(t) := e^{-2\pi i k t}$$

$$\mu_k|_{\mathcal{J}_n} = \mu_{k+N}|_{\mathcal{J}} \quad \forall N \in (2n+1)\mathbb{Z}$$

③



$n = 2$ $n = 4$ $n = 8$

▷ $T_n \nu_k = T \nu_{\widetilde{k}} = \nu_{\widetilde{k}}$ $\quad -n \le \widetilde{k} \le n$, $\widetilde{k} = k \bmod (2n+1)$

$$f(t) = \sum_{j=-\infty}^{\infty} \widehat{f}_j \mu_j(t) \quad \blacktriangleright \quad T_n(f)(t) = \sum_{j=-n}^{n} \gamma_j \mu_j(t) \,, \quad \gamma_j = \sum_{\ell=-\infty}^{\ell} \widehat{f}_{j+\ell(2n+1)} \,. \tag{6.5.2.10}$$

▷ Interpolation error

$E = f - T_n f = \sum_{j \in \mathbb{Z}} \widehat{E}_j \nu_j$

$$\widehat{E}_j = \begin{cases} -\sum_{\ell \in \mathbb{Z}\setminus\{0\}} \widehat{f}_{j+\ell(2n+1)} & \text{, if } j \in \{-n,\dots,n\} \,, \\ \widehat{f}_j & \text{, if } |j| > n \,, \end{cases} \qquad j \in \mathbb{Z} \,. \tag{6.5.2.11}$$

$\widehat{f}_j - \gamma_j$

Theorem 4.2.6.33. Isometry property of the Fourier transform

If the Fourier coefficients satisfy $\sum_{k\in\mathbb{Z}} |\widehat{c}_j|^2 < \infty$, *then the Fourier series*

$$c(t) = \sum_{k\in\mathbb{Z}} \widehat{c}_k \exp(-2\pi\imath k t)$$

yields a function $c \in L^2([0,1])$ *that satisfies*

$$\|c\|^2_{L^2([0,1])} := \int_0^1 |c(t)|^2 \,\mathrm{d}t = \sum_{k\in\mathbb{Z}} |\widehat{c}_j|^2 \,. \quad \longrightarrow \quad \widehat{c}_j \text{ bounded}$$

$$\|f - T_n f\|^2_{L^2(]0,1[)} = \sum_{j=-n}^{n} \left| \sum_{\ell\in\mathbb{Z}\setminus\{0\}} \widehat{f}_{j+\ell(2n+1)} \right|^2 + \sum_{|j|>n} |\widehat{f}_j|^2 \,, \tag{6.5.2.13}$$

$$\|f - T_n f\|_{L^\infty(]0,1[)} \le \sum_{j=-n}^{n} \sum_{\ell\in\mathbb{Z}\setminus\{0\}} |\widehat{f}_{j+\ell(2n+1)}| + \sum_{|j|>n} |\widehat{f}_j| \,. \tag{6.5.2.14}$$

↑ by Δ-inequality

Decay laws for $\widehat{f}_j$ ⟶ estimates

↑

Smoothness of f

(4)

$$f(t) = \sum_{j\in\mathbb{Z}} \hat{f}_j \, e^{2\pi i j t}$$

$$\frac{df}{dt}(t) = \sum_{j\in\mathbb{Z}} \underbrace{\hat{f}_j (2\pi i j)}_{= (\widehat{\frac{df}{dt}})_j} e^{2\pi i j t}$$

Lemma 6.5.2.17. Fourier coefficients of derivatives

For the Fourier coefficients of the derivatives a 1-periodic function $f \in C^{k-1}(\mathbb{R})$, $k \in \mathbb{N}$, with integrable k-th derivative $f^{(k)}$ holds

$$\widehat{(f^{(k)})}_j = (2\pi\imath j)^k \hat{f}_j \,, \quad j \in \mathbb{Z} \,.$$

↳ bounded

Lemma 6.5.2.20. Decay of Fourier coefficients

If $f \in C^{k-1}(\mathbb{R})$ with integrable k-th derivative, then $\hat{f}_j = O(|j|^{-k})$ for $|j| \to \infty$

Theorem 6.5.2.23. Finite-smoothness L^2-error estimate for trigonometric interpolation

If $f \in C^{k-1}(\mathbb{R})$, $k \in \mathbb{N}$, with square-integrable k-th derivative ($f^{(k)} \in L^2(]0,1[)$), then

$$\|f - \mathsf{T}_n f\|_{L^2(]0,1[)} \le \sqrt{1+c_k}\,(2\pi n)^{-k} \left\| f^{(k)} \right\|_{L^2(]0,1[)} \,, \tag{6.5.2.24}$$

with $c_k = 2\sum_{\ell=1}^{\infty} (2\ell-1)^{-2k} < \infty$.

Algebraic convergence, rate k

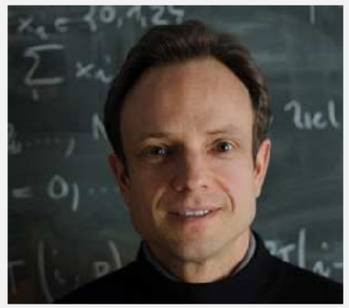
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 6.5.3: Trigonometric Interpolation of Analytic Periodic Functions

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 15, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.
- Elementary complex analysis

Dependencies. Section 6.5.1, Section 6.2.2.3

Duration: 25 minutes

VI. Approximation of Functions in 1D

6.5 Approximation by Trigonometric Polynomials

6.5.3. Trigonometric Interpolation of Analytic Periodic Functions

Definition 6.2.2.35. Real-analytic functions

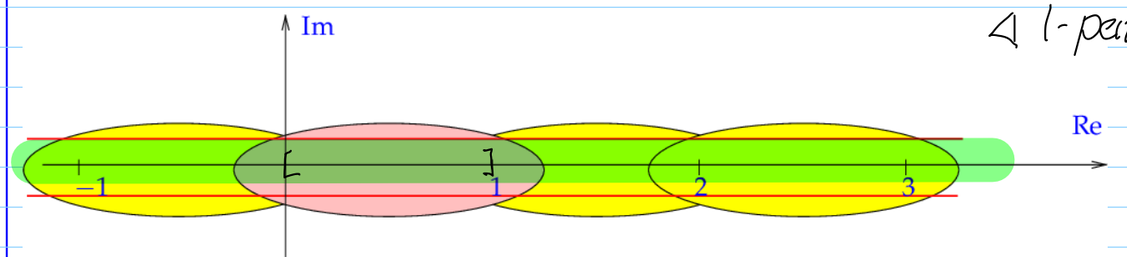
A function $f \in C^\infty(I)$ defined on an open interval $I \subset \mathbb{R}$ is called **(real-)analytic** on I, if it possesses a convergent Taylor series at every point $t_0 \in I$:

$$\forall t_0 \in I: \quad \exists \rho = \rho(t_0) > 0: \quad f(t) = \sum_{k=0}^{\infty} \frac{f^{(k)}(t_0)}{k!} (t - t_0)^k \quad \forall t \in I : \ |t - t_0| < \rho(t_0) \ .$$

$\rho(t_0)$ is called the **radius of convergence** of the Taylor series.

Naturally defined on $D \subset \mathbb{C}$

Δ 1-periodic

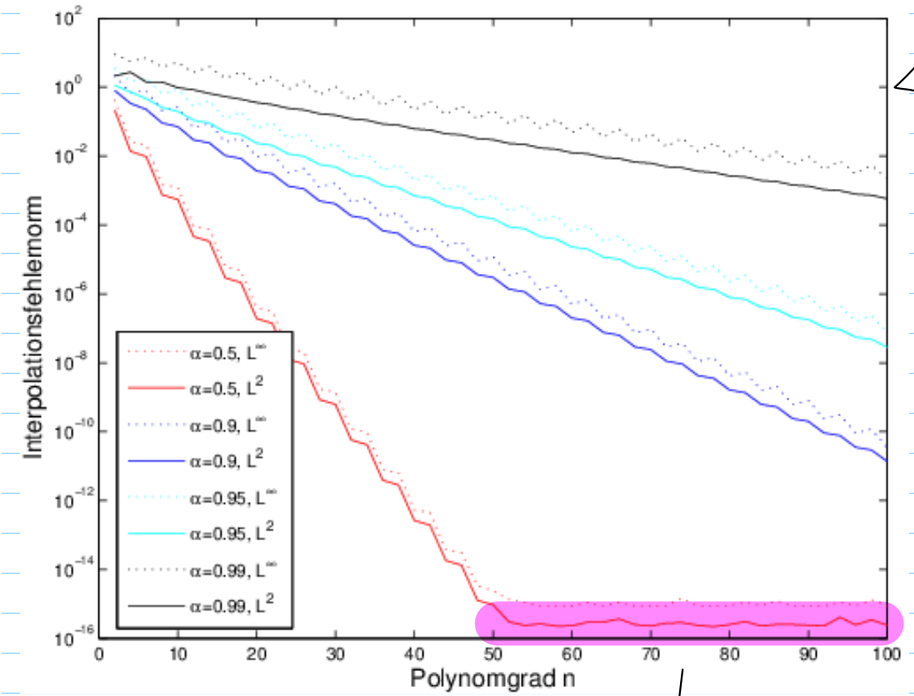


(2)

Exp. 6.5.3.1 (Trigonometric Interpolation of analytic functions)

$$f(t) = \frac{1}{\sqrt{1-\alpha\sin(2\pi t)}}\,, \quad t \in \mathbb{R}, \quad 0<\alpha<1$$

[equidistant]



△ lin-log plot

exponential convergence

⏸

?

§ 6.5.3.3 (Decay of Fourier coefficients of analytic functions)

Theorem 6.5.3.4. Exponential decay of Fourier coefficients of analytic functions

If $f : \mathbb{R} \to \mathbb{C}$ is 1-periodic and has an analytic extension to the strip

$$\bar{S} := \{z \in \mathbb{C}: -\eta \le \operatorname{Im} z \le \eta\}\,, \quad \text{for some } \eta > 0\,,$$

then its Fourier coefficients decay according to

$$|\hat{f}_j| \le q^{|k|} \cdot \|f\|_{L^\infty(\bar{S})} \quad \forall k \in \mathbb{Z} \quad \text{with} \quad q := \exp(-2\pi\eta) \in]0,1[\,. \tag{6.5.3.5}$$

Proof: $k \ge 0$

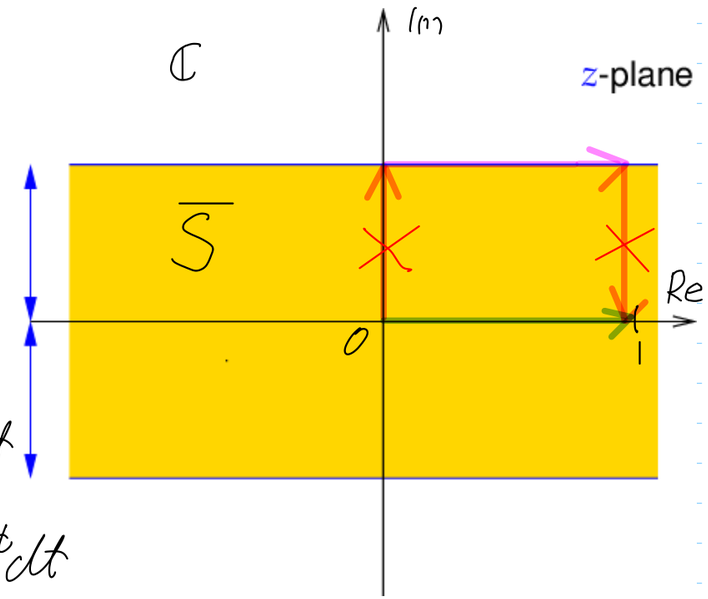
$$\hat{f}_k = \int_0^1 f(t)\, e^{2\pi i k t}\, dt$$

by Cauchy integral theorem

$$\hat{f}_k = \int_0^1 f(t+i\eta)\, e^{2\pi i k (t+i\eta)}\, dt$$

$$= e^{-2\pi k\eta} \int_0^1 f(t+i\eta)\, e^{2\pi i k t}\, dt$$

$$|\hat{f}_k| \le e^{-2\pi k\eta}\, \|f\|_{\infty,\bar{S}}$$



③ Recall:

$$\|f - \mathsf{T}_n f\|_{L^2(]0,1[)}^2 = \sum_{j=-n}^{n} \left| \sum_{\ell \in \mathbb{Z}\setminus\{0\}} \widehat{f}_{j+\ell(2n+1)} \right|^2 + \sum_{|j|>n} |\widehat{f}_j|^2 , \tag{6.5.2.13}$$

$$\|f - \mathsf{T}_n f\|_{L^\infty(]0,1[)} \le \sum_{j=-n}^{n} \sum_{\ell \in \mathbb{Z}\setminus\{0\}} |\widehat{f}_{j+\ell(2n+1)}| + \sum_{|j|>n} |\widehat{f}_j| . \tag{6.5.2.14}$$

↑ equid. trig. intp into $\mathcal{S}_{2n}^{\mathsf{T}}$

Theorem 6.5.3.13. Exponential convergence of trigonometric interpolation for analytic interpolands

If $f : \mathbb{R} \to \mathbb{C}$ is 1-periodic and possesses an analytic extension to the strip

$$\bar{S} := \{z \in \mathbb{C} : -\eta \le \operatorname{Im} z \le \eta\} , \quad \text{for some } \eta > 0 ,$$

then there is $C_\eta > 0$ depending only on η such that

$$\|f - \mathsf{T}_n f\|_* \le C_\eta e^{-\pi\eta n} \|f\|_{L^\infty(\bar{S})} , \quad n \in \mathbb{N} , \quad (* = L^2(]0,1[), L^\infty(]0,1[)) . \tag{6.5.3.14}$$

↑ exponential convergence

§ 6.5.3.15 (Exp. 6.5.3.1. explained)

$$f(t) = \frac{1}{\sqrt{1 - \alpha \sin(2\pi t)}} , \quad t \in \mathbb{R} , \quad 0 < \alpha < 1$$

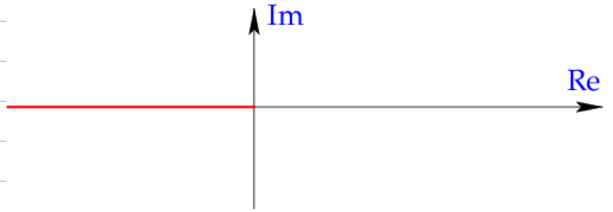
goal: Find domain of analyticity $D \subset \mathbb{C}$ of f

Easier: "Domain singularity" $P := \mathbb{C} \setminus D$

Lemma 6.5.3.17. Principal branch of the square root

The square root function $t \mapsto \sqrt{t}$, $t \ge 0$, can be extended to an analytic function on

$$\mathbb{C} \setminus \mathbb{R}_0^- := \{z \in \mathbb{C} : \operatorname{Re}(z) > 0 \quad \text{or} \quad \operatorname{Im}(z) \neq 0\} .$$



◁ domain of singularity of $z \mapsto \sqrt{z}$, principal branch

▷ $P := \{ z \in \mathbb{C} : 1 - \alpha \sin(2\pi z) \in \mathbb{R}_0^- \}$

$z = x + \imath y$:

$$1 + \alpha \sin(2\pi z) \in \mathbb{R}_0^-$$

$$\Updownarrow$$

$$\sin(2\pi z) = \sin(2\pi x)\cosh(2\pi y) + \imath \cos(2\pi x)\sinh(2\pi y) \in]-\infty, -1 - \tfrac{1}{\alpha}]$$

$$\Updownarrow$$

$$\sin(2\pi x)\cosh(2\pi y) \le -1 - \tfrac{1}{\alpha} \quad \text{and} \quad \cos(2\pi x)\sinh(2\pi y) = 0 .$$

(4)

$\Rightarrow \quad y = 0 \quad \vee \quad x \in \mathbb{Z} + \frac{1}{2}$

↳ contradicts

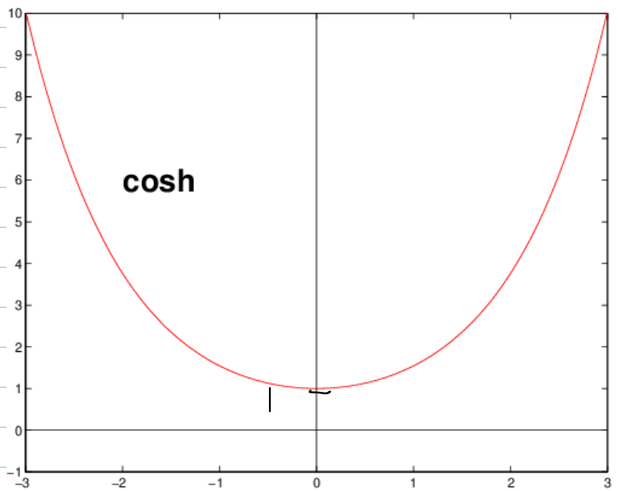
$\downarrow$

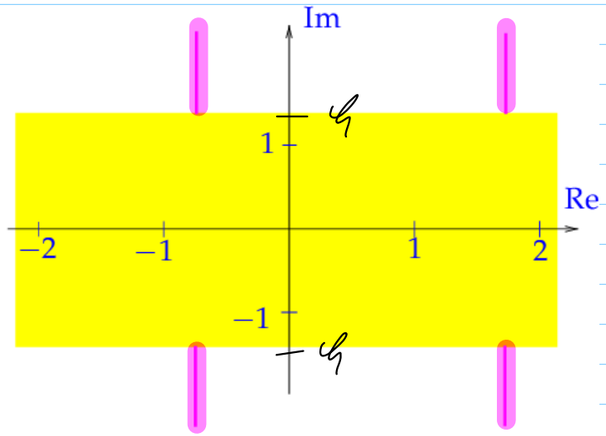
$\sin(2\pi x) \in \{1, -1\}$

$\Downarrow$

$x \in \mathbb{Z} + \frac{3}{4}$

$\Rightarrow \quad \cosh(2\pi y) \geq 1 + \frac{1}{2}$





➢ f analytic in strip

$S := \{z \in \mathbb{C} : -\zeta < \mathrm{Im}(z) < \zeta\}.$

$\cosh(2\pi\zeta) = 1 + \frac{1}{2}$

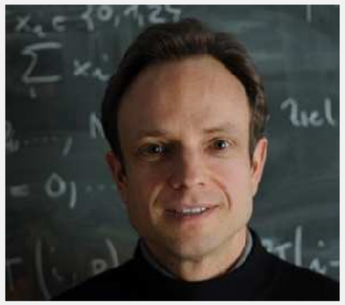
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 6.6.1: Piecewise Polynomial Lagrange Interpolation

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 18, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 5.2.2, Section 5.3.2, Section 6.2.2.2,

Duration: 30 minutes

VI Approximation of Functions in 1D

6.6. Approximation by Piecewise Polynomials

↓

2

Interpolation scheme + sampling → approximation scheme

$$f : I \subset \mathbb{R} \to \mathbb{K} \xrightarrow{\text{sampling}} (t_i, y_i := f(t_i))_{i=0}^{m} \xrightarrow{\text{interpolation}} \tilde{f} := \mathrm{I}_{\mathcal{T}}\mathbf{y} \quad (\tilde{f}(t_i) = y_i)\,.$$

free choice of nodes $t_i \in I$

Now: p.w. polynomial

(Ch. 5: p.w. linear, cubic)

Benefits:
- Locality
- Cheap algorithm
- shape preservation

(2)

Idea: use piecewise polynomials with respect to a grid/mesh

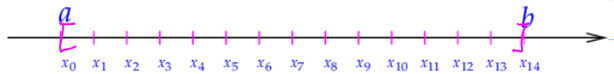
↳ partition of $I := [a,b]$: free to chose

$$\mathcal{M} := \{a = x_0 < x_1 < \ldots < x_{m-1} < x_m = b\} \qquad (6.6.0.2)$$

to approximate function $f : [a,b] \mapsto \mathbb{R}$, $a < b$.

Terminology:

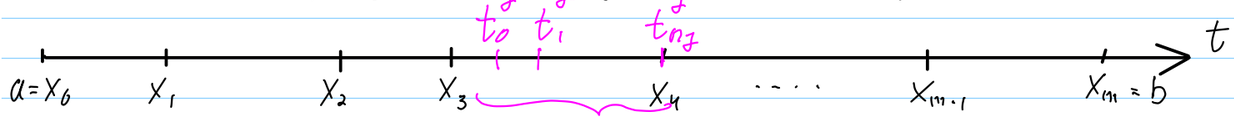
- $x_j \hat{=}$ nodes of the mesh $\mathcal{M}$,
- $[x_{j-1}, x_j[\hat{=}$ intervals/cells of the mesh,
- $h_{\mathcal{M}} := \max_j |x_j - x_{j-1}| \hat{=}$ mesh width,
- If $x_j = a + jh \hat{=}$ equidistant (uniform) mesh with meshwidth $h > 0$



6.6.1 Piecewise Polynomial Lagrange Interpolation

Idea: Local Lagrange polynomial interpolation on cells

Fix polynomial degree n_j for j-th cell $[x_{j-1}, x_j]$



$n_j + 1$ interpolation nodes $\in [x_{j-1}, x_j]$

General local Lagrange interpolation on a mesh (PPLIP)

❶ Choose local degree $n_j \in \mathbb{N}_0$ for each cell of the mesh, $j = 1, \ldots, m$.

❷ Choose set of *local* interpolation nodes

$$\mathcal{T}^j := \{t_0^j, \ldots, t_{n_j}^j\} \subset I_j := [x_{j-1}, x_j] \,, \quad j = 1, \ldots, m \,,$$

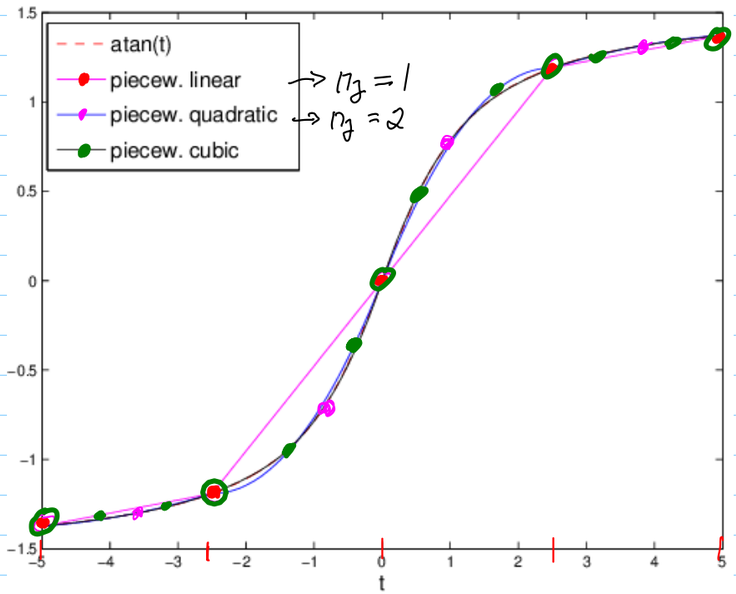
for each mesh cell/grid interval I_j.

❸ Define piecewise polynomial (PP) interpolant $s : [x_0, x_m] \to \mathbb{K}$:

$$s_j := s_{|I_j} \in \mathcal{P}_{n_j} \quad \text{and} \quad s_j(t_i^j) = f(t_i^j) \quad i = 0, \ldots, n_j \,, \quad j = 1, \ldots, m \,. \qquad (6.6.1.2)$$

Owing to Thm. 5.2.2.7, s_j is well defined.

I.C. ▷ Linear interpolation operator $I_{\mathcal{M}}$



$\mathcal{M} = \{-5, -\frac{5}{2}, 0, \frac{5}{2}, 5\}$

p.w. quadratic:

$t_0^j = x_{j-1}$

$t_1^j = \frac{1}{2}(x_j + x_{j-1})$

$t_2^j = x_j$

③

PPLIP - interpolant $\in C^0([a,b])$?

Yes $\Leftrightarrow$ $t_{n_j}^j = x_j = t_0^{j+1}$, $j = 1,\dots,m-1$

C^1-continuity ?

$\longrightarrow$ not achievable by PPLIP

Focus: Asymptotics of interpolation error

Fix $n_j = n$ & same interpolation scheme $\forall$ cells

Consider sequences of meshes with $h_m \longrightarrow 0$

$\hat{=}$ h-convergence

$$\|f - I_m f\|_\infty \leq C\, T(h_m)$$

$f \in C^{n+1}(I)$:

Thm. 6.2.2.15 $\Rightarrow$

$$\|f - \mathrm{L}_\tau f\|_{L^\infty(I)} \leq \frac{\|f^{(n+1)}\|_{L^\infty(I)}}{(n+1)!} \underbrace{\max_{t\in I} |(t-t_0)\cdot\cdots\cdot(t-t_n)|}_{\leq\, h_j^{n+1},\ h_j := x_j - x_{j-1}} . \qquad (6.2.2.22)$$

Apply to $I = [x_{j-1}, x_j]$ with $n = n_j$

Take max over all cells

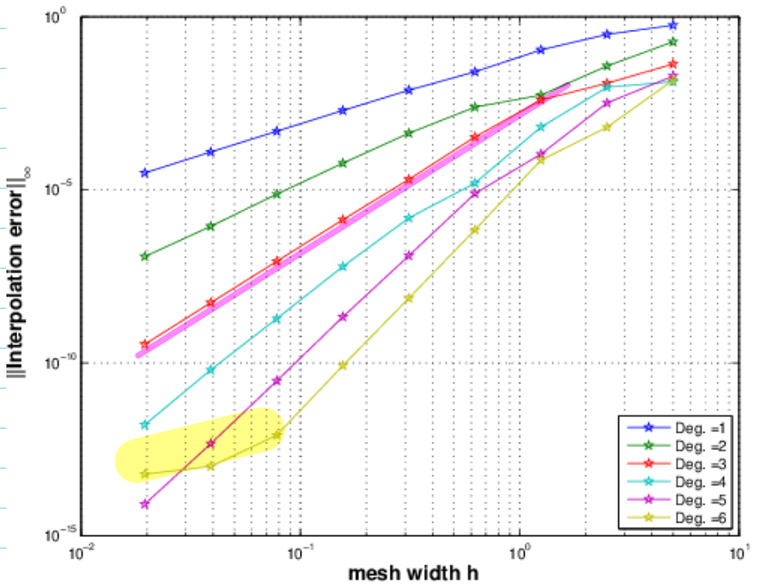
$$\|f - I_m f\|_{\infty,[a,b]} \leq \frac{1}{(n+1)} \|f^{(n+1)}\|_{[a,b],\infty} h_m^{n+1}$$
$$= O(h_m^{n+1})$$

$\hat{=}$ algebraic conv. with rate $n+1$

Ex. 6.6.1.7. (h-convergence of PPLIP)

$f(t) = \arctan t$, $I = [-5,5]$, equid. mesh.

[Equidistant local interpolation nodes]



△ log-plot

[error points on lines]

rates $\approx n+1$

ETH Lecture 401-0663-00L Numerical Methods for CSE

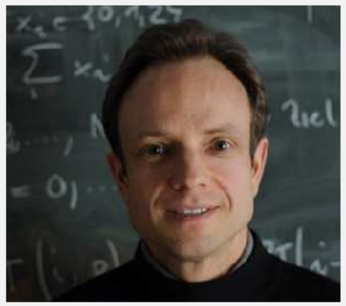
Course Video

Section 6.6.2: Cubic Hermite and Spline Interpolation: Error Estimates

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 19, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 5.3.3, Section 5.4.2

Duration: minutes

VI Approximation of Functions in 1D

6.6. Approximation by Piecewise Polynomials

Interpolation scheme + sampling → approximation scheme

[generalized]

$$f : I \subset \mathbb{R} \to \mathbb{K} \xrightarrow{\text{sampling}} (t_i, y_i := f(t_i))_{i=0}^{m} \xrightarrow{\text{interpolation}} \widetilde{f} := \mathrm{I}_{\mathcal{T}}\mathbf{y} \quad (\widetilde{f}(t_i) = y_i)\,.$$

free choice of nodes $t_i \in I$

[cubic Hermite intp. Sect. 5.5.3]

[cubic spline intp. 5.4.2]

(2)

6.6.2. Cubic Hermite Interpolation: Error Estimates

Definition 6.6.2.1. Piecewise cubic Hermite interpolant (with exact slopes) → Def. 5.3.3.1

Given $f \in C^1([a,b])$ and a mesh $\mathcal{M} := \{a = x_0 < x_1 < \ldots < x_{m-1} < x_m = b\}$ the piecewise cubic Hermite interpolant (with exact slopes) $s : [a,b] \to \mathbb{R}$ is defined as

$$s_{|[x_{j-1},x_j]} \in \mathcal{P}_3\,, \quad j = 1,\ldots,m \quad , \quad s(x_j) = f(x_j)\,, \; s'(x_j) = f'(x_j)\,, \quad j = 0,\ldots,m\,.$$

↓

Generalized interpolation (also based on derivative values)

→ $s \in C^1([a,b])$

Exp. 6.6.2.2: (Convergence of Hermite interpolation with exact slopes)

h-convergence:

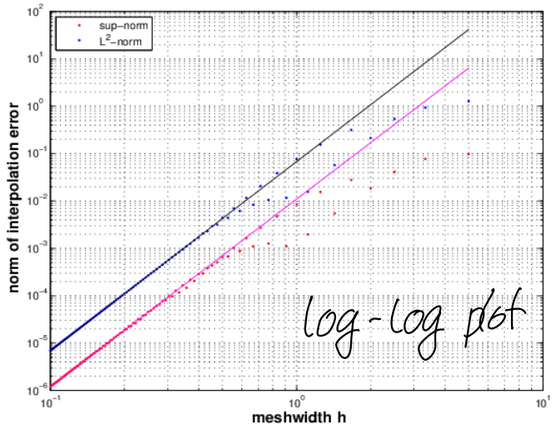
Piecewise cubic Hermite interpolation of

$$f(x) = \arctan(x)\,.$$

- domain: $I = (-5,5)$
- mesh $\mathcal{T} = \{-5 + hj\}_{j=0}^{n} \subset I$, $h = \frac{10}{n}$,
- exact slopes $c_i = f'(t_i)$, $i = 0,\ldots,n$

algebraic convergence $O(h^4)$

→ equidistant mesh



Theorem 6.6.2.3. Convergence of approximation by cubic Hermite interpolation

Let s be the cubic Hermite interpolant of $f \in C^4([a,b])$ on a mesh $\mathcal{M} := \{a = x_0 < x_1 < \ldots < x_{m-1} < x_m = b\}$ according to Def. 6.6.2.1. Then

$$\|f - s\|_{L^\infty([a,b])} \leq \frac{1}{4!} h_{\mathcal{M}}^4 \left\| f^{(4)} \right\|_{L^\infty([a,b])}\,,$$

with the meshwidth $h_{\mathcal{M}} := \max_j |x_j - x_{j-1}|$.

6.6.3. Cubic Spline Interpolation: Error Estimates

- Here: mesh = knot set
- Natural / periodic CSI not meaningful for generic $f : [a,b] \to \mathbb{R}$

Definition 6.6.3.1. Complete cubic spline interpolant

Given $f \in C^1([a,b])$ and a mesh (= knot set) $\mathcal{M} := \{a = x_0 < x_1 < \ldots < x_{m-1} < x_m = b\}$ the **complete cubic spline interpolant** s is defined by the conditions

$$s \in \mathcal{S}_{3,\mathcal{M}}\,, \; s(x_j) = f(x_j)\,, \quad j = 0,\ldots,m\,, \; s'(a) = f'(a)\,, \; s'(b) = f'(b)\,.$$

→ valid definition

↓

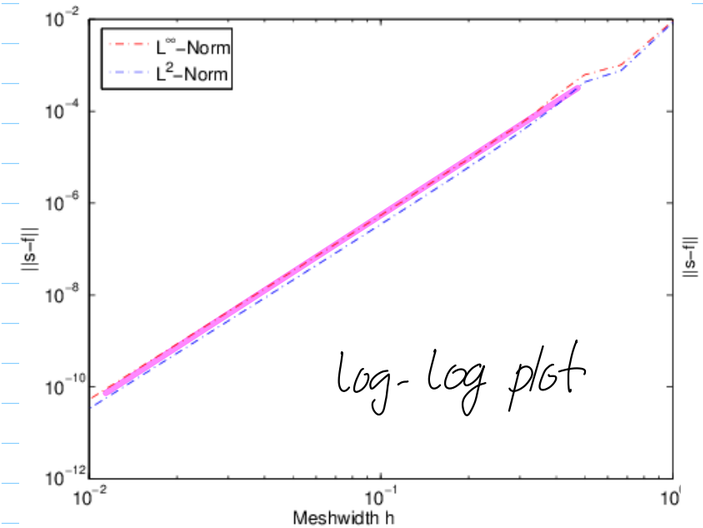
generalized CSI

③

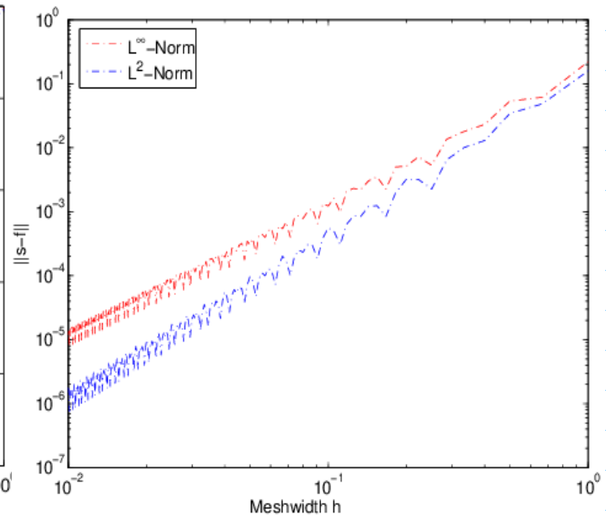
Exp.: 6.6.3.1 (Approximation by complete cubic spline interpolants)

$I = [-1,1]$, equidistant mesh

$$f_1(t) = \frac{1}{1+e^{-2t}} \in C^\infty(I), \quad f_2(t) = \begin{cases} 0 & \text{, if } t < -\frac{2}{5}, \\ \frac{1}{2}(1+\cos(\pi(t-\frac{3}{5}))) & \text{, if } -\frac{2}{5} < t < \frac{3}{5}, \\ 1 & \text{otherwise.} \end{cases} \in C^1(I).$$



log-log plot



$\|f_1 - s\|_{L^\infty([-1,1])} = O(h^4)$

$\|f_2 - s\|_{L^\infty([-1,1])} = O(h^2)$

→ low smoothness limits convergence

Theory:

$$f \in C^4([t_0,t_n]) \qquad \|f-s\|_{L^\infty([t_0,t_n])} \le \frac{5}{384} h^4 \left\| f^{(4)} \right\|_{L^\infty([t_0,t_n])}. \tag{6.6.3.3}$$

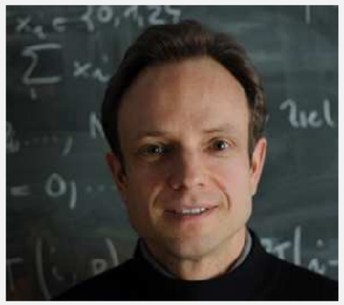
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 7.1: Numerical Quadrature: Introduction

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 21, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



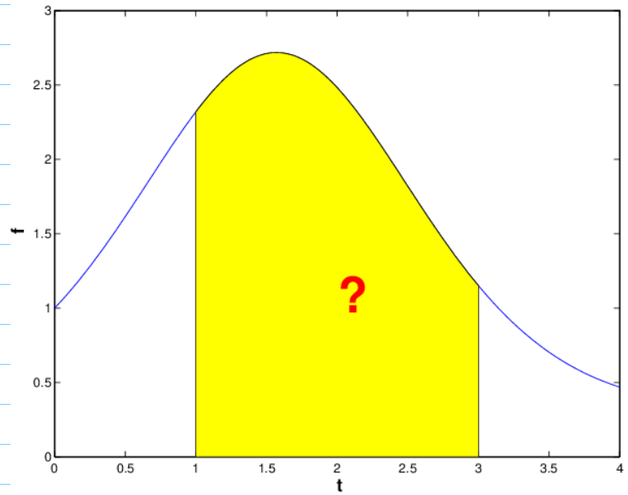
Prerequisites.

- Knowledge of C++
- Calculus: interpretation of finite integrals
- Power formula of circuit theory

Duration: 6 minutes

VIII Numerical Quadrature

7.1. Introduction



Task:

Compute $\int_a^b f(t)\,dt$

$\Vert$

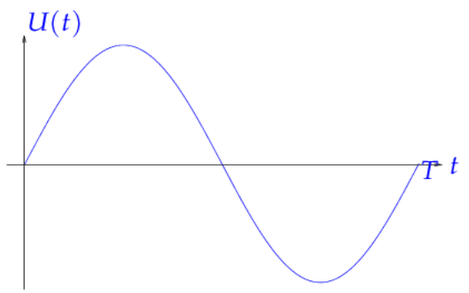
Approximate

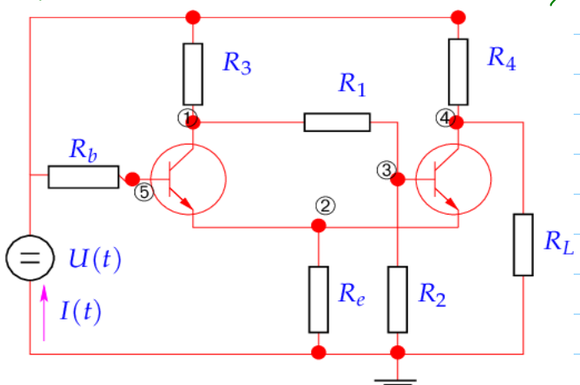
given in procedural form

(only point evaluations possible)

Ex 7.1.0.4 (Heat production in electrical circuits)

assume time-harmonic periodic excitation with period $T > 0$.





(2)

$$W_{therm} = \int_0^T U(t)\, I(U(t))\, dt$$

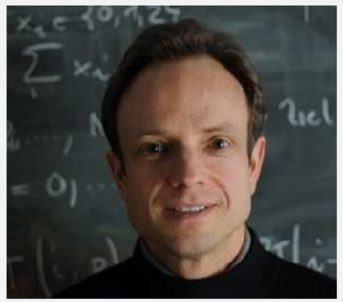
not explicit: result of an expensive comp. for every $U(t)$

▷ Integrand: `double f(double t)`

ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 7.2: Quadrature Formulas/Rules

Prof. R. Hiptmair, SAM, ETH Zurich



Date: November 21, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich

Prerequisites.

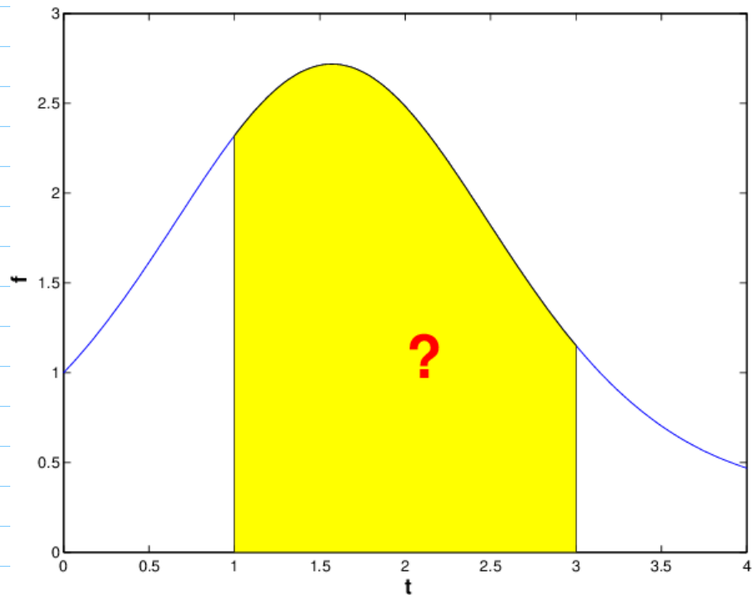
- Substitution/transformation formula for integrals

Dependencies. Section 6.1

Duration: 25 minutes

VIII Numerical Quadrature

7.2. Quadrature Formulas/Rules



Task:

Approximate

$$I := \int_a^b f(t)\,dt$$

f given in procedural form

double f(double)

②

Note : $\begin{cases} C^0([a,b]) \longrightarrow \mathbb{R} \\ f \longmapsto I \end{cases}$ linear

Definition 7.2.0.1. Quadrature formula/quadrature rule

An n-point quadrature formula/quadrature rule (QR) on $[a,b]$ provides an approximation of the value of an integral through a *weighted sum* of point values of the integrand:

$$\int_a^b f(t)\,\mathrm{d}t \approx Q_n(f) := \sum_{j=1}^{n} w_j^n f(c_j^n)\,. \tag{7.2.0.2}$$

weights $\in \mathbb{R}$ nodes/points $\in [a,b]$

$\hat{=}$ n-pt. Q.R. : cost(Q.R.) $\sim n$

Rem. 7.2.0.4 : (Transformation of quadrature rules)

$\longleftrightarrow$ transformation* of approximation schemes *

Given: Q.R. $(\hat{c}_j, \hat{w}_j)_{j=1}^n$ on reference interval $\hat{I} := [-1,1]$

Sought: Q.R. on $[a,b]$

$$\phi(\hat{t}) = a + \tfrac{1}{2}(\hat{t}+1)(b-a)\,,\ \hat{t} \in \hat{I}$$

* $\phi : [-1,1] \longrightarrow [a,b]$

$$\int_a^b f(t)\,dt = \int_{-1}^{1} f(\phi(\hat{t}))\left|\frac{d\phi}{d\hat{t}}(\hat{t})\right| d\hat{t}$$

$\hookrightarrow = \tfrac{1}{2}(b-a)$

$$\approx \tfrac{1}{2}(b-a)\sum_{j=1}^{n} \hat{w}_j f(\phi(\hat{c}_j))$$

quadrature formula for general interval $[a,b]$, $a,b \in \mathbb{R}$:

$$\int_a^b f(t)\,\mathrm{d}t \approx \tfrac{1}{2}(b-a)\sum_{j=1}^{n} \hat{w}_j \hat{f}(\hat{c}_j) = \sum_{j=1}^{n} w_j f(c_j) \quad \text{with} \quad \begin{aligned} c_j &= \tfrac{1}{2}(1-\hat{c}_j)a + \tfrac{1}{2}(1+\hat{c}_j)b\,, \\ w_j &= \tfrac{1}{2}(b-a)\hat{w}_j\,. \end{aligned}$$

Rem 7.2.0.6 (Tables of quadrature rules)

In codes : Store $(\hat{c}_j, \hat{w}_j)_{j=1}^n$ for reference interval for many different n.

NumCSE@ETHZ

§ 7.2.0.7 (Quadrature rules by interpolation schemes)

$\mathcal{J} := \{t_1, \dots, t_n\} \hat{=}$ interp. node, $t_j \in I$

I.S. $\quad I_{\mathcal{J}} : \mathbb{R}^n \longrightarrow C^0(I)$

I.C. $\quad (I_{\mathcal{J}} y)(t_j) = (y)_j \,, \; j = 1, \dots, n$

Idea: $\int_a^b f(t)\,dt \approx \int_a^b I_{\mathcal{J}}([f(t_j)]_{j=1}^n)(\tau)\,d\tau$

(j-th coordinate vector)

Assume: $I_{\mathcal{J}}$ is linear

$$= \sum_{j=1}^n f(t_j) \underbrace{\int_a^b I_{\mathcal{J}}(e_j)(\tau)\,d\tau}_{\text{weight } w_j}$$

quad. nodes; cardinal function (e.g. Lagrange polynomials)

interpolation schemes	→	approximation schemes	→	quadrature schemes
[Ch. 5]		[Ch. 6]		[Ch. 7]

§ 7.2.0.12 (Convergence of numerical quadrature)

Focus: families of Q.R., n-dependence of quadrature error $E_n(f) = \left| \int_a^b f(t)\,dt - Q_n(f) \right|$

asymptotics for $n \longrightarrow \infty$

§ 7.2.0.13 (Quadrature error from interpolation error)

Q.R. from interpolation scheme $I_{\mathcal{J}}$

$$E_n(f) = \left| \int_a^b f(t)\,dt - \int_a^b I_{\mathcal{J}}([f(t_j)]_{j=1}^n](\tau)\,d\tau \right|$$

$$\leq |b-a| \, \| f - I_{\mathcal{J}}([f(t_j)]_{j=1}^n \|_{\infty,[a,b]}$$

($\max$-norm of interpolation error)

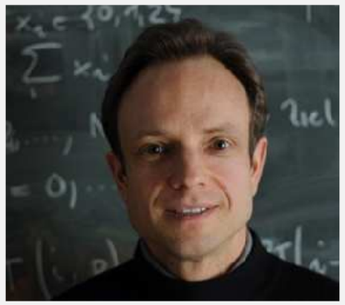
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 7.3: Polynomial Quadrature Formulas

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 21, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 5.2

Duration: 16 minutes

VIII Numerical Quadrature

7.3 Polynomial Quadrature Formulas

Recall: Q.R. by interpolation

$$\int_a^b f(t)\,dt \approx$$

$$\int_a^b I_{\mathcal{T}}[f(t_0),\ldots,f(t_n)]^\top \,\mathrm{d}t = \sum_{j=0}^{n} f(t_j) \underbrace{\int_a^b (I_{\mathcal{T}}(\mathbf{e}_j))(t)\,\mathrm{d}t}_{\text{weight } w_j}. \qquad (7.2.0.11)$$

↑ interpolation scheme based on node set $\mathcal{T} = \{t_1,\ldots,t_n\}$

Now: $I_{\mathcal{T}} \hat{=}$ Lagrange pol. interp.

$$\int_a^b f(t)\,dt = \sum_{j=1}^{n} f(t_j) \int_a^b L_j(t)\,dt$$

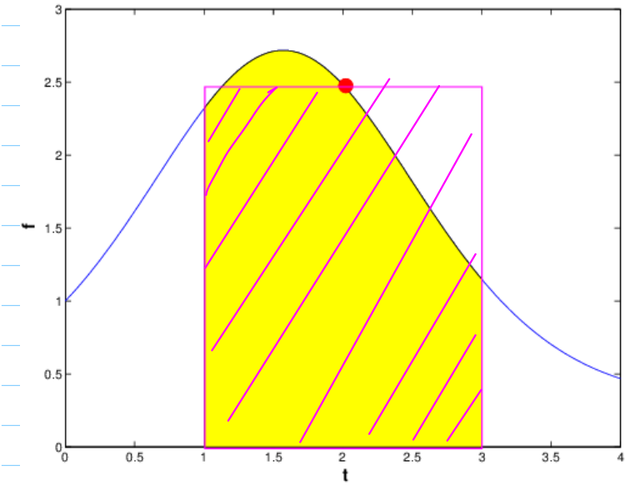
$= w_j$ (weight)

Lagrange polynomials

(2)

Ex. 7.3.0.3 (Midpoint rule)

$n = 1$ (degree 0), $t_j = \frac{1}{2}(b+a)$



$$\int_a^b f(t)\,\mathrm{d}t \approx Q_{\mathrm{mp}}(f) = \overbrace{(b-a)}^{w_1} f(\tfrac{1}{2}(a+b)) .$$

"midpoint"

Ex. (Trapezoidal rule)

$n = 2$ (degree 1), $t_1 = a$, $t_2 = b$

On $I = [0,1]$: $L_1(t) = 1 - t$

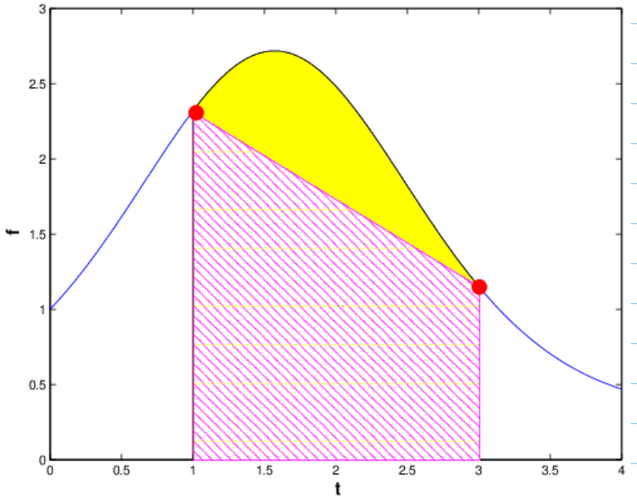
$L_2(t) = t$

$\triangleright$ $w_1 = w_2 = \frac{1}{2}$

```
> trapez := newtoncotes(1);
```

$$\widehat{Q}_{\mathrm{trp}}(f) := \frac{1}{2}(f(0)+f(1)) \qquad (7.3.0.5)$$

$$\left(\int_a^b f(t)\,\mathrm{d}t \approx \frac{b-a}{2}(f(a)+f(b)) \right)$$



Ex 7.3.0.4 (Newton-Cotes formulas)

Equidistant quadrature nodes in $[a,b]$

$$t_j = a + j\frac{b-a}{n-1}, \quad j = 0,\dots,n-1$$

$\triangleright$ Dangerous for $n \gg 1$: huge weights, $|w_j| \gg 1$ with alternating sign: $\Rightarrow$ cancellation

NumCSE@ETHZ

Rem. 7.3.0.7 (Clenshaw-Curtis quadrature rules)

$\hat{=}$ polynomial quadrature w/ Chebyshev nodes

▷ weights $w_j > 0$

§ 7.3.0.9: Error estimates for polynomial quadrature

$f \in C^{n+1}(I)$

$$\|f - L_T f\|_{L^\infty(I)} \leq \frac{\left\|f^{(n+1)}\right\|_{L^\infty(I)}}{(n+1)!} \underbrace{\max_{t\in I} |(t-t_0)\cdot\cdots\cdot(t-t_n)|}_{\downarrow} . \qquad (6.2.2.22)$$

$$f \in C^n([a,b]) \;\Rightarrow\; \left| \int_a^b f(t)\,\mathrm{d}t - Q_n(f) \right| \leq \frac{1}{n!}(b-a)^{n+1} \left\|f^{(n)}\right\|_{L^\infty([a,b])} . \qquad (7.3.0.10)$$

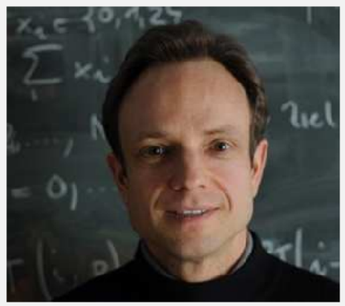
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 7.4.1: Order of a Quadrature Rule

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 21, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich

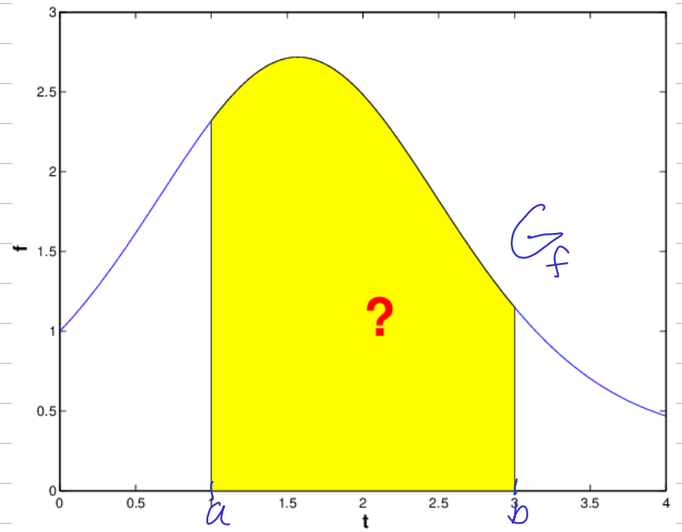


Dependencies. Section 5.2.1, Section 7.2

Duration: 15 minutes

VIII Numerical Quadrature

7.4 Gauss Quadrature



Approximate $\int_a^b f(t)\,dt$

Definition 7.2.0.1. Quadrature formula/quadrature rule

An n-point quadrature formula/quadrature rule (QR) on $[a,b]$ provides an approximation of the value of an integral through a *weighted sum* of point values of the integrand:

$$\int_a^b f(t)\,\mathrm{d}t \approx Q_n(f) := \sum_{j=1}^{n} w_j^n f(c_j^n)\,. \tag{7.2.0.2}$$

(2)

7.4.1 Order of a Quadrature Rule

Gauge of quality of Q.R.:

Definition 7.4.1.1. Order of a quadrature rule ⟷ polynomial exactness

The order of quadrature rule $Q_n : C^0([a,b]) \to \mathbb{R}$ is defined as

$$\text{order}(Q_n) := \max\{m \in \mathbb{N}_0 : \quad Q_n(p) = \int_a^b p(t)\,\mathrm{d}t \quad \forall p \in \mathcal{P}_m\} + 1\,, \tag{7.4.1.2}$$

that is, as the maximal degree $+1$ of polynomials for which the quadrature rule is guaranteed to be exact.

II Order of polynomial n-pt. Q.R. $= n$

- Order is stable under affine bij. of Q.R.

Corollary 7.4.1.4. Invariance of order under affine transformation

An affine transformation of a quadrature rule according to Rem. 7.2.0.4 does not change its order.

Reason: $\Phi : [-1,1] \to [a,b]$ affine:

$p \in \mathcal{P}_{n-1} \Rightarrow t \mapsto p(\Phi(t)) \in \mathcal{P}_{n-1}$

For any node set $\{c_1, \dots, c_n\}$ ∃ weights $\{w_j\}_{j=1}^n$ that yield order-n Q.R.

$$\forall p \in \mathcal{P}_{n-1} : \quad p = \sum_{j=1}^n p(t_j) L_j$$

(L_j: Lagrange pol.)

$$\int_a^b p(t)\,dt = \sum_{j=1}^n p(t_j) \underbrace{\int_a^b L_j(t)\,dt}_{= \text{weight } w_j}$$

Theorem 7.4.1.6. Sufficient order conditions for quadrature rules

An n-point quadrature rule on $[a,b]$ (→ Def. 7.2.0.1)

$$Q_n(f) := \sum_{j=1}^n w_j f(c_j)\,, \quad f \in C^0([a,b])\,,$$

with nodes $t_j \in [a,b]$ and weights $w_j \in \mathbb{R}$, $j = 1,\dots,n$, has order $\geq n$, if and only if

$$w_j = \int_a^b L_{j-1}(t)\,dt\,, \quad j = 1,\dots,n\,,$$

where L_k, $k = 0,\dots,n-1$, is the k-th Lagrange polynomial (5.2.2.4) associated with the ordered node set $\{c_1, c_2, \dots, c_n\}$.

NumCSE@ETHZ

Rem 7.4.1.7 : (Linear system for quadrature weights)

$\{p_1, \dots, p_n\} \hat{=}$ basis of $\mathcal{P}_{n-1}$

Aim: $Q_n(p) = \int_a^b p(t)\,dt \quad \forall p \in \mathcal{P}_{n-1}$

$\Updownarrow$

$$Q_n(p_j) = \int_a^b p_j(t)\,\mathrm{d}t \quad \forall j = 1,\dots,n \quad \Leftrightarrow \quad Q_n \text{ has order } \geq n\,. \tag{7.4.1.8}$$

$\Updownarrow \quad Q_n(f) = \sum_{j=1}^{n} w_j f(t_j)$

nyn LSE:

$$\begin{bmatrix} p_1(c_1) & \dots & p_1(c_n) \\ \vdots & & \vdots \\ p_n(c_1) & \dots & p_n(c_n) \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} \int_a^b p_1(t)\,\mathrm{d}t \\ \vdots \\ \int_a^b p_n(t)\,\mathrm{d}t \end{bmatrix}. \tag{7.4.1.9}$$

↑ weights

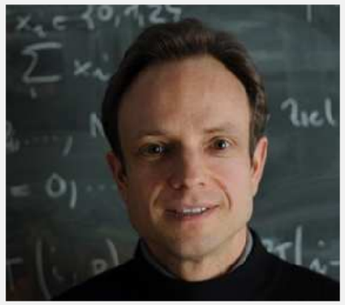
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 7.4.2: Maximal-Order Quadrature Rules

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 21, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.
- Orthogonality and Gram-Schmidt orthogonalization

Dependencies. Section 7.2, Section 7.4.1, Section 6.3.1.3

Duration: 30 minutes

VII Numerical Quadrature
7.4 Gauss Quadrature

Definition 7.2.0.1. Quadrature formula/quadrature rule

An n-point quadrature formula/quadrature rule (QR) on $[a,b]$ provides an approximation of the value of an integral through a *weighted sum* of point values of the integrand:

$$\int_a^b f(t)\,\mathrm{d}t \approx Q_n(f) := \sum_{j=1}^{n} w_j^n f(c_j^n)\,. \tag{7.2.0.2}$$

Definition 7.4.1.1. Order of a quadrature rule

The order of quadrature rule $Q_n : C^0([a,b]) \to \mathbb{R}$ is defined as

$$\mathrm{order}(Q_n) := \max\{m \in \mathbb{N}_0:\ Q_n(p) = \int_a^b p(t)\,\mathrm{d}t \quad \forall p \in \mathcal{P}_m\} + 1\,, \tag{7.4.1.2}$$

that is, as the maximal degree $+1$ of polynomials for which the quadrature rule is guaranteed to be exact.

$\forall$ n-pt Q.R. order $\geq n$ possible

7.4.2 Maximal-Order Quadrature Rules

Theorem 7.4.2.1. Maximal order of n-point quadrature rule

The maximal order of an n-point quadrature rule is $2n$.

$\Leftrightarrow$ $\nexists$ n-pt Q.R. of order $2n+1$

Proof (indirect): Q.R. w/ nodes $\{c_1, \dots, c_n\}$

Assume: $Q_n(p) = \int_a^b p(t)\,dt \quad \forall p \in \mathcal{P}_{2n}$

[order $= 2n+1$]

$$q(t) := \prod_{j=1}^{n} (t - c_j)^2 \in \mathcal{P}_{2n}, \quad q \geq 0$$

$$\int_a^b q(t)\,dt \neq Q_n(q) = 0$$

> 0 because $q(c_j) = 0$!

□

NumCSE@ETHZ

n-pt Q.R. of order $2n$?

Heuristics: n-pt Q.R. involves $2n$ d.o.f.

$= \dim \mathcal{P}_{2n-1}$

Find c_j, w_j, $j = 1, \dots, n$:

$$Q_n(\{t \to t^\ell\}) = \sum_{j=1}^{n} w_j c_j^\ell = \int_a^b t^\ell\,dt, \quad \ell = 0, \dots, 2n-1$$

$2n \times 2n$ non-linear system of equations

Ex. 7.4.2.2: (2-pt quadrature rule of order 4)

$$Q_n(p) = \int_a^b p(t)\,dt \;\forall p \in \mathcal{P}_3 \;\Leftrightarrow\; Q_n(\{t \mapsto t^q\}) = \frac{1}{q+1}(b^{q+1} - a^{q+1}), \quad q = 0,1,2,3.$$

$\Updownarrow$

4 equations for weights w_j and nodes c_j, $j = 1,2$ $(a = -1, b = 1)$, *cf.* Rem. 7.4.1.7

$$\int_{-1}^{1} 1\,dt = 2 = 1w_1 + 1w_2 \;,\quad \int_{-1}^{1} t\,dt = 0 = c_1 w_1 + c_2 w_2$$
$$\int_{-1}^{1} t^2\,dt = \frac{2}{3} = c_1^2 w_1 + c_2^2 w_2 \;,\quad \int_{-1}^{1} t^3\,dt = 0 = c_1^3 w_1 + c_2^3 w_2 \,. \tag{7.4.2.3}$$

NumCSE@ETHZ

➢ weights & nodes: $\left\{w_2 = 1, w_1 = 1, c_1 = 1/3\sqrt{3}, c_2 = -1/3\sqrt{3}\right\}$

▬ quadrature formula (order 4): $\int_{-1}^{1} f(x)\,dx \approx f\left(\frac{1}{\sqrt{3}}\right) + f\left(-\frac{1}{\sqrt{3}}\right)$ (7.4.2.4)

§ 7.4.2.5 (Construction of n-point quadrature rules with maximal order $2n$)

Optimist's **assumption**: ∃ family of n-point quadrature formulas on $[-1,1]$

$$Q_n(f) := \sum_{j=1}^{n} w_j^{(n)} f(c_j^{(n)}) \approx \int_{-1}^{1} f(t)\,\mathrm{d}t\,, \quad w_j^{(n)} \in \mathbb{R}\,, n \in \mathbb{N}\,,$$

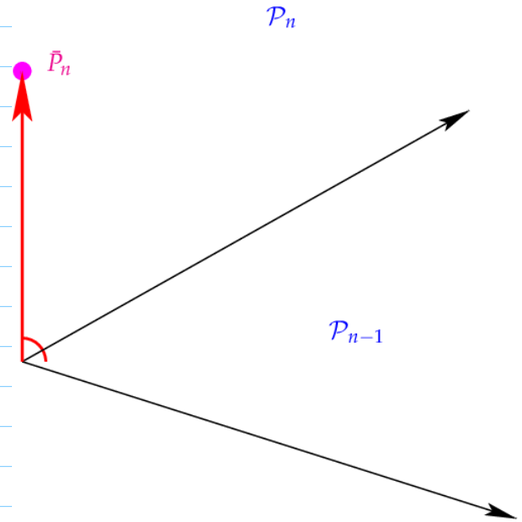
of order $2n$ ⇔ exact for polynomials $\in \mathcal{P}_{2n-1}$. (7.4.2.6)

Define $\bar{P}_n(t) := (t - c_1^{(n)}) \cdot \cdots \cdot (t - c_n^{(n)})\,, \quad t \in \mathbb{R} \;\Rightarrow\; \bar{P}_n \in \mathcal{P}_n\,.$

$q \in \mathcal{P}_{n-1}$:

$$\int_{-1}^{1} \underbrace{q(t)\,\bar{P}_n(t)}_{\in \mathcal{P}_{2n-1}}\,dt = \sum_{j=1}^{n} w_j^{(n)}\, q(c_j^{(n)})\, \underbrace{\bar{P}_n(c_j^{(n)})}_{=0} = 0$$

$L^2([-1,1])$-orthogonality : $\int_{-1}^{1} q\,\bar{P}_n\,dt = 0 \quad \forall q \in \mathcal{P}_{n-1}$ (7.4.2.7)



$\mathcal{P}_n$ equipped with the inner product $(p,q) \mapsto \int_{-1}^{1} p(t)q(t)\,\mathrm{d}t$ can be viewed as an Euclidean space:

◁ $\bar{P}_n \perp \mathcal{P}_{n-1}$

- $\bar{P}_n$ leading coefficient = 1

▷ $\bar{P}_n$ unique

▷ $c_j^{(n)}$ fixed !

The nodes of an n-point quadrature formula of order $2n$, if it exists, must coincide with the unique zeros of the polynomials $\bar{P}_n \in \mathcal{P}_n \setminus \{0\}$ satisfying (7.4.2.7).

Theorem 7.4.2.11. Existence of n-point quadrature formulas of order $2n$

Let $\{\bar{P}_n\}_{n\in\mathbb{N}_0}$ be a family of non-zero polynomials that satisfies

- $\bar{P}_n \in \mathcal{P}_n$,
- $\int_{-1}^{1} q(t)\bar{P}_n(t)\,\mathrm{d}t = 0$ *for all* $q \in \mathcal{P}_{n-1}$ ($L^2([-1,1])$-*orthogonality*),
- *The set* $\{c_j^{(n)}\}_{j=1}^{m}$, $m \le n$, *of real zeros of* $\bar{P}_n$ *is contained in* $[-1,1]$.

} Assumption

Then the quadrature rule (→ Def. 7.2.0.1) $Q_n(f) := \sum_{j=1}^{m} w_j^{(n)} f(c_j^{(n)})$

with weights chosen according to Thm. 7.4.1.6 provides a quadrature formula of order $2n$ on $[-1,1]$.

↳ Q.R. has order $\ge n$

Proof: $p \in \mathcal{P}_{2n-1}$

Polynomial division: $p = h\bar{P}_n + r$ with $h, r \in \mathcal{P}_{n-1}$

$$\int_{-1}^{1} p(t)\,\mathrm{d}t = \underbrace{\int_{-1}^{1} h(t)\bar{P}_n(t)\,\mathrm{d}t}_{=0 \text{ by (7.4.2.12)}} + \int_{-1}^{1} r(t)\,\mathrm{d}t \overset{(*)}{=} \sum_{j=1}^{m} w_j^{(n)} r(c_j^{(n)}), \qquad (7.4.2.14)$$

: orthogonality

Note: $r(c_j^{(n)}) = p(c_j^{(n)})$

$$\sum_{j=1}^{m} w_j^{(n)} p(c_j^{(n)}) \overset{(7.4.2.13)}{=} \sum_{j=1}^{m} w_j^{(n)} h(c_j^{(n)}) \underbrace{\bar{P}_n(c_j^{(n)})}_{=0} + \sum_{j=1}^{m} w_j^{(n)} r(c_j^{(n)}) \overset{(7.4.2.14)}{=} \int_{-1}^{1} p(t)\,\mathrm{d}t.$$

□

P_n exist & unique

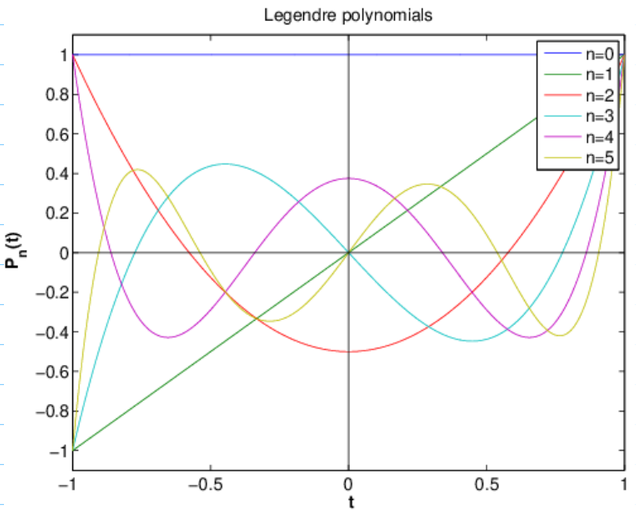
Definition 7.4.2.16. Legendre polynomials

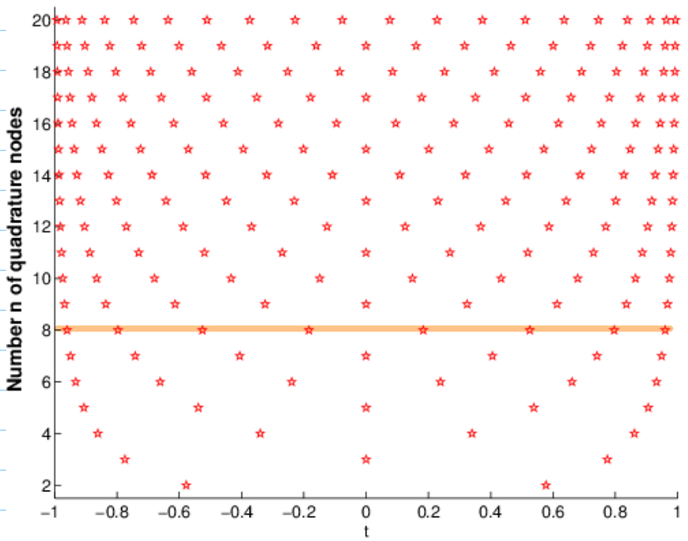
The n-th Legendre polynomial P_n is defined by

- $P_n \in \mathcal{P}_n$,
- $\int_{-1}^{1} P_n(t)q(t)\,\mathrm{d}t = 0 \quad \forall q \in \mathcal{P}_{n-1}$,
- $P_n(1) = 1$.

orthogonal polynomial

still open





Lemma 7.4.2.17. Zeros of Legendre polynomials

P_n has n distinct zeros in $]-1,1[$.

Zeros of Legendre polynomials = Gauss points

nodes of the unique n-pt Q.R. of order 2n!

(5)

Proof (indirect):

Assume: P_n has only $k < n$ zeros $x_j \in [-1,1]$, at which it changes sign

$$q(t) := \pm \prod_{j=1}^{k} (t - x_j)$$

$\Rightarrow$ $q \in \mathcal{P}_k$ & $P_n q \geq 0$ in $[-1,1]$

$$\int_{-1}^{1} P_n q \, dt > 0$$

contradicts orthogonality

Definition 7.4.2.18. Gauss-Legendre quadrature formulas

The n-point Quadrature formulas whose nodes, the Gauss points, are given by the zeros of the n-th Legendre polynomial ($\rightarrow$ Def. 7.4.2.16), and whose weights are chosen according to Thm. 7.4.1.6, are called **Gauss-Legendre quadrature formulas**.

ETH Lecture 401-0663-00L Numerical Methods for CSE

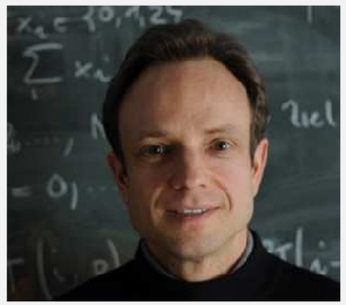
Course Video

Section 7.4.3: (Gauss-Legendre) Quadrature Error Estimates

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 22, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 6.2.2

Duration: 32 minutes

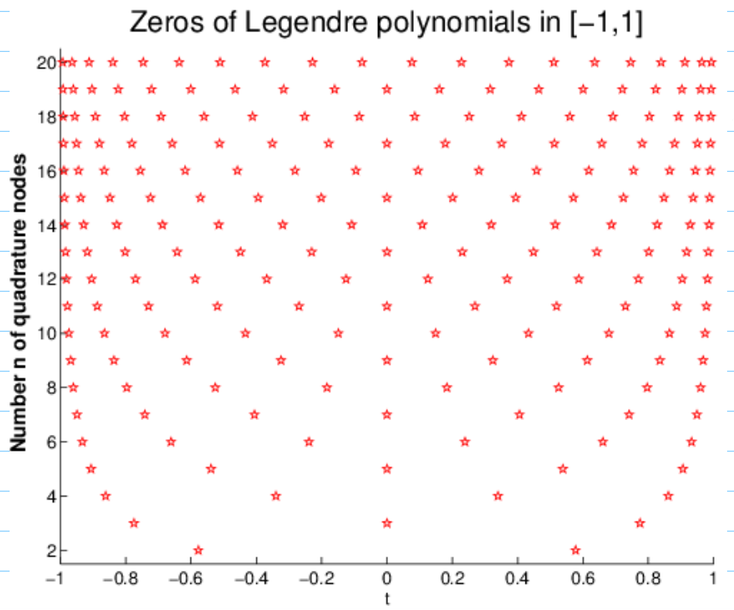
VII Numerical Quadrature

7.4 Gauss Quadrature

Definition 7.2.0.1. Quadrature formula/quadrature rule

An n-point quadrature formula/quadrature rule (QR) on $[a,b]$ provides an approximation of the value of an integral through a *weighted sum* of point values of the integrand:

$$\int_a^b f(t)\,\mathrm{d}t \approx Q_n(f) := \sum_{j=1}^{n} w_j^n f(c_j^n)\,. \tag{7.2.0.2}$$



Gauss nodes

↓

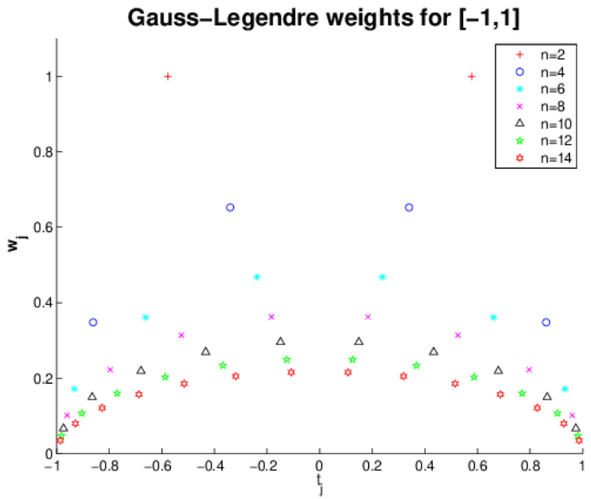
unique n-pt Q.R. achieving order 2n

7.4.3 Quadrature Error Estimates

Obviously ▷

Lemma 7.4.3.1. Positivity of Gauss-Legendre quadrature weights

The weights of the Gauss-Legendre quadrature formulas are positive.



[also CC-rules]

Proof: [Gauss nodes $c_j^{(n)}$]

$$q_k(t) := \prod_{\substack{j=1 \\ j\neq k}}^{n} (t - c_j^{(n)})^2 \in \mathcal{P}_{2n-2}, \quad k = 1, \dots, n$$

$q_k(c_j^{(n)}) = 0$ for $j \neq k$

$$0 < \int_{-1}^{1} q_k(t)\,dt = \sum_{j=1}^{n} w_j q_k(c_j^{(n)}) = w_k \underbrace{q_k(c_k^{(n)})}_{>0}$$

□

§ 7.4.3.2 (Quadrature error and best-approximation error)

$Q_n \hat{=}$ Q.R. of order $q \geq 1$ on $[a,b]$

Quadrature error, $f \in C^0([a,b])$

$$E_n(f) = E_n(f-p) \quad \forall p \in \mathcal{P}_{q-1}$$

$$= \left| \int_a^b (f-p)(t)\,dt - \sum_{j=1}^{n} w_j (f-p)(c_j) \right|$$

$$\leq |b-a| \, \|f-p\|_\infty + \sum_{j=1}^{n} |w_j| \, \|f-p\|_\infty$$

$$\leq \left(|b-a| + \sum_{j=1}^{\infty} |w_j| \right) \|f-p\|_\infty$$

If $w_j \geq 0$ $\overset{\text{order } q\geq 1}{\Longrightarrow}$ $\sum_{j=1}^{n} |w_j| = b-a$

▷ $E_n(f) \leq 2(b-a)\,\|f-p\|_\infty \quad \forall p \in \mathcal{P}_{q-1}$

NumCSE@ETHZ

Theorem 7.4.3.3. Quadrature error estimate for quadrature rules with positive weights

For every n-point quadrature rule Q_n as in (7.2.0.2) of order $q \in \mathbb{N}$ with weights $w_j \geq 0$, $j = 1,\dots,n$ the quadrature error satisfies

$$E_n(f) := \left| \int_a^b f(t)\,dt - Q_n(f) \right| \leq 2|b-a| \underbrace{\inf_{p\in\mathcal{P}_{q-1}} \|f-p\|_{L^\infty([a,b])}}_{\text{best approximation error}} \quad \forall f \in C^0([a,b]) \,. \tag{7.4.3.4}$$

▷ Apply estimates of Ch. 6

Theorem 6.2.1.11. L^∞ polynomial best approximation estimate

If $f \in C^r([-1,1])$ (r times continuously differentiable), $r \in \mathbb{N}$, then, for any polynomial degree $n \geq r$,

$$\inf_{p\in\mathcal{P}_n} \|f-p\|_{L^\infty([-1,1])} \leq (1+\pi^2/2)^r \frac{(n-r)!}{n!} \left\| f^{(r)} \right\|_{L^\infty([-1,1])} ,$$

where

$$\left\| f^{(r)} \right\|_{L^\infty([-1,1])} := \max_{x\in[-1,1]} |f^{(r)}(x)| .$$

▷ Asymptotics of $E_n(f)$ for $n \to \infty$

[for Q, R. w/ $w_j > 0$]

- Case of limited smoothness of f:

$f \in C^r([a,b])$, $f \notin C^{r+1}([a,b])$, $r < q$

$\Rightarrow$ $E_n(f) = O(n^{-r})$ for $n \to \infty$

$\hat{=}$ algebraic cvg with rate r

- Analytic C^∞-integrand

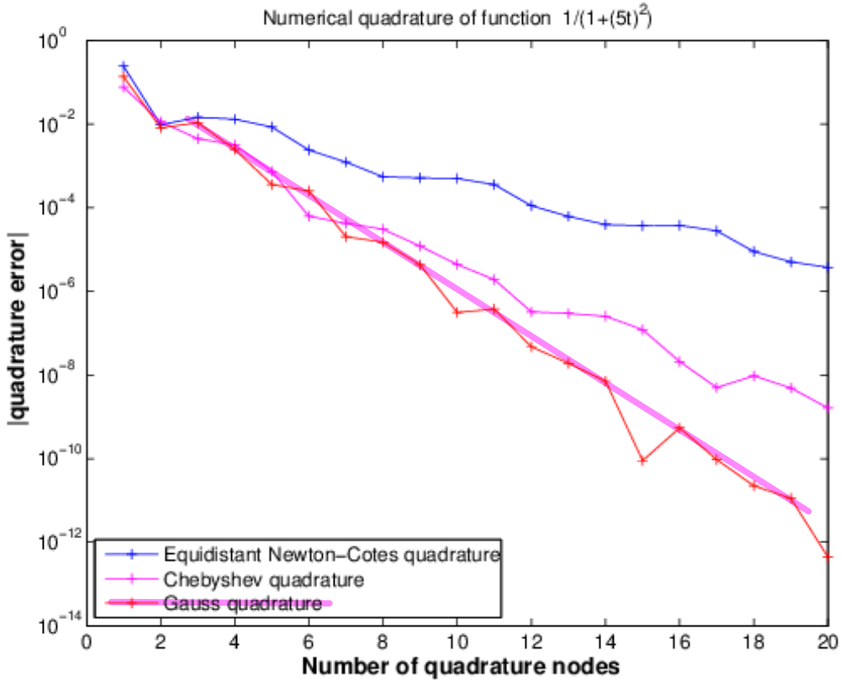
$\exists q \in [0,1[$: $E_n(f) = O(q^n)$ for $n \to \infty$

$\hat{=}$ exponential cvg.

Ex. 7.4.3.9 : (Convergence of global quadrature rules)

$I = [0,1]$

1. $f_1(t) = \frac{1}{1+(5t)^2}$, an analytic function, see Rem. 6.2.2.59,
2. $f_2(t) = \sqrt{t}$, merely continuous, derivatives singular in $t = 0$.

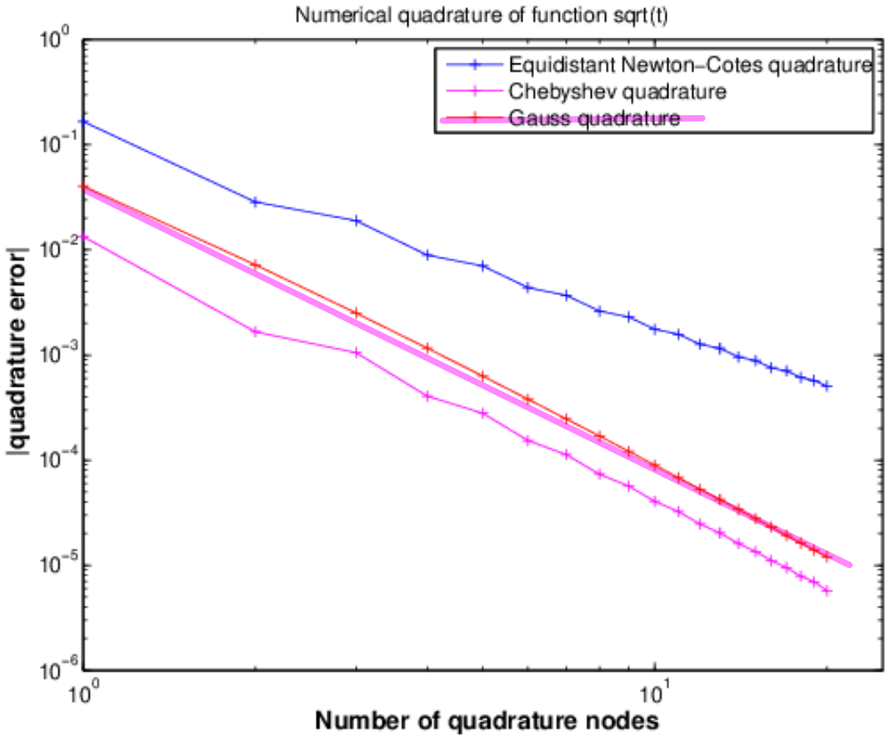


lin - log plot

⇓

empiric exp - cvg.

quadrature error, $f_1(t) := \frac{1}{1+(5t)^2}$ on $[0,1]$



log - log plot

⇓

emp. alg. cvg.

quadrature error, $f_2(t) := \sqrt{t}$ on $[0,1]$

NumCSE@ETHZ

Rem. 7.4.3.10 (Removing a singularity by transformation)

$$I := \int_0^b \underbrace{\sqrt{t}}\, f(t)\,dt \quad \text{with } f \in C^\infty([0,b])$$

↳ non-smooth integrand

[slow alg. cvg. of GL-Q.R.]

Idea: substitution $s = \sqrt{t}$ $\left(\begin{array}{l} ds = \frac{1}{2\sqrt{t}}\,dt \\ dt = 2s\,ds \end{array}\right)$

$$\triangleright \quad \int_0^b \sqrt{t}\, f(t)\,dt = 2\int_0^{\sqrt{b}} \underbrace{s\, f(s^2)}\, ds$$

$\in C^\infty([0,\sqrt{b}])$

[analytic]

exponential cvg. of G.-L. Q.R.

Rem 7.4.3.12 (The message of asymptotic estimates)

What does $E_n(f) = O(n^{-r})$ tell us?

[Implicit sharpness: $E_n(f) \approx \underbrace{C}_{\text{unknown, elusive}}\, n^{-r}$ (*)]

▷ no information about $E_n(f)$ for concrete n

(*) tells us how much additional work $\sim n$ is needed to reduce $E_n(f)$ by a factor $S > 1$

goal: $\dfrac{E_n(f)}{E_m(f)} \overset{!}{=} S$

(*) $\Rightarrow$ $\dfrac{C n^{-r}}{C m^{-r}} = S \Rightarrow \boxed{m = n \cdot S^{1/r}}$

In the case of *algebraic convergence* with rate $r \in \mathbb{R}$ a reduction of the quadrature error by a factor of ρ is bought by an increase of the number of quadrature points by a factor of $\rho^{1/r}$.

NumCSE@ETHZ

Exp. cvg.: $E_n(f) = O(q^n)$

[$E_n(f) \approx C q^n$ (+)]

Goal: $\frac{E_n(f)}{E_m(f)} \overset{!}{=} \rho$

(+) $\Rightarrow$ $\frac{C q^n}{C q^m} \overset{!}{=} \rho \Rightarrow m - n \geq -\frac{\log \rho}{\log q}$

In the case of *exponential convergence* (7.4.3.15) a fixed increase of the number of quadrature points by $-\log\rho : \log q$ results in a reduction of the quadrature error by a factor of $\rho > 1$.

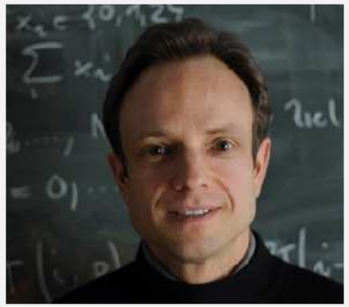
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 7.5: Composite Quadrature

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 22, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 6.6.1

Duration: minutes

VII Numerical Quadrature

7.5 Composite Quadrature

interpolation schemes → approximation schemes → quadrature schemes

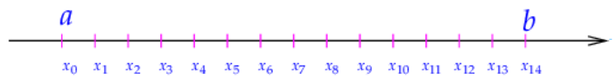
↳ Q.R. induced by piecewise polynomial interpolation

A different view:

- Start from mesh

Terminology:

- $x_j \widehat{=}$ nodes of the mesh $\mathcal{M}$, $j = 1, \ldots, m$
- $[x_{j-1}, x_j[\widehat{=}$ intervals/cells of the mesh,
- $h_{\mathcal{M}} := \max_j |x_j - x_{j-1}| \widehat{=}$ mesh width,
- If $x_j = a + jh \widehat{=}$ equidistant (uniform) mesh with meshwidth $h > 0$



$$\int_a^b f(t)\,\mathrm{d}t = \sum_{j=1}^{m} \int_{x_{j-1}}^{x_j} f(t)\,\mathrm{d}t\,. \tag{7.5.0.1}$$

↳ apply an n_j-pt local Q.R.

General construction of composite quadrature rules

Idea:
- Partition integration domain $[a,b]$ by a mesh/grid ($\rightarrow$ Section 6.6)
  $\mathcal{M} := \{a = x_0 < x_1 < \ldots < x_m = b\}$
- Apply quadrature formulas from Section 7.3, Section 7.4 locally on mesh intervals $I_j := [x_{j-1}, x_j]$, $j = 1, \ldots, m$, and sum up.

$\triangleright$ composite quadrature rule

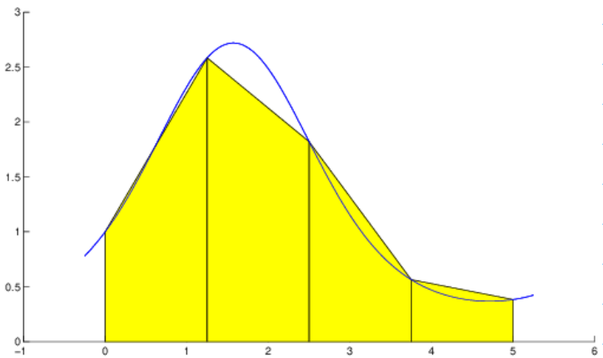
cost = #{f-eval} $\sim \sum_{j=0}^{m} n_j$

Ex. 7.5.0.3 (Simple composite polynomial quadrature rules)

$n_j = 2$

Composite trapezoidal rule, *cf.* (7.3.0.5)

$$\int_a^b f(t)\,\mathrm{d}t = \tfrac{1}{2}(x_1 - x_0)f(a) + \sum_{j=1}^{m-1} \tfrac{1}{2}(x_{j+1} - x_{j-1})f(x_j) + \tfrac{1}{2}(x_m - x_{m-1})f(b)\,. \qquad (7.5.0.4)$$

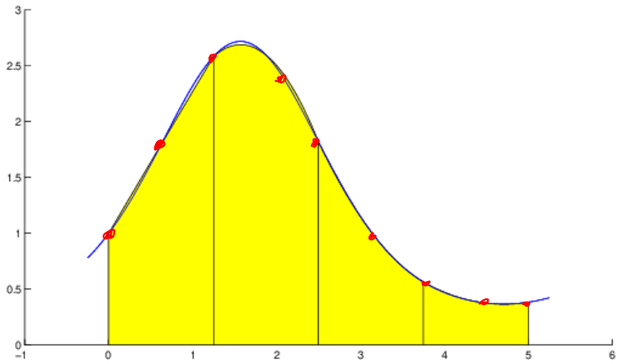


$\triangleright$ Local order = 2

$n_j = 3$

Composite Simpson rule, *cf.* (7.3.0.6)

$$\int_a^b f(t)\,\mathrm{d}t = \tfrac{1}{6}(x_1 - x_0)f(a) + \sum_{j=1}^{m-1} \tfrac{1}{6}(x_{j+1} - x_{j-1})f(x_j) + \sum_{j=1}^{m} \tfrac{2}{3}(x_j - x_{j-1})f(\tfrac{1}{2}(x_j + x_{j-1})) + \tfrac{1}{6}(x_m - x_{m-1})f(b)\,. \qquad (7.5.0.5)$$



$\triangleright$ Local order = 4

§ 7.5.0.9 (Quadrature error estimate for composite polynomial quadrature rules)

In $[x_{j-1}, x_j]$: Use order-q n_j-pts local Q.R. with weights > 0 !

$\triangleright$ Q.E. on $[x_{j-1}, x_j]$

$\hookrightarrow \leq 2\,|x_j - x_{j-1}| \inf_{p \in \mathcal{P}_{q-1}} \|f - p\|_{\infty,]x_{j-1}, x_j[}$

③

$f \in C^{n+1}([a,b])$:

Thm. 6.2.2.15 $\Rightarrow$ $$\|f - \mathrm{L}_{\mathcal{T}} f\|_{L^\infty(I)} \leq \frac{\|f^{(n+1)}\|_{L^\infty(I)}}{(n+1)!} \underbrace{\max_{t \in I} |(t-t_0)\cdot\cdots\cdot(t-t_n)|}_{\leq |I|^{n+1}} . \qquad (6.2.2.22)$$

$f \in C^r([a,b])$

$$\left| \int_{x_{j-1}}^{x_j} f(t)\,\mathrm{d}t - Q_{n_j}^j(f) \right| \leq C |x_j - x_{j-1}|^{\min\{r,q\}+1} \left\| f^{(\min\{r,q\})} \right\|_{L^\infty([x_{j-1},x_j])} . \qquad (7.3.0.10)$$

Adding up:

$$\left| \int_{x_0}^{x_m} f(t)\,\mathrm{d}t - Q(f) \right| \leq C \sum_{j=1}^{m} h_j^{\min\{r,q_j\}+1} \left\| f^{(\min\{r,q_j\})} \right\|_{L^\infty([x_{j-1},x_j])} ,$$

$$\left| \int_{x_0}^{x_m} f(t)\,\mathrm{d}t - Q(f) \right| \leq C\, h_{\mathcal{M}}^{\min\{q,r\}} |b-a| \left\| f^{(\min\{q,r\})} \right\|_{L^\infty([a,b])} , \qquad (7.5.0.10)$$

"h-convergence"

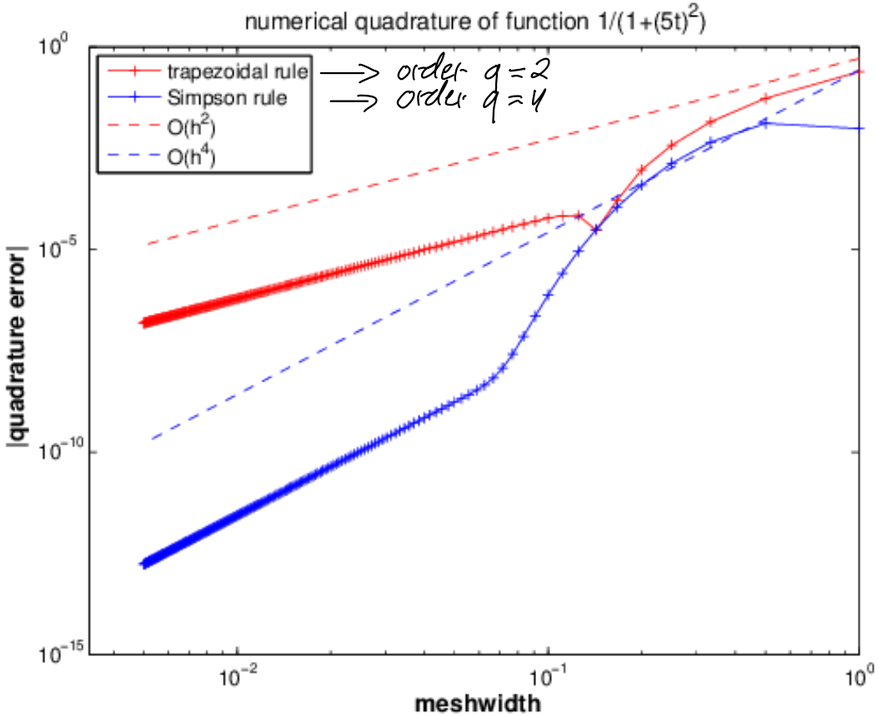
smoothness limits cvg.

Algebraic cvg. for $h_{\mathcal{M}} \to 0$ with rate $\min\{q,r\}$

Exp.: (Quadrature error for composite quadrature rules)

▷ On $[0,1]$, equidistant mesh

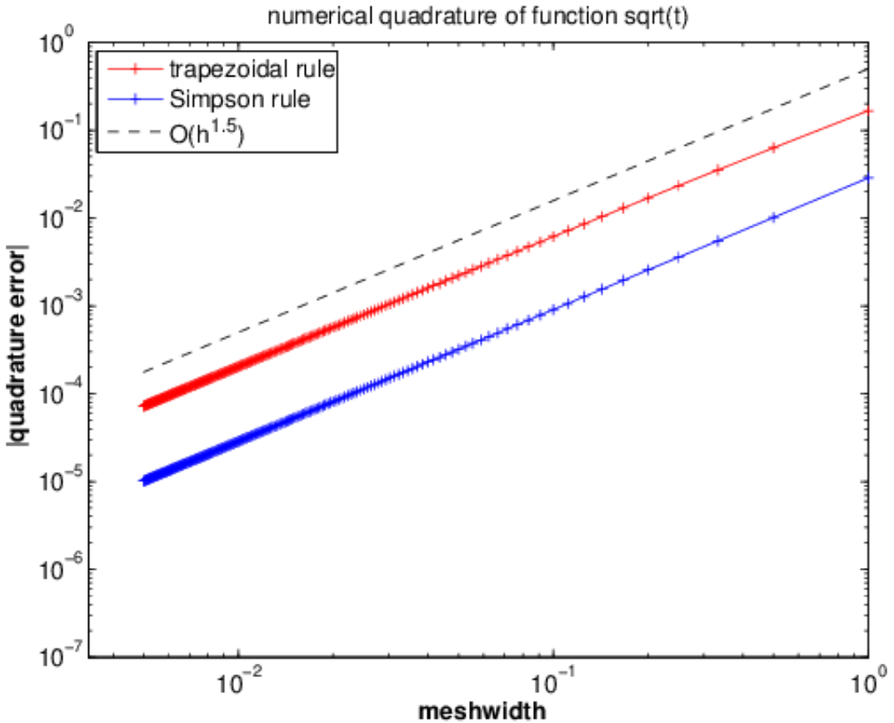
$$x_j = j/m, \quad j = 0, \ldots, m$$



△ log-log plot ⇓ alg. cvg.

quadrature error, $f_1(t) := \frac{1}{1+(5t)^2}$ on $[0,1]$

↳ C^∞-integrand



quadrature error, $f_2(t) := \sqrt{t}$ on $[0,1]$

↳ $\notin C^1([0,1])$, singular

A log-log plot

↓

alg. cvg.

Alg. cvg. in $h_m \to 0$ with rate 1.5

Rem. 7.5.0.13 (Composite quadrature rule vs. global quadrature rules)

↳ G.-L. Q.R.

on equidistant meshes

	Comp. Q.R.	G.-L. Q.R.
$f \in C^r([a,b])$:	$O(n^{-\min\{q,r\}})$	$O(n^{-r})$
$f \in C^\infty([a,b])$ (analytic)	$O(n^{-q})$	exp. cvg.

[G.-L. Q.R. is at least as good, always achieves optimal rate]

Use Gauss-Legendre quadrature instead of fixed order composite quadrature on equidistant meshes. *

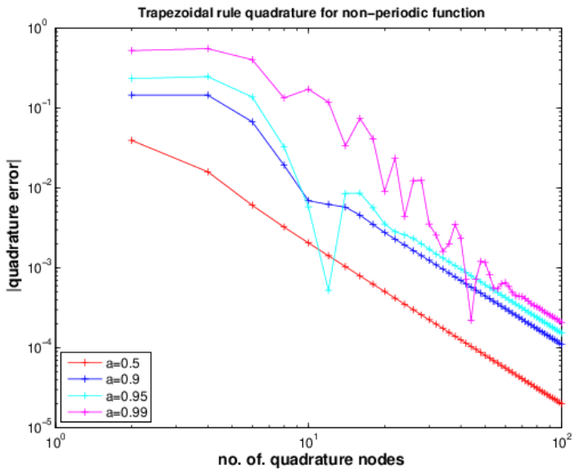
* Comp. Q.R. may be better on locally refined meshes

Exp. 7.5.0.16 (Empiric convergence of equidistant trapezoidal rule)

$$\int_a^b f(t)\,\mathrm{d}t \approx T_m(f) := h\left(\tfrac{1}{2}f(a) + \sum_{k=1}^{m-1} f(kh) + \tfrac{1}{2}f(b)\right), \quad h := \frac{b-a}{m}. \qquad (7.5.0.17)$$

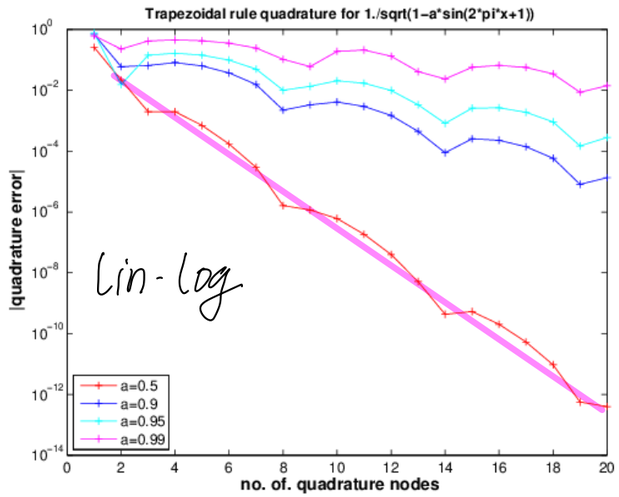
$$f(t) = \frac{1}{\sqrt{1 - a\sin(2\pi t - 1)}}, \quad 0 < a < 1. \quad \in C^\infty$$

1-periodic



quadrature error for $T_n(f)$ on $[0, \frac{1}{2}]$

merely algebraic convergence



quadrature error for $T_n(f)$ on $[0,1]$

exponential convergence !!

§ 7.5.0.18 (The magic of the equidistant trapezoidal rule)

f = 1-periodic

$T_n(f) = \frac{1}{n}\sum_{j=0}^{n-1} f(j/n)$: trapezoidal rule

For $f_k(t) = e^{2\pi i k t}$, $k \in \mathbb{Z}$

$$\int_0^1 e^{2\pi i k t}\,dt = \delta_{0,k} \qquad T_n(f_k) = \frac{1}{n}\sum_{j=0}^{n-1} e^{2\pi i \frac{kj}{n}} = \begin{cases} 1, & k \in n\mathbb{Z} \\ 0, & \text{else} \end{cases}$$

$$\Rightarrow \quad T_n(f_k) = \int_0^1 f_k(t)\,dt, \quad -n < k < n$$

Lemma 7.5.0.22. Exact quadrature by equidistant trapezoidal rule

The n-point equidistant trapezoidal quadrature rule

$$\int_a^b f(t)\,\mathrm{d}t \approx T_n(f) := h\left(\tfrac{1}{2}f(a) + \sum_{k=1}^{n-1} f(kh) + \tfrac{1}{2}f(b)\right), \quad h := \frac{b-a}{n}. \qquad (7.5.0.17)$$

is exact for trigonometric polynomials (→ Section 6.5.1) of degree $\leq 2n-2$.

NumCSE@ETHZ

$$\mathcal{S}_{2n}^{T} = \operatorname{Span}\{ t \mapsto e^{2\pi i k t}, -n \le k \le n \}$$

▷ $$\left| \int_0^1 f(t)\,dt - T_n(f) \right| \le 2 \inf_{p \in \mathcal{S}_{2n-2}^{T}} \|f - p\|_{L^\infty([0,1])}$$

↑ best approximation error estimates by trig. pol.

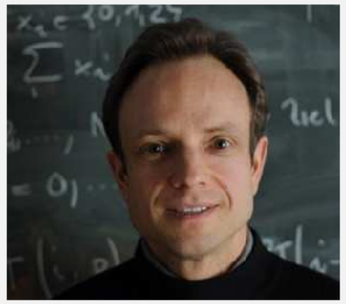
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 7.6: Adaptive Quadrature

Prof. R. Hiptmair, SAM, ETH Zurich

Date: November 22, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 7.5

Duration: 20 minutes

VIII Numerical Quadrature

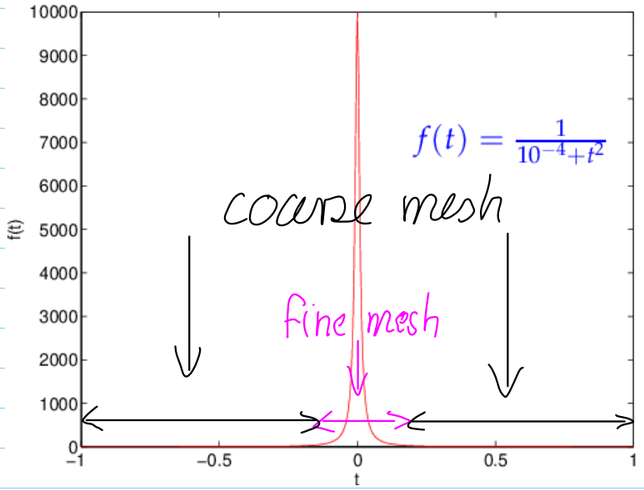
7.6. Adaptive Quadrature

General construction of composite quadrature rules

Idea:
- Partition integration domain $[a,b]$ by a mesh/grid (→ Section 6.6)
  $\mathcal{M} := \{a = x_0 < x_1 < \ldots < x_m = b\}$
- Apply quadrature formulas from Section 7.3, Section 7.4 locally on mesh intervals $I_j := [x_{j-1}, x_j]$, $j = 1, \ldots, m$, and sum up.

composite quadrature rule

Ex. 7.6.0.2 (Rationale for adaptive quadrature)



◁ Integrate by composite trapezoidal rule

Goal: Automated choice of good mesh

§ 7.6.0.4 (Goal: Equidistribution of errors)

Ex: composite T.R., local quad. errors

$$\left|\int_{x_{k-1}}^{x_k} f(t)\,\mathrm{d}t - \frac{1}{2}(f(x_{k-1})+f(x_k))\right| \le \frac{1}{8}h_k^3\|f''\|_{L^\infty([x_{k-1},x_k])}, \quad h_k := x_k - x_{k-1}. \tag{7.6.0.3}$$

Heuristics: choosing mesh such that local errors are equidistribution

h_k : $h_k^3\|f''\|_{\infty,[x_{k-1},x_k]} = C \quad \forall k$

A priori:
based on prior knowledge about f

A posteriori:
"automatic" based on information gained during the computation.

▷ A posteriori adaptive mesh refinement

Adaptation loop for numerical quadrature

(1) ESTIMATE: based on available information compute an approximation for the quadrature error on every mesh interval.
(2) CHECK TERMINATION: if total error sufficient small → **STOP**
(3) MARK: single out mesh intervals with the largest or above average error contributions.
(4) REFINE: add node(s) inside the marked mesh intervals. GOTO (1)

Idea: local error estimation by comparing local results of two quadrature formulas Q_1, Q_2 of *different* order → local error estimates

heuristics: $\mathrm{error}(Q_2) \ll \mathrm{error}(Q_1) \Rightarrow \mathrm{error}(Q_1) \approx Q_2(f) - Q_1(f)$.

Here: Q_1 = trapezoidal rule (order 2) ↔ Q_2 = Simpson rule (order 4)

Details: Initial mesh $\mathcal{M} = \{x_j\}_{j=0}^m$

❶ (Error estimation)

For $I_k = [x_{k-1}, x_k]$, $k = 1,\ldots,m$ (midpoints $p_k := \frac{1}{2}(x_{k-1}+x_k)$)

$$\mathrm{EST}_k := \left|\underbrace{\frac{h_k}{6}(f(x_{k-1})+4f(p_k)+f(x_k))}_{\text{Simpson rule}} - \underbrace{\frac{h_k}{4}(f(x_{k-1})+2f(p_k)+f(x_k))}_{\text{trapezoidal rule on split mesh interval}}\right|. \tag{7.6.0.11}$$

order = 4 order = 2

❷ (Check termination)

Simpson rule on $\mathcal{M}$ $\Rightarrow$ intermediate approximation $I \approx \int_a^b f(t)\,\mathrm{d}t$

If $\sum_{k=1}^{m} \mathrm{EST}_k \leq \mathrm{RTOL} \cdot I$ ($RTOL$:= prescribed relative tolerance) $\Rightarrow$ **STOP*** (7.6.0.12)

↳ estimate for total error of trapezoidal rule

* return approximation by Simpson rule

❸ (Marking)

Marked intervals: $\mathcal{S} := \{k \in \{1,\dots,m\} : \mathrm{EST}_k \geq \eta \cdot \frac{1}{m}\sum_{j=1}^{m} \mathrm{EST}_j\}$, $\eta \approx 0.9$. (7.6.0.13)

(average local error)

❹ (Local mesh refinement)

new mesh: $\mathcal{M}^* := \mathcal{M} \cup \{p_k := \frac{1}{2}(x_{k-1} + x_k) : k \in \mathcal{S}\}$. (7.6.0.14)

Then continue with step ❶ and mesh $\mathcal{M} \leftarrow \mathcal{M}^*$.

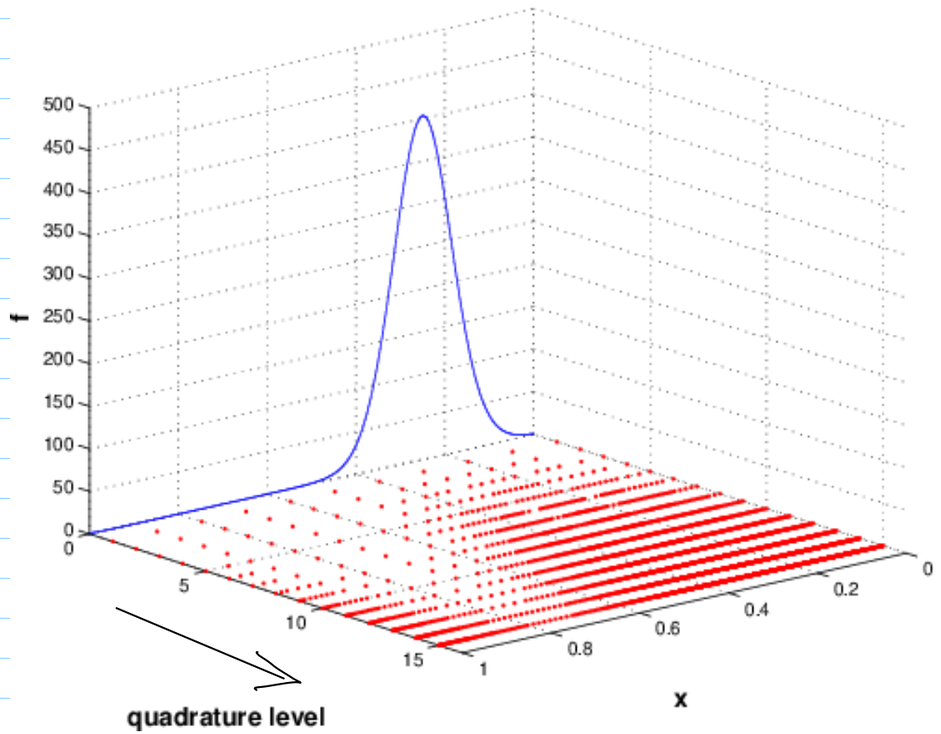
↑ add midpoints of marked cells

↓

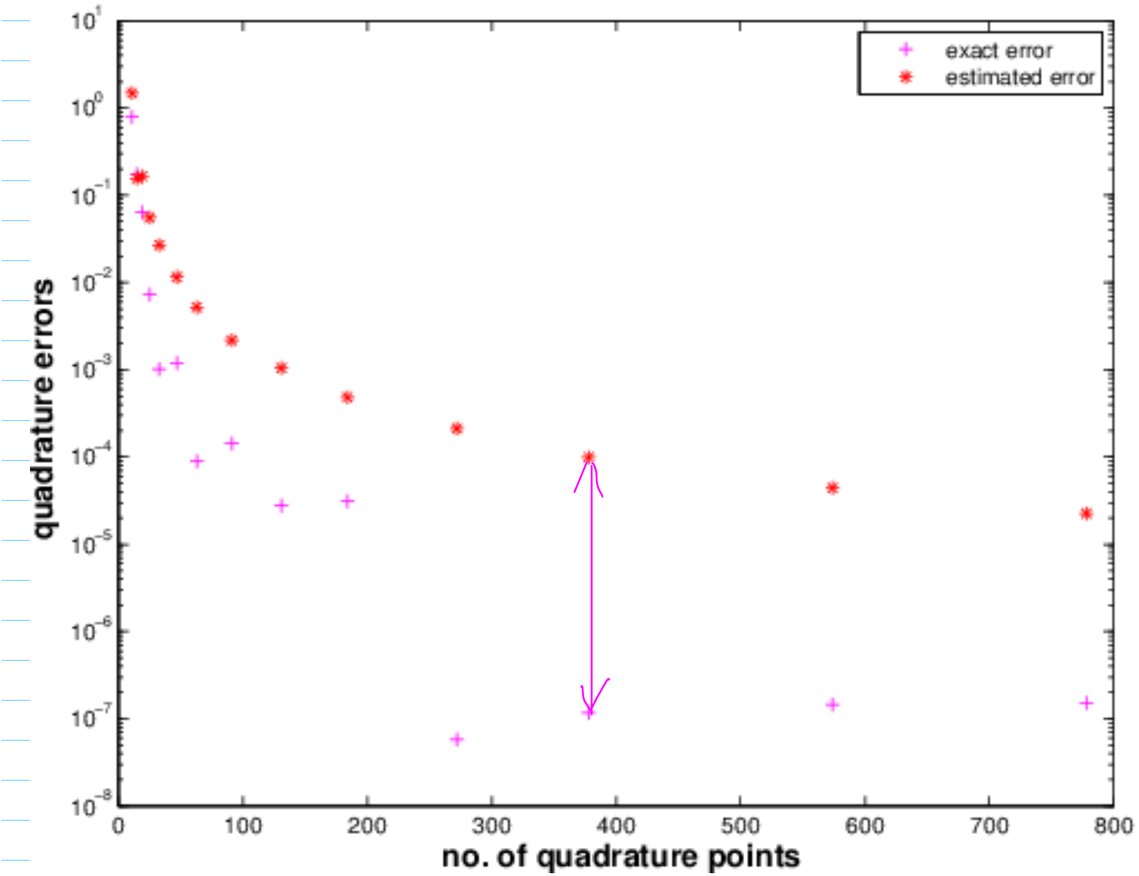
$\hat{=}$ adaptive loop

Exp. 7.6.0.18 (h-adaptive numerical quadrature)

$f(t) = \exp(6 \sin(2\pi t))$ on $[0,1]$



* = quadrature nodes "reasonably distributed"



▷ EST_K far off,
but refinement control work

ETH Lecture 401-0663-00L Numerical Methods for CSE

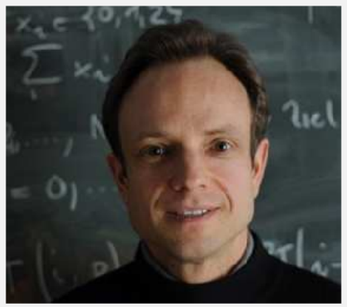
Course Video

Section 8.1: Iterative Methods for Non-Linear Systems of Equations: Introduction

Prof. R. Hiptmair, SAM, ETH Zurich

Date: December 1, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Ex. 2.1.0.3

Duration: 10 minutes

VIII Iterative Methods for Non-Linear Systems of Equations

8.1. Introduction

Abstract NLSE: $F(x) = 0$

$F : D \subset \mathbb{R}^n \longrightarrow \mathbb{R}^n$

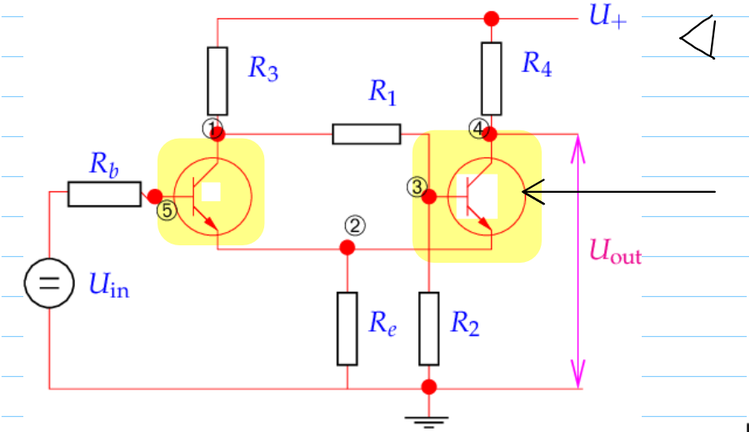
$n \in \mathbb{N}$: no. of (scalar) equs. = no. of (scalar) unknowns

LSE: $A \in \mathbb{R}^{n,n}$, $b \in \mathbb{R}^n$

$Ax = b \iff F(x) = 0$, $F(x) := b - Ax$

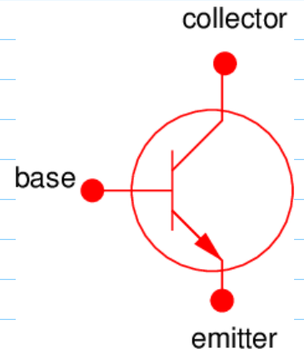
(2)

Ex. 8.1.0.1. (Non-linear electric circuit)



◁ Schmitt trigger circuit

NPN bipolar transistors



$$
\begin{aligned}
I_C &= I_S\left(e^{\frac{U_{BE}}{U_T}} - e^{\frac{U_{BC}}{U_T}}\right) - \frac{I_S}{\beta_R}\left(e^{\frac{U_{BC}}{U_T}} - 1\right) = I_C(U_{BE}, U_{BC})\,, \\
I_B &= \frac{I_S}{\beta_F}\left(e^{\frac{U_{BE}}{U_T}} - 1\right) + \frac{I_S}{\beta_R}\left(e^{\frac{U_{BC}}{U_T}} - 1\right) = I_B(U_{BE}, U_{BC})\,, \qquad (8.1.0.2) \\
I_E &= I_S\left(e^{\frac{U_{BE}}{U_T}} - e^{\frac{U_{BC}}{U_T}}\right) + \frac{I_S}{\beta_F}\left(e^{\frac{U_{BE}}{U_T}} - 1\right) = I_E(U_{BE}, U_{BC})\,.
\end{aligned}
$$

I_C, I_B, I_E: current in collector/base/emitter,
U_{BE}, U_{BC}: potential drop between base-emitter, base-collector.

Nodal analysis: $U_j \hat{=}$ potential at ⑦

$$
\begin{aligned}
&① : \quad R_3(U_1 - U_+) + R_1(U_1 - U_3) + I_C(U_5 - U_1, U_5 - U_2) = 0\,, \\
&② : \quad R_e U_2 + I_E(U_5 - U_1, U_5 - U_2) + I_E(U_3 - U_4, U_3 - U_2) = 0\,, \\
&③ : \quad R_1(U_3 - U_1) + I_B(U_3 - U_4, U_3 - U_2) = 0\,, \qquad (8.1.0.3) \\
&④ : \quad R_4(U_4 - U_+) + I_C(U_3 - U_4, U_3 - U_2) = 0\,, \\
&⑤ : \quad R_b(U_5 - U_{\text{in}}) + I_B(U_5 - U_1, U_5 - U_2) = 0\,.
\end{aligned}
$$

5 equations $\leftrightarrow$ 5 unknowns U_1, U_2, U_3, U_4, U_5 $\longrightarrow$ $\underline{x} \in \mathbb{R}^5$

$\Updownarrow$

$F(\underline{x}) = 0$

§ 8.1.0.5 (Generic/general non-linear systems of equations)

$$F(\underline{x}) = 0$$

- No general theory
- Often: F only implicit

NumCSE@ETHZ

▷ Review questions 8.1.0.6

[RQ 8.1.0.6.C requires knowledge about nodal analysis of circuits; may be skipped]

ETH Lecture 401-0663-00L Numerical Methods for CSE

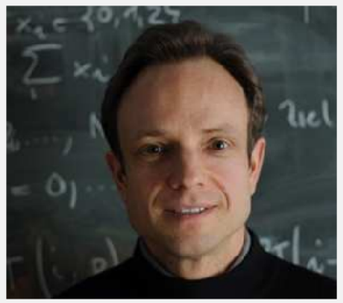
Course Video

Section 8.2.1: Iterative Methods: Fundamental Concepts

Prof. R. Hiptmair, SAM, ETH Zurich

Date: December 1, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- Sequences and convergence

Duration: 10 minutes

Note: Possible minor *mismatch of video and tablet notes*!

[Corrections and updates can be incorporated into tablet notes only]

VIII Iterative Methods for Non-Linear Systems of Equations

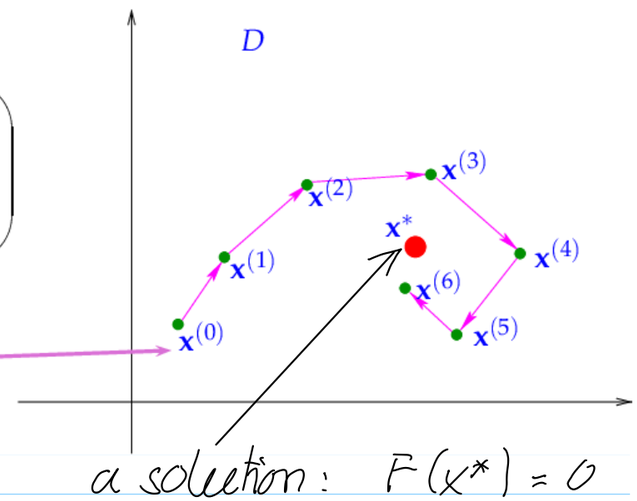
8.2. Iterative Methods

8.2.1. Fundamental Concepts

NLSE : $F(x) = 0$, $F : D \subset \mathbb{R}^n \to \mathbb{R}^n$

An iterative method for (approximately) solving the non-linear equation $F(\mathbf{x}) = 0$ is an algorithm generating an arbitrarily long sequence $\left(\mathbf{x}^{(k)}\right)_k$ of approximate solutions.

$\mathbf{x}^{(k)} \triangleq k$-th iterate



(2)

Convergence: [Def. 8.2.1.3] $\underline{x}^* := \lim_{k\to\infty} \underline{x}^{(k)}$ exists

More concrete: m-point iteration

$$\underline{x}^{(k+1)} = \Phi(\underline{x}^{(k)}, \dots, \underline{x}^{(k-m+1)}), \quad k \in \mathbb{N}$$

↳ iteration function

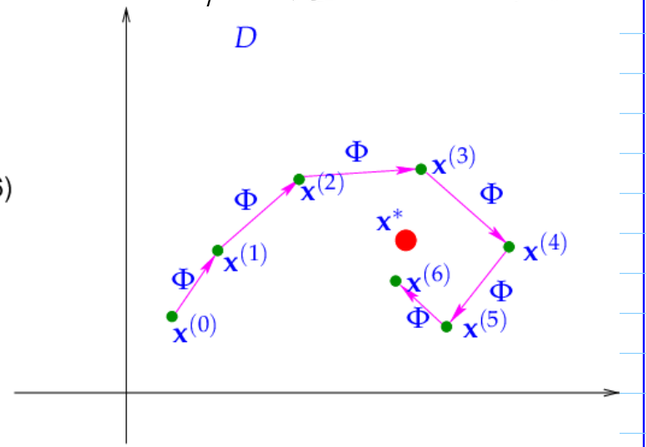
Needed: m initial guess $\underline{x}^{(0)}, \dots, \underline{x}^{(m-1)} \in \mathbb{R}^n$

Visualization of a 1-point iteration:

$$\mathbf{x}^{(k+1)} = \Phi(\mathbf{x}^{(k)}), \qquad (8.2.1.6)$$

with an iteration function

$$\Phi : D \subset \mathbb{R}^n \to \mathbb{R}^n .$$



Definition 8.2.1.7. Consistency of iterative methods

A statioanry m-point iterative method

$$\mathbf{x}^{(k+1)} = \Phi_F(\mathbf{x}^{(k)}, \dots, \mathbf{x}^{(k-m+1)}), \quad m \in \mathbb{N}, \qquad (8.2.1.5)$$

is **consistent** with the non-linear system of equations $F(\mathbf{x}) = \mathbf{0}$, if and only if

$$\Phi_F(\mathbf{x}^*, \dots, \mathbf{x}^*) = \mathbf{x}^* \iff F(\mathbf{x}^*) = \mathbf{0} .$$

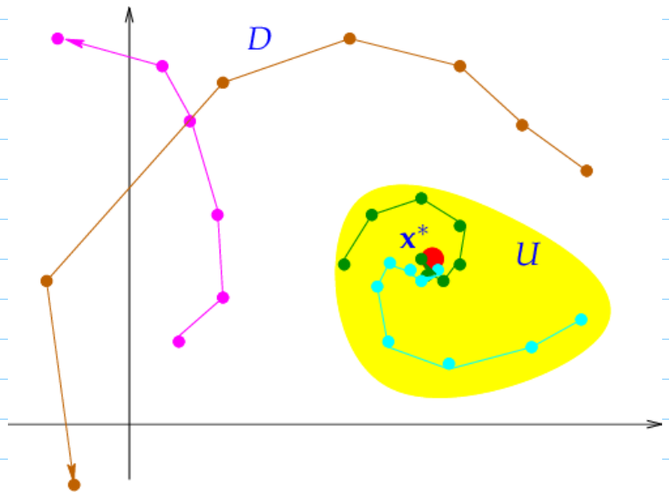
$\underline{x}^{(k)}$ cvgt.
Φ_F consistent
Φ_F continuous
$\Rightarrow F(\underline{x}^*) = 0$

Theorem 8.2.1.8. Consistency and convergence

The limit of a convergent sequence $\left(\mathbf{x}^{(k)}\right)_{k\in\mathbb{N}_0}$ generated by an m-point stationary iterative method with a continuous iteration function Φ_F that is consistent with $F(\mathbf{x}) = \mathbf{0}$ is a solution:

$$\mathbf{x}^{(k+1)} := \Phi_F(\mathbf{x}^{(k)}, \dots, \mathbf{x}^{(k-m+1)}), \quad \mathbf{x}^* := \lim_{k\to\infty} \mathbf{x}^{(k)} \quad \Rightarrow \quad F(\mathbf{x}^*) = \mathbf{0} .$$

§ 8.2.1.9 (Local convergence)



◁ Cvg. will heavily depend initial guesses

Local cvg.

NumCSE@ETHZ

▷ Review questions 8.2.1.11

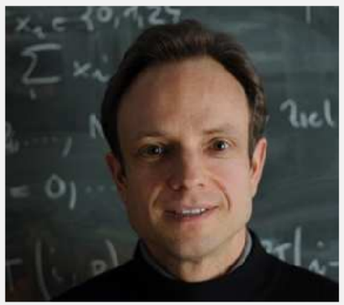
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 8.2.2: Iterative Methods: Speed of Convergence

Prof. R. Hiptmair, SAM, ETH Zurich

Date: December 1, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.
- Computing with logarithms
- Norms

Dependencies. Section 8.2.1

Duration: 26 minutes

VIII Iterative Methods for Non-Linear Systems of Equations

8.2. Iterative Methods

8.2.2. Speed of Convergence

How fast $\|\underline{x}^{(k)} - \underline{x}^*\| \to 0$?

Definition 8.2.2.1. Linear convergence

A sequence $\mathbf{x}^{(k)}$, $k = 0,1,2,\ldots$, in $\mathbb{R}^n$ **converges linearly** to $\mathbf{x}^* \in \mathbb{R}^n$,

$$\exists 0 < L < 1: \quad \left\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\right\| \leq L \left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\| \quad \forall k \in \mathbb{N}_0 .$$

Smallest L: rate of cvg.

Rem 8.2.2.6 : (Detecting linear convergence)

Assume sharpness : $\varepsilon_k := \|x^{(k+1)} - x^*\| \approx L \|x^{(k)} - x^*\|$

$\varepsilon_{k+1} \approx L \varepsilon_k$:

(i) Quotients : $\varepsilon_{k+1}/\varepsilon_k \approx L < 1$

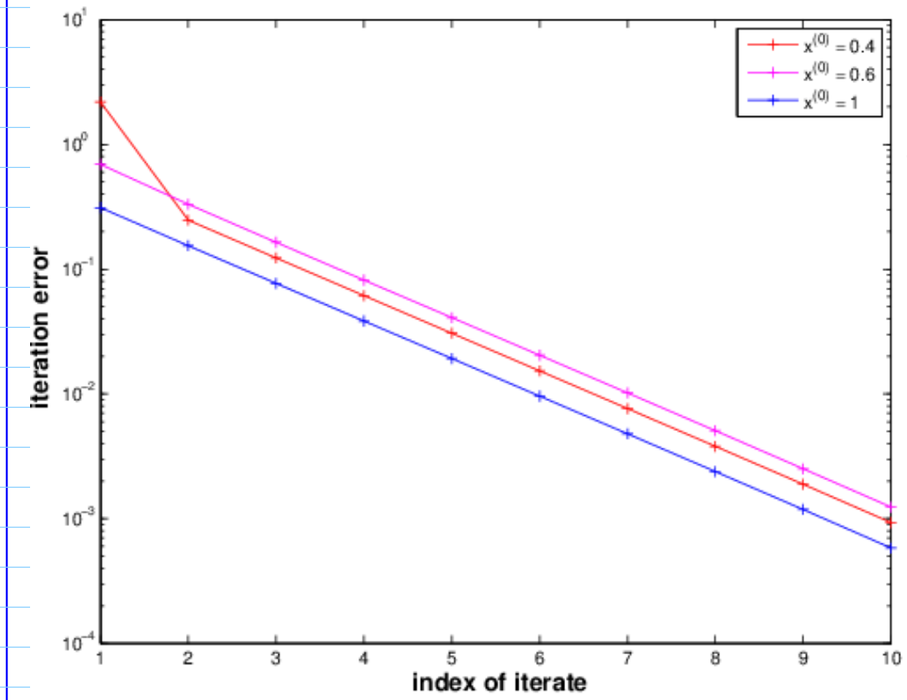
(ii) $\varepsilon_k \approx L^k \varepsilon_0$

$\log \varepsilon_k \approx k \underbrace{\log L}_{<0} + \log \varepsilon_0$

▷ Error points aligned in lin-log plot

k	$x^{(0)} = 0.4$		$x^{(0)} = 0.6$		$x^{(0)} = 1$	
	$x^{(k)}$	$\frac{\lvert x^{(k)} - x^{(15)}\rvert}{\lvert x^{(k-1)} - x^{(15)}\rvert}$	$x^{(k)}$	$\frac{\lvert x^{(k)} - x^{(15)}\rvert}{\lvert x^{(k-1)} - x^{(15)}\rvert}$	$x^{(k)}$	$\frac{\lvert x^{(k)} - x^{(15)}\rvert}{\lvert x^{(k-1)} - x^{(15)}\rvert}$
2	3.3887	0.1128	3.4727	0.4791	2.9873	0.4959
3	3.2645	0.4974	3.3056	0.4953	3.0646	0.4989
4	3.2030	0.4992	3.2234	0.4988	3.1031	0.4996
5	3.1723	0.4996	3.1825	0.4995	3.1224	0.4997
6	3.1569	0.4995	3.1620	0.4994	3.1320	0.4995
7	3.1493	0.4990	3.1518	0.4990	3.1368	0.4990
8	3.1454	0.4980	3.1467	0.4980	3.1392	0.4980

rate ≈ 0.5



◁ lin-log plot

Ex. 8.2.2.8 (Linearly convergent iteration)

$n = 1$: $x^{(k+1)} := x^{(k)} + \frac{\cos(x^{(k)}) + 1}{\sin(x^{(k)})}$, $k \in \mathbb{N}_0$

Definition 8.2.2.10. Order of convergence → , ,

A **convergent** sequence $\mathbf{x}^{(k)}$, $k = 0, 1, 2, \ldots$, in $\mathbb{R}^n$ with limit $\mathbf{x}^* \in \mathbb{R}^n$ converges with order p, if

$$\exists C > 0: \quad \left\| \mathbf{x}^{(k+1)} - \mathbf{x}^* \right\| \leq C \left\| \mathbf{x}^{(k)} - \mathbf{x}^* \right\|^p \quad \forall k \in \mathbb{N}_0 \,, \tag{8.2.2.11}$$

and, in addition, $C < 1$ in the case $p = 1$ (linear convergence → Def. 8.2.2.1).

$P \geq 1$

unknown !

Rem 8.2.2.12 (Detecting order $p>1$ of convergence)

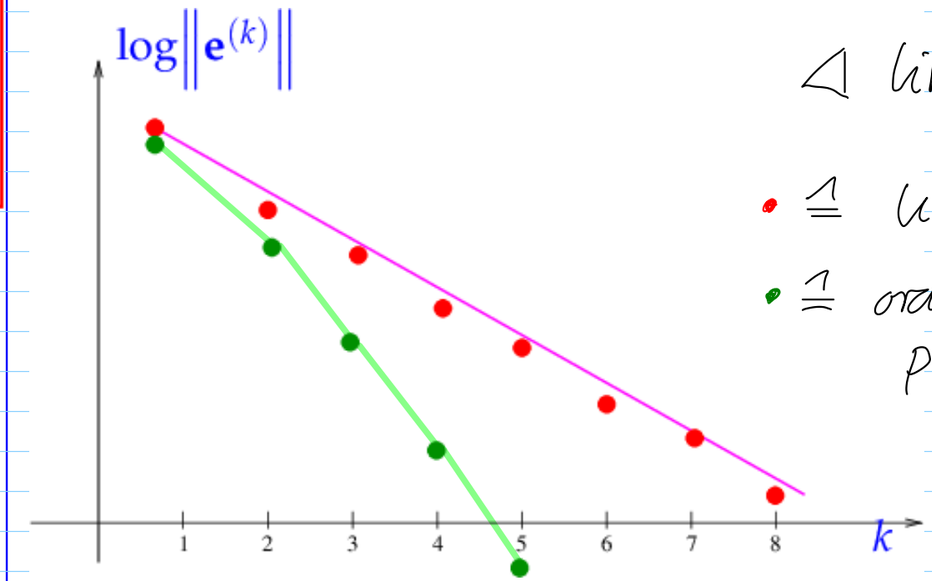
Assume sharpness : $\varepsilon_{k+1} \approx C \varepsilon_k^p$

$$\log \varepsilon_{k+1} \approx \log C + p \log \varepsilon_k$$
$$\log \varepsilon_k \approx \log C + p \log \varepsilon_{k-1}$$

$$\frac{\log \varepsilon_{k+1} - \log \varepsilon_k}{\log \varepsilon_k - \log \varepsilon_{k-1}} \approx p$$

↓

tabulate this



△ lin-log

- ≙ linear cvg.
- ≙ order-p cvg., $p > 1$

Ex. 8.2.2.13 (Order 2 = quadratic convergence)

$a > 0$:

$$x^{(k+1)} = \frac{1}{2}\left(x^{(k)} + \frac{a}{x^{(k)}}\right) \Rightarrow \underbrace{|x^{(k+1)} - \overbrace{\sqrt{a}}^{x^*}|}_{\varepsilon_{k+1}} = \frac{1}{2x^{(k)}} \underbrace{|x^{(k)} - \sqrt{a}|^2}_{\varepsilon_k^2} . \tag{8.2.2.14}$$

↓ 1-point iteration

↓ $\leq 1/2\sqrt{a}$, since $x^{(k)} \geq \sqrt{a}$

▷ $p = 2$, $C = \frac{1}{2\sqrt{a}}$ in Def. 8.2.2.10

④ $a = 2$:

k	$x^{(k)}$	$e^{(k)} := x^{(k)} - \sqrt{2}$	$\log \frac{\lvert e^{(k)}\rvert}{\lvert e^{(k-1)}\rvert} : \log \frac{\lvert e^{(k-1)}\rvert}{\lvert e^{(k-2)}\rvert}$
0	2.00000000000000000	0.58578643762690485	
1	1.50000000000000000	0.08578643762690485	
2	1.41666666666666652	0.00245310429357137	1.850
3	1.41421568627450966	0.00000212390141452	1.984
4	1.41421356237468987	0.00000000000159472	2.000
5	1.41421356237309492	0.00000000000000022	0.630

Note the doubling of the number of correct digits in each step ! [impact of roundoff !]

$\hat{=}$ relative error 10^{-m}

$$x^{(k)} = x^* (1 + \delta_k)$$

quad. cvg.:

$$|x^{(k+1)} - x^*| \approx C |x^{(k)} - x^*|^2$$

$$\Rightarrow \quad |\delta_{k+1} x^*| \approx C |\delta_k x^*|^2$$

$$\Rightarrow \quad \delta_{k+1} \approx \underbrace{C|x^*|}_{\text{a constant} \approx 1} \delta_k^2$$

▷ Review questions 8.2.16

ETH Lecture 401-0663-00L Numerical Methods for CSE

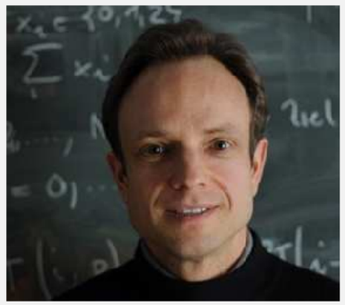
Course Video

Section 8.2.3: Iterative Methods: Termination Criteria/Stopping Rules

Prof. R. Hiptmair, SAM, ETH Zurich

Date: December 1, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 8.2.2

Duration: 25 minutes

Note: Possible minor *mismatch of video and tablet notes*!

[Corrections and updates can be incorporated into tablet notes only]

VIII Iterative Methods for Non-Linear Systems of Equations

8.2. Iterative Methods

8.2.3. Termination Criteria / Stopping Rules

Sequence $(\mathbf{x}^{(k)})_{k\in\mathbb{N}_0}$. Desired

$\left\|\mathbf{x}^{(K)} - \mathbf{x}^*\right\| \leq \tau_{\text{abs}}$, τ_{abs} $\hat{=}$ prescribed (absolute) tolerance.

or

$\left\|\mathbf{x}^{(K)} - \mathbf{x}^*\right\| \leq \tau_{\text{rel}}\|\mathbf{x}^*\|$, τ_{rel} $\hat{=}$ prescribed (relative) tolerance.

$\mathbf{x}^* \hat{=}$ solution, $\mathbf{F}(\mathbf{x}^*) = 0$

Ideal: Not accessible

STOP at step $K = \min\{k \in \mathbb{N}_0 : \left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\| \leq \begin{cases} \tau_{\text{abs}} \\ \textbf{or} \\ \tau_{\text{rel}}\|\mathbf{x}^*\| \end{cases}\}$ (8.2.3.3)

(2)

§ 8.2.3.4 Practical stopping rules

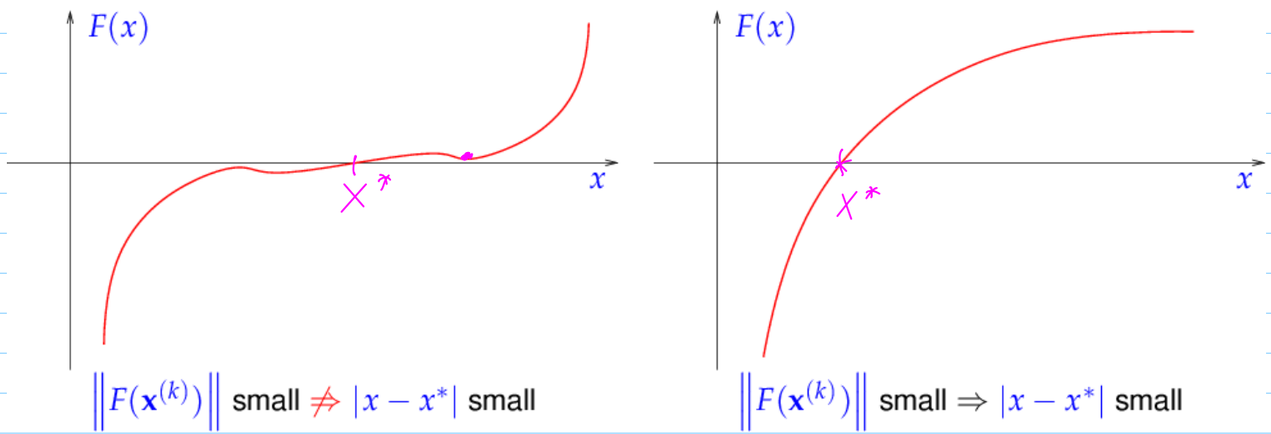
- A priori termination

→ STOP after a fixed no. of step

- Residual-based termination

STOP, when $\underbrace{\|F(x^{(k)})\|}_{\text{may tell little about } \|x^{(k)}-x^*\|} \leq \tau \triangleq$ prescribed tolerance

n = 1:



$\left\|F(\mathbf{x}^{(k)})\right\|$ small $\not\Rightarrow$ $|x-x^*|$ small

$\left\|F(\mathbf{x}^{(k)})\right\|$ small $\Rightarrow$ $|x-x^*|$ small

- Correction-based termination

STOP convergent iteration $\{\mathbf{x}^{(k)}\}_{k\in\mathbb{N}_0}$, when

$$\left\|\mathbf{x}^{(k+1)}-\mathbf{x}^{(k)}\right\| \leq \begin{cases} \tau_{\text{abs}} \\ \text{or} \\ \tau_{\text{rel}}\left\|\mathbf{x}^{(k+1)}\right\| \end{cases},$$

τ_{abs}, τ_{rel} prescribed absolute / relative tolerances > 0.

↳ correction

Order p-iteration, $p > 1$: $x^{(k+1)} \approx x^*$

Rem. 8.2.3.5 (STOP, when stationary in $\mathbb{M}$)

C++ code 8.2.3.6: Square root iteration → Ex. 8.2.2.13

```
2  double sqrtit(double a)
3  {
4    double x_old = -1;
5    double x = a;
6    while (x_old != x) {
7      x_old = x;
8      x = 0.5*(x+a/x);
9    }
10   return x;
11 }
```

◁ Exploit that $\mathbb{M}$ is finite

Guaranteed rel. err. $\approx$ EPS

§ 8.2.3.7 (A posteriori termination criterion for linearly convergent iterations)

$$\|x^{(k+1)} - x^*\| \leq L \|x^{(k)} - x^*\|, \quad L < 1$$

$$\|x^{(k)} \pm x^{(k+1)} - x^*\| \overset{\Delta\text{-ineq.}}{\leq} \|x^{(k+1)} - x^*\| + \|x^{(k+1)} - x^{(k)}\|$$

$$\leq L \|x^{(k)} - x^*\| + \|x^{(k+1)} - x^{(k)}\|$$

[$L < 1$]

$$(1-L)\|x^{(k)} - x^*\| \leq \|x^{(k+1)} - x^{(k)}\|$$

$$\|x^{(k)} - x^*\| \leq \frac{L}{1-L}\|x^{(k+1)} - x^{(k)}\|$$

$$\|x^{(k+1)} - x^*\| \leq \frac{L}{1-L}\|x^{(k+1)} - x^{(k)}\|$$

($\|x^{(k+1)} - x^{(k)}\|$: available)

→ Reliable error estimate, if only we know upper bound $\overline{L}$, $L \leq \overline{L} < 1$

Ex 8.2.3.9. (A posteriori error bound for linearly convergent iteration)

$n = 1$:

$$x^{(k+1)} = x^{(k)} + \frac{\cos x^{(k)} + 1}{\sin x^{(k)}} \Rightarrow x^{(k)} \to \pi \quad \text{for } x^{(0)} \text{ close to } \pi .$$

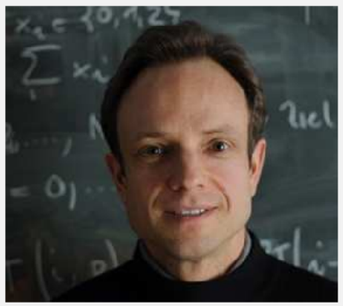
Observed: $L \approx 1/2$

k	$\lvert x^{(k)} - \pi \rvert$	$\frac{L}{1-L}\lvert x^{(k)} - x^{(k-1)} \rvert$	slack of bound
1	2.191562221997101	4.933154875586894	2.741592653589793
2	0.247139097781070	1.944423124216031	1.697284026434961
3	0.122936737876834	0.124202359904236	0.001265622027401
4	0.061390835206217	0.061545902670618	0.000155067464401
5	0.030685773472263	0.030705061733954	0.000019288261691
6	0.015341682696235	0.015344090776028	0.000002408079792
7	0.007670690889185	0.007670991807050	0.000000300917864
8	0.003835326638666	0.003835364250520	0.000000037611854
9	0.001917660968637	0.001917665670029	0.000000004701392
10	0.000958830190489	0.000958830778147	0.000000000587658
11	0.000479415058549	0.000479415131941	0.000000000073392
12	0.000239707524646	0.000239707533903	0.000000000009257
13	0.000119853761949	0.000119853762696	0.000000000000747
14	0.000059926881308	0.000059926880641	0.000000000000667
15	0.000029963440745	0.000029963440563	0.000000000000181

▷ Review questions 8.2.3.10

ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 8.3: Fixed-Point Iterations



Prof. R. Hiptmair, SAM, ETH Zurich

Date: December 1, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich

Prerequisites.
- Mean-value theorem
- Taylor expansion

Dependencies. Section 8.2.1, Section 8.2.2

Duration: 20 minutes

VIII Iterative Methods for Non-Linear Systems of Equations

8.3. Fixed-Point Iterations

NLSE: $F(x) = 0$, $F: D \subset \mathbb{R}^n \to \mathbb{R}^n$

Fixed-point iteration $\hat{=}$ 1-point iteration

A fixed point iteration is defined by iteration function $\Phi : U \subset \mathbb{R}^n \mapsto \mathbb{R}^n$:

iteration function $\Phi : U \subset \mathbb{R}^n \mapsto \mathbb{R}^n$
initial guess $\mathbf{x}^{(0)} \in U$

$\Rightarrow$ iterates $(\mathbf{x}^{(k)})_{k\in\mathbb{N}_0}$: $\mathbf{x}^{(k+1)} := \Phi(\mathbf{x}^{(k)})$.

$\to$ 1-point method, *cf.* (8.2.1.5)

Name:

Φ continuous $\Downarrow$

$$x^* = \lim_{k\to\infty} x^{(k)} \Longrightarrow \underbrace{x^* = \Phi(x^*)}$$

x^* is a fixed point of Φ

(2)

8.3.1. Consistent Fixed-Point Iterations

Definition 8.3.1.1. Consistency of fixed point iterations, *c.f.* Def. 8.2.1.7

A fixed point iteration $\mathbf{x}^{(k+1)} = \Phi(\mathbf{x}^{(k)})$ is consistent with $F(\mathbf{x}) = 0$, if, for $\mathbf{x} \in U \cap D$,

$$F(\mathbf{x}) = 0 \quad \Leftrightarrow \quad \Phi(\mathbf{x}) = \mathbf{x} \,.$$

General construction of fixed point iterations that is consistent with $F(\mathbf{x}) = 0$:

1. Rewrite equivalently $F(\mathbf{x}) = 0 \quad \Leftrightarrow \quad \Phi(\mathbf{x}) = \mathbf{x}$ and then
2. use the fixed point iteration

$$\mathbf{x}^{(k+1)} := \Phi(\mathbf{x}^{(k)}) \,. \tag{8.3.1.2}$$

→ Many possibilities !

Exp. 8.3.1.3 (Consistent fixed-point iterations)

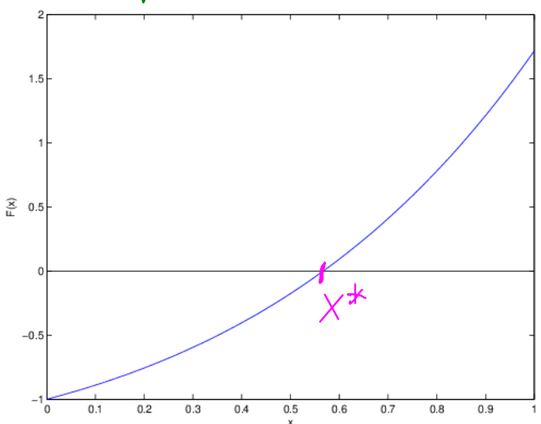
$n = 1$:

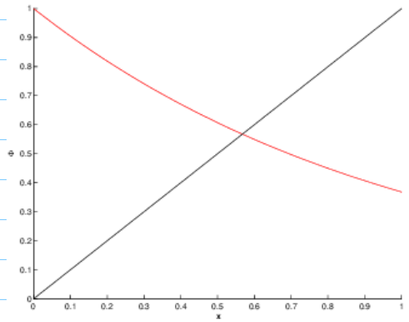
$$F(x) = xe^x - 1 \,, \quad x \in [0,1] \,.$$

Different fixed-point forms:

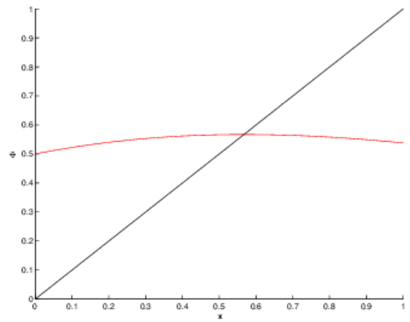
$$\begin{aligned} \Phi_1(x) &= e^{-x} \,, \\ \Phi_2(x) &= \frac{1+x}{1+e^x} \,, \\ \Phi_3(x) &= x + 1 - xe^x \,. \end{aligned}$$

$\Phi_i(x) = x \;\Leftrightarrow\; F(x) = 0$

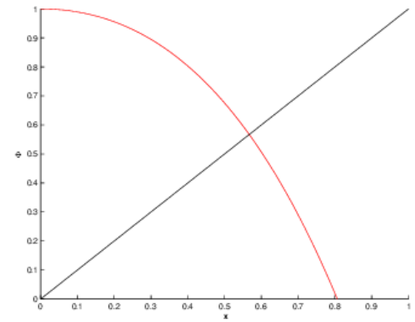




function Φ_1



function Φ_2



function Φ_3

k	$\lvert x_1^{(k+1)} - x^* \rvert$	$\lvert x_2^{(k+1)} - x^* \rvert$	$\lvert x_3^{(k+1)} - x^* \rvert$
0	0.067143290409784	0.067143290409784	0.067143290409784
1	0.039387369302849	0.000832287212566	0.108496074240152
2	0.021904078517179	0.000000125374922	0.219330611898582
3	0.012559804468284	0.000000000000003	0.288178118764323
4	0.007078662470882	0.000000000000000	0.723649245792953
5	0.004028858567431	0.000000000000000	0.410183132337935
6	0.002280343429460	0.000000000000000	1.186907542305364
7	0.001294757160282	0.000000000000000	0.146569797006362
8	0.000733837662863	0.000000000000000	0.310516641279937
9	0.000416343852458	0.000000000000000	0.357777386500765
10	0.000236077474313	0.000000000000000	0.974565695952037

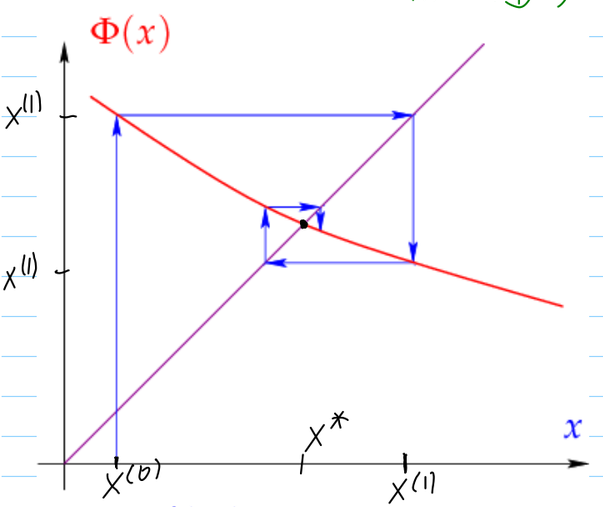
↓ linear cvg. ↓ quadratic cvg. ↓ no cvg.

8.3.2. Convergence of Fixed-Point Iterations

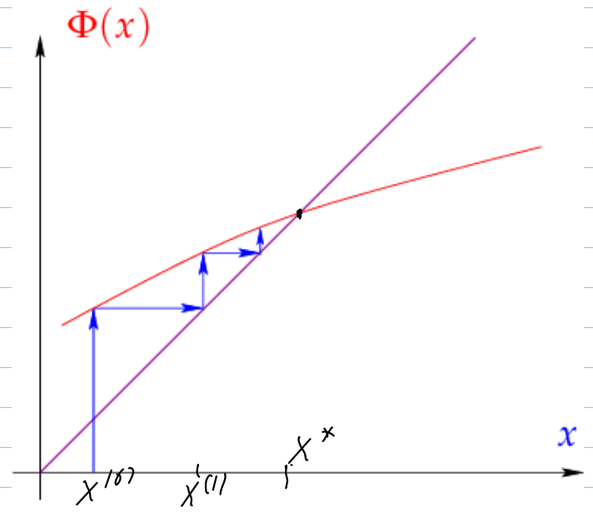
NumCSE@ETHZ

▷ $\Phi'(x^*)$ is crucial ↔ slope G_Φ

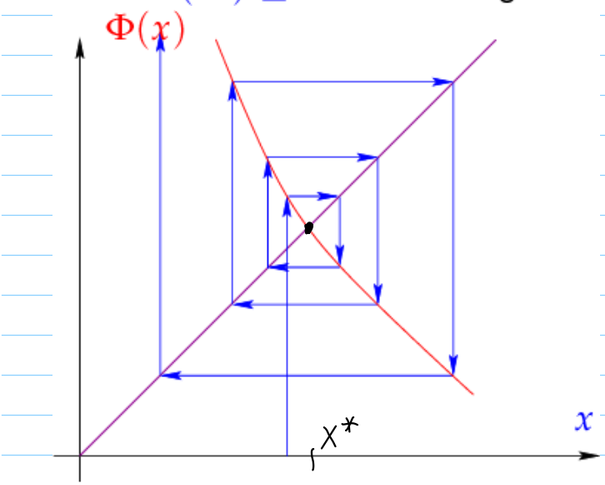
Rem 8.3.2.2 : (Visualization of fixed-point iterations in 1D)



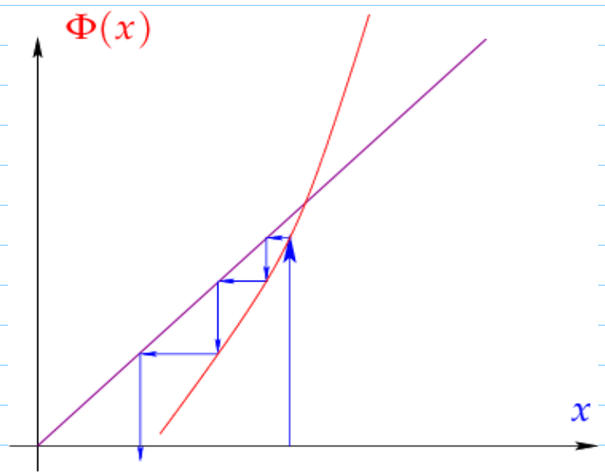
$-1 < \Phi'(x^*) \leq 0$ ➢ convergence



$0 \leq \Phi'(x^*) < 1$ ➢ convergence

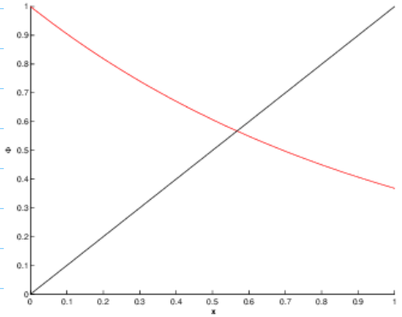


$\Phi'(x^*) < -1$ ➢ divergence

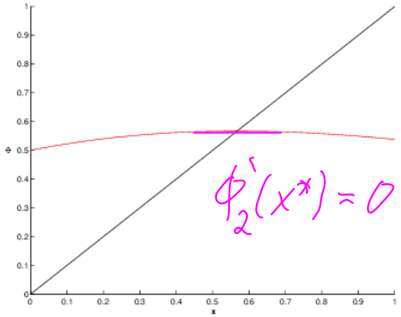


$1 < \Phi'(x^*)$ ➢ divergence

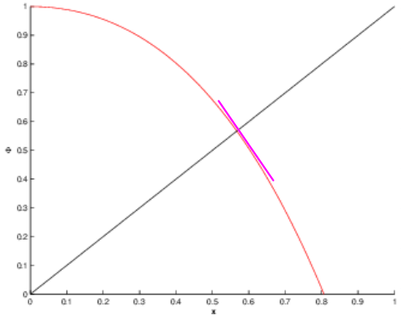
Exp 8.3.1.3 :



Φ_1: linear convergence ?



Φ_2: quadratic convergence ?



function Φ_3: no convergence

Lemma 8.3.2.7. Sufficient condition for local linear convergence of fixed point iteration →

If $\Phi : U \subset \mathbb{R}^n \mapsto \mathbb{R}^n$, $\Phi(\mathbf{x}^*) = \mathbf{x}^*$, Φ differentiable in $\mathbf{x}^*$, and $\|D\Phi(\mathbf{x}^*)\| < 1$, then the fixed point iteration

$$\mathbf{x}^{(k+1)} := \Phi(\mathbf{x}^{(k)}) \,, \tag{8.3.1.2}$$

converges locally and at least linearly that is (matrix norm, Def. 1.5.5.10 !)

$$\exists 0 \leq L < 1: \quad \left\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\right\| \leq L \left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\| \quad \forall k \in \mathbb{N}_0 \,,$$

provided that the initial guess $\mathbf{x}^{(0)}$ belongs to some neighborhood of $\mathbf{x}^*$.

$x^{(0)}$ "sufficiently close" to x^*

(4)

$$\|x^{(k+1)} - x^*\| = \|\phi(x^{(k)}) - \phi(x^*)\|$$
$$= \|D\phi(x^*)(x^{(k)} - x^*) + O(\|x^{(k)} - x^*\|^2)\|$$

▷ Review questions 8.3.2.21

Lemma 8.3.2.15. Higher order local convergence of fixed point iterations

If $\Phi : U \subset \mathbb{R} \mapsto \mathbb{R}$ is $m+1$ times continuously differentiable, $\Phi(x^*) = x^*$ for some x^* in the interior of U, and $\Phi^{(l)}(x^*) = 0$ for $l = 1, \ldots, m$, $m \geq 1$, then the fixed point iteration (8.3.1.2) *converges locally to x^* with order $\geq m+1$ (→ Def. 8.2.2.10).*

$\phi'(x^*) = 0 \Rightarrow$ At least quadratic cvg.

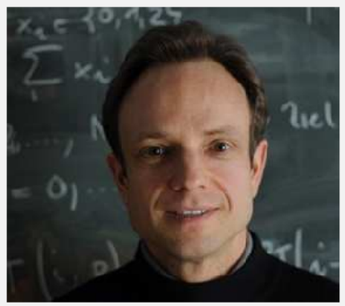
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 8.4.1: Finding Zeros of Scalar Functions: Bisection

Prof. R. Hiptmair, SAM, ETH Zurich

Date: December 1, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 8.2.1, Section 8.2.2

Duration: 10 minutes

VIII Iterative Methods for Non-Linear Systems of Equations

8.4. Finding Zeros of Scalar Functions

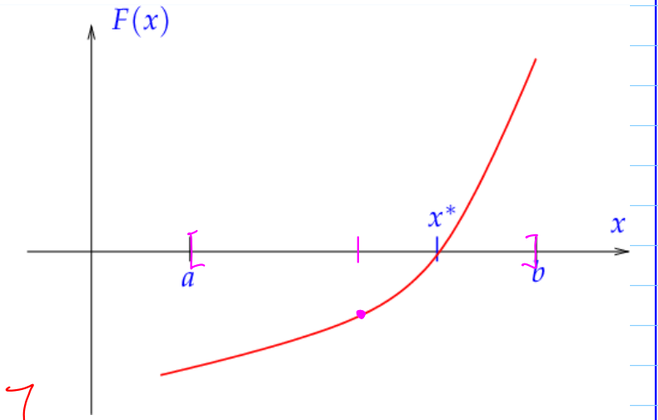
Now: $n = 1$

$$F : I \subset \mathbb{R} \longrightarrow \mathbb{R}$$

Find: $x^* \in \mathbb{R}$: $F(x^*) = 0$

8.4.1 Bisection

Assume : F continuous



Input: $a, b \in I$ such that

$F(a)F(b) < 0$. (different signs!)

▶ $\exists\, x^* \in]\min\{a,b\}, \max\{a,b\}[:$
$F(x^*) = 0$,

[intermediate value theorem]

C++ code 8.4.1.2: Bisection method for solving $F(x) = 0$ on $[a,b]$

```
// Searching zero of F in [a,b] by bisection
template <typename Func, typename Scalar>
Scalar bisect(Func&& F, Scalar a, Scalar b, Scalar tol)
{
  if (a > b) std::swap(a,b); // sort interval bounds
  if (F(a)*F(b) > 0) throw "f(a) and f(b) have same sign";
  static_assert(std::is_floating_point<Scalar>::value,
                "Scalar must be a floating point type");
  int v=F(a) < 0 ? 1 : -1;
  Scalar x = (a+b)/2; // determine midpoint
  // termination, relies on machine arithmetic if tol = 0
  while (b-a > tol) { //
    assert(a<=x && x<=b); // assert invariant
    // sgn(f(x)) = sgn(f(b)), then use x as next right boundary
    if (v*F(x) > 0) b=x;
    // sgn(f(x)) = sgn(f(a)), then use x as next left boundary
    else a=x;
    x = (a+b)/2; // determine next midpoint
  }
  return x;
}
```

Idea : Enclose x^* in geometrically decreasing intervals $[a^{(k)}, b^{(k)}]$ on which f changes sign : $f(a^{(k)})f(b^{(k)}) \leq 0$

$$x^{(k)} := a^{(k)}$$

"Linear-type" convergence, rate = ½

$$|b^{(k)} - a^{(k)}| \leq 2^{-k}|b-a|$$

$$\Rightarrow |x^{(k)} - x^*| \leq 2^{-k}|b-a|$$

+ Guaranteed convergence : robust method
+ F-evaluations only : "derivative-free"
+ Work $\forall$ continuous F

NumCSE@ETHZ

▷ Review questions 8.4.1.4

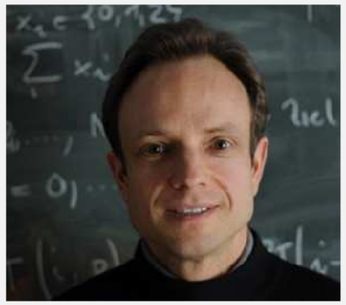
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 8.4.2.1: Newton Method in the Scalar Case

Prof. R. Hiptmair, SAM, ETH Zurich

Date: December 1, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.
- Differentiation: product rule and chain rule

Dependencies. Section 8.2.1, Section 8.2.2

Duration: 36 minutes

VIII Iterative Methods for Non-Linear Systems of Equations

8.4. Finding Zeros of Scalar Functions

Find $x^* \in I : f(x^*) = 0$, $f : I \subset \mathbb{R} \to \mathbb{R}$

8.4.2. Model Function Methods

Idea: Given recent iterates (approximate zeroes) $x^{(k)}, x^{(k-1)}, \ldots, x^{(k-m+1)}$, $m \in \mathbb{N}$

❶ replace F with a k-dependent model function $\widetilde{F}_k$

(based on function values $F(x^{(k)}), F(x^{(k-1)}), \ldots, F(x^{(k-m+1)})$ and, possibly, derivative values $F'(x^{(k)}), F'(x^{(k-1)}), \ldots, F'(x^{(k-m+1)})$)

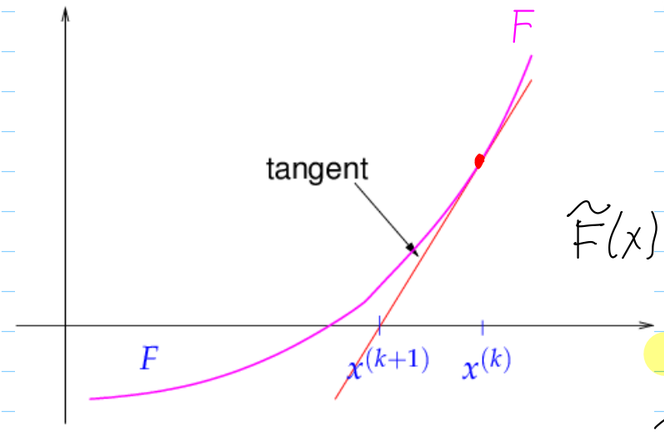
❷ $x^{(k+1)} :=$ zero of $\widetilde{F}_k$: $\widetilde{F}_k(x^{(k+1)}) = 0$

(has to be readily available $\leftrightarrow$ analytic formula)

Locally around $x^{(k)}$: $F(x) \approx F_k(x)$

(2)

8.4.2.1. Newton Method in the Scalar Case



Assume: $f \in C^1$

Model function $\tilde{F}$

$\hat{=}$ tangent at G_f in $(x^{(k)}, F(x^{(k)}))$

$$\tilde{F}(x) = F(x^{(k)}) + F'(x^{(k)})(x - x^{(k)})$$

$$x^{(k+1)} = x^{(k)} - \frac{F(x^{(k)})}{F'(x^{(k)})}$$

$\hat{=}$ Newton iteration

Newton correction

$F'(x^{(k)}) \neq 0$ required

C++11-code 8.4.2.2: Newton method in the scalar case $n = 1$

```
template <typename FuncType,typename DervType,typename Scalar>
Scalar newton1D(FuncType &&F, DervType &&DF,
const Scalar &x0, double rtol, double atol)
{                    ↳ initial guess
  Scalar s, x = x0;
  do {
    s = F(x)/DF(x);  // compute Newton correction
    x -= s;          // compute next iterate
  }
  // correction based termination (relative and absolute)
  while ((std::abs(s) > rtol*std::abs(x)) && (std::abs(s) > atol));
  return (x);
}
```

Ex. 8.4.2.3 (Square-root iteration as a Newton iteration)

$F(x) = x^2 - a$, $a > 0$

$F'(x) = 2x$

$$\triangleright \quad x^{(k+1)} = x^{(k)} - \frac{(x^{(k)})^2 - a}{2x^{(k)}} = \frac{1}{2}\left(x^{(k)} + \frac{a}{x^{(k)}}\right),$$

Observed: quadratic cvg.

§ 8.4.2.5 (Convergence of Newton's method in 1D)

Newton iteration as fixed-point iteration

$$x^{(k+1)} = x^{(k)} - \frac{F(x^{(k)})}{F'(x^{(k)})} = \Phi(x^{(k)}), \quad \Phi(x) = x - \frac{F(x)}{F'(x)}$$

Lemma 8.3.2.15. Higher order local convergence of fixed point iterations

If $\Phi : U \subset \mathbb{R} \mapsto \mathbb{R}$ is $m+1$ times continuously differentiable, $\Phi(x^*) = x^*$ for some x^* in the interior of U, and $\Phi^{(l)}(x^*) = 0$ for $l = 1, \ldots, m$, $m \geq 1$, then the fixed point iteration (8.3.1.2) *converges locally to x^* with order $\geq m+1$ ($\rightarrow$ Def. 8.2.2.10).*

③

$$\phi'(x) = 1 - \frac{F'(x)^2 - F(x)F''(x)}{F'(x)^2} = \frac{F(x)F''(x)}{F'(x)^2}$$

$$\left.\begin{array}{l} F(x^*) = 0 \\ F'(x^*) \neq 0 \end{array}\right\} \Rightarrow \quad \phi'(x^*) = 0$$

- $|x^{(k+1)} - x^*| = |\phi(x^{(k)}) - \phi(x^*)|$

Taylor: $f \in C^2$ $= |\phi'(x^*)(x^{(k)} - x^*) + O(|x^{(k)} - x^*|^2)|$

Convergence of Newton's method in 1D

Newton's method locally quadratically converges (→ Def. 8.2.2.10) to a zero x^* of F, if $F'(x^*) \neq 0$

Ex. 8.4.2.4: (Newton method in 1D)

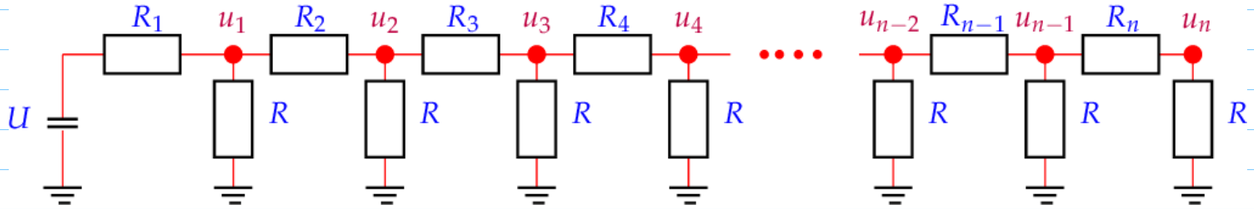
$$F(x) = xe^x - 1 \Rightarrow \quad F'(x) = e^x(1+x) \quad \Rightarrow \quad x^{(k+1)} = x^{(k)} - \frac{x^{(k)}e^{x^{(k)}} - 1}{e^{x^{(k)}}(1+x^{(k)})} = \frac{(x^{(k)})^2 + e^{-x^{(k)}}}{1+x^{(k)}}$$

$$F(x) = x - e^{-x} \quad \Rightarrow \quad F'(x) = 1 + e^{-x} \quad \Rightarrow \quad x^{(k+1)} = x^{(k)} - \frac{x^{(k)} - e^{-x^{(k)}}}{1+e^{-x^{(k)}}} = \frac{1+x^{(k)}}{1+e^{x^{(k)}}}\,.$$

same x^* — different Newton iterations

Ex 8.4.2.9. (Implicit differentiation of F)

Linear circuit:



Find R such that potential $u_j = 1$, $j \in \{1,..,n\}$?

Nodal analysis:

$$\text{Node 1:} \quad \frac{1}{R_1}(u_1 - U) + \frac{1}{R}u_1 + \frac{1}{R_2}(u_1 - u_2) = 0\,,$$

$$\text{Node } j\text{:} \quad \frac{1}{R_{j-1}}(u_j - u_{j-1}) + \frac{1}{R}u_j + \frac{1}{R_j}(u_j - u_{j+1}) = 0\,, \quad j = 2,\ldots,n-1\,, \tag{8.4.2.10}$$

$$\text{Node } n\text{:} \quad \frac{1}{R_n}(u_n - u_{n-1}) + \frac{1}{R}u_n = 0\,.$$

$$\left(\mathbf{A} + \frac{1}{R}\cdot\mathbf{I}\right)\mathbf{u} = \mathbf{b}\,, \tag{8.4.2.11}$$

$$\mathbf{A} = \begin{bmatrix} \frac{1}{R_1}+\frac{1}{R_2} & -\frac{1}{R_2} & & & & \\ -\frac{1}{R_2} & \frac{1}{R_2}+\frac{1}{R_3} & -\frac{1}{R_3} & & & \\ & -\frac{1}{R_3} & \frac{1}{R_3}+\frac{1}{R_4} & -\frac{1}{R_4} & & \\ & & \ddots & \ddots & \ddots & \\ & & & -\frac{1}{R_{n-2}} & \frac{1}{R_{n-2}}+\frac{1}{R_{n-1}} & -\frac{1}{R_{n-1}} \\ & & & & -\frac{1}{R_{n-1}} & \frac{1}{R_n}+\frac{1}{R_{n-1}} \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} \frac{U}{R_1} \\ 0 \\ \vdots \\ \vdots \\ 0 \end{bmatrix}.$$

④

Find $x := 1/R > 0$:

$$F(x) = 0 \quad \text{with} \quad F : \begin{cases} \mathbb{R} & \to & \mathbb{R} \\ x & \mapsto & \mathbf{w}^T (\mathbf{A} + x\mathbf{I})^{-1}\mathbf{b} - 1 \end{cases}, \tag{8.4.2.12}$$

$\mathbf{b} = \vec{e}_j$ (coordinate vector); $\mathbf{w}^T(\mathbf{A}+x\mathbf{I})^{-1}\mathbf{b}$: $\underline{u}$

$$\begin{cases} F(x) := \underline{w}^T \underline{u}(x) - 1 \\ (A + xI)\,\underline{u}(x) = \underline{b} \end{cases}, \quad x > 0$$

$F'(x)$? Trick : Implicit differentiation

$\frac{d}{dx}$: $(A + xI)\cdot \underline{u}(x) = \underline{b}$

product rule $I \cdot \underline{u}(x) + (A + xI)\,\underline{u}'(x) = 0$

$\triangleright$ $\underline{u}'(x) = -(A + xI)^{-1}\underline{u}(x)$

$\triangleright$ $F'(x) = w^T \underline{u}'(x)$

$\triangleright$ Newton iteration

2 LSE with tridiagonal matrices

$$x^{(k+1)} = x^{(k)} + F'(x^{(k)})^{-1} F(x^{(k)}) = x^{(k)} + \frac{\mathbf{w}^T\mathbf{u}(x^{(k)}) - 1}{\mathbf{w}^T\mathbf{z}}, \tag{8.4.2.14}$$

$$(\mathbf{A} + x^{(k)}\mathbf{I})\mathbf{u}(x^{(k)}) = \mathbf{b}, \quad (\mathbf{A} + x^{(k)}\mathbf{I})\mathbf{z} = \mathbf{u}(x^{(k)}).$$

§ 8.4.2.18 (Preconditioning of Newton's method)

If $F \approx \hat{F} \Rightarrow g(x) := \hat{F}^{-1}(F(x)) \approx x$

↑ Assume : $\hat{F}^{-1}$ can be computed easily

$F(x) = 0 \iff G(x) := g(x) - \hat{F}^{-1}(0) = 0$

G is "almost linear" : expect large domain of quadratic convergence

$$(\hat{F}^{-1})' = \frac{1}{\hat{F}' \circ \hat{F}^{-1}} \Rightarrow g'(x) = \frac{F'(x)}{\hat{F}'(g(x))}$$

$\triangleright$ Newton it. for G :

$$x^{(k+1)} = x^{(k)} - \frac{G(x^{(k)})\hat{F}'(g(x^{(k)}))}{F'(x^{(k)})}, \tag{8.4.2.19}$$

$g(x^{(k)}) = \hat{F}^{-1}(F(x^{(k)}))$, $G(x^{(k)}) = g(x^{(k)}) - y_0$, $y_0 := \hat{F}^{-1}(0)$.

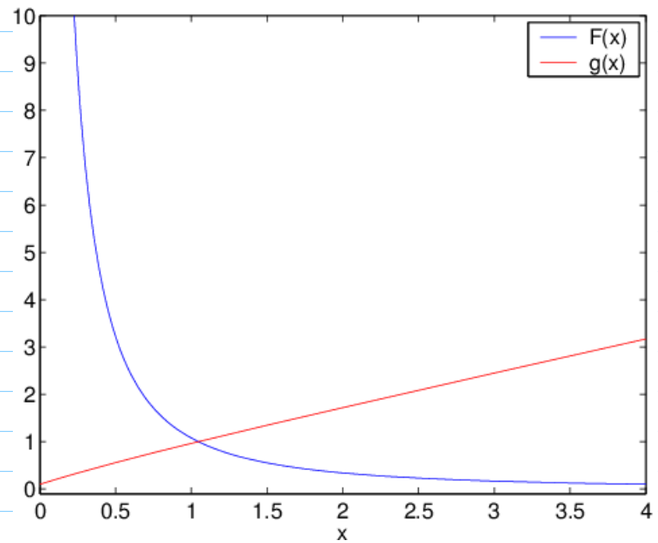
(5)

Ex. 8.4.2.20 (Preconditioned Newton method)

$$F(x) = \frac{1}{(x+1)^2} + \frac{1}{(x+0.1)^2} - 1, \quad x > 0$$

$$F(x) \approx \hat{F}(x) = \frac{1}{x^2} - 1$$

$$\hat{F}^{-1}(y) = \frac{1}{\sqrt{y+1}}$$



Convergence recorded for $x^{(0)} = 0$:

k	$x^{(k)}$	$F(x^{(k)})$	$x^{(k)} - x^{(k-1)}$	$x^{(k)} - x^*$
1	0.91312431341979	0.24747993091128	0.91312431341979	-0.13307818147469
2	1.04517022155323	0.00161402574513	0.13204590813344	-0.00103227334125
3	1.04620244004116	0.00000008565847	0.00103221848793	-0.00000005485332
4	1.04620249489448	0.00000000000000	0.00000005485332	-0.00000000000000

▷ Review questions 8.4.2.16

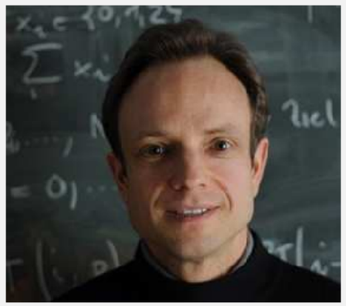
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 8.4.2.3: Multi-Point Methods

Prof. R. Hiptmair, SAM, ETH Zurich

Date: December 2, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.
- Inverse functions in 1D

Dependencies. Section 8.2.1, Section 8.2.2

Duration: 23 minutes

VIII Iterative Methods for Non-Linear Systems of Equations

8.4. Finding Zeros of Scalar Functions

Find $x^* \in I$: $F(x^*) = 0$, $F : I \subset \mathbb{R} \to \mathbb{R}$

8.4.2. Model Function Methods

Idea: Given recent iterates (approximate zeroes) $x^{(k)}, x^{(k-1)}, \ldots, x^{(k-m+1)}$, $m \in \mathbb{N}$

❶ replace F with a k-dependent model function $\widetilde{F}_k$

(based on function values $F(x^{(k)}), F(x^{(k-1)}), \ldots, F(x^{(k-m+1)})$ and, possibly, derivative values $F'(x^{(k)}), F'(x^{(k-1)}), \ldots, F'(x^{(k-m+1)})$)

❷ $x^{(k+1)} :=$ zero of $\widetilde{F}_k$: $\widetilde{F}_k(x^{(k+1)}) = 0$

(has to be readily available $\leftrightarrow$ analytic formula)

(2)

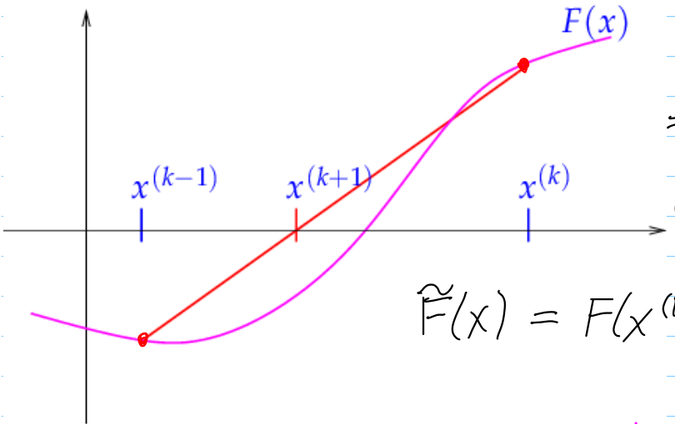
8.4.2.3 Multi-Point Methods

Construction of multi-point iterations in 1D

Idea: Replace F with interpolating polynomial

producing interpolatory model function methods

§ 8.4.2.28 (The secant method)



Model function $\widetilde{F}$ = line connecting $(x^{(k-1)}, F(x^{(k-1)}))$, $(x^{(k)}, F(x^{(k)}))$

$$\widetilde{F}(x) = F(x^{(k)}) + \frac{F(x^{(k)}) - F(x^{(k-1)})}{x^{(k)} - x^{(k-1)}}(x - x^{(k)})$$

$$x^{(k+1)} = x^{(k)} - \left[\frac{F(x^{(k)}) - F(x^{(k-1)})}{x^{(k)} - x^{(k-1)}}\right]^{-1} F(x^{(k)})$$

Newton: $x^{(k+1)} = x^{(k)} - F'(x^{(k)})^{-1} F(x^{(k)})$

replaces $F'(x^{(k)})$ in Newton if.

secant meth. = 2-point method

⇒ Two initial guesses $x^{(0)}, x^{(1)}$ required

C++ code 8.4.2.31: Secant method for 1D non-linear equaton

```
2  // Secand method for solving F(x) = 0 for F : D ⊂ ℝ → ℝ,
3  // initial guesses x_0, x_1,
4  // tolerances atol (absolute), rtol (relative)
5  template <typename Func>
6  double secant(double x0, double x1, Func &&F, double rtol, double atol,
7                unsigned int maxIt) {
8    double fo = F(x0);
9    for (unsigned int i = 0; i < maxIt; ++i) {
10     double fn = F(x1);
11     double s = fn * (x1 - x0) / (fn - fo); // secant correction
12     x0 = x1;
13     x1 = x1 - s;
14     // correction based termination (relative and absolute)
15     if (abs(s) < max(atol, rtol * min(abs(x0), abs(x1))))
16       return x1;
17     fo = fn;
18   }
19   return x1;
20 }
```

\+ derivative-free

\+ 1 F-evaluation per step

Exp. 8.4.2.32 (Convergence of secant method)

$F(x) = xe^x - 1$, $x^{(0)} = 0$, $x^{(1)} = 5$

≈ order of cvg.

$F(x) = xe^x - 1$, using secant method with initial guesses $x^{(0)} = 0$, $x^{(1)} = 5$.

k	$x^{(k)}$	$F(x^{(k)})$	$e^{(k)} := x^{(k)} - x^*$	$\frac{\log\lvert e^{(k+1)}\rvert - \log\lvert e^{(k)}\rvert}{\log\lvert e^{(k)}\rvert - \log\lvert e^{(k-1)}\rvert}$
2	0.00673794699909	-0.99321649977589	-0.56040534341070	
3	0.01342122983571	-0.98639742654892	-0.55372206057408	24.43308649757745
4	0.98017620833821	1.61209684919288	0.41303291792843	2.70802321457994
5	0.38040476787948	-0.44351476841567	-0.18673852253030	1.48753625853887
6	0.50981028847430	-0.15117846201565	-0.05733300193548	1.51452723840131
7	0.57673091089295	0.02670169957932	0.00958762048317	1.70075240166256
8	0.56668541543431	-0.00126473620459	-0.00045787497547	1.59458505614449
9	0.56713970649585	-0.00000990312376	-0.00000358391394	1.62641838319117
10	0.56714329175406	0.00000000371452	0.00000000134427	
11	0.56714329040978	-0.00000000000001	-0.00000000000000	

▷ Fractional order of cvg, $p \approx 1.62$

§ 8.4.2.39 (Inverse Interpolation)

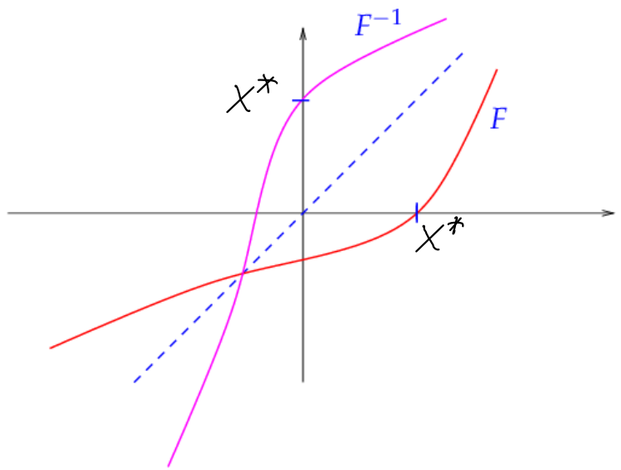
Assume: $F : I \subset \mathbb{R} \mapsto \mathbb{R}$ one-to-one (monotone)

$$F(x^*) = 0 \quad \Rightarrow \quad F^{-1}(0) = x^* .$$

- Interpolate F^{-1} by polynomial p of degree $m-1$ determined by
$$p(F(x^{(k-j)})) = x^{(k-j)} , \quad j = 0, \ldots, m-1 .$$
- New approximate zero $x^{(k+1)} := p(0)$ → m-point method

all different, if $x^{(k-j)}$ all different

The graph of F^{-1} can be obtained by reflecting the graph of F at the angular bisector. ▷



$$F(x^*) = 0 \quad \Leftrightarrow \quad F^{-1}(0) = x^*$$

$m = 2$ ⇒ secant method

New method for $m = 3$:

quadratic inverse interpolation

$$x^{(k+1)} = \frac{F_0^2(F_1x_2 - F_2x_1) + F_1^2(F_2x_0 - F_0x_2) + F_2^2(F_0x_1 - F_1x_0)}{F_0^2(F_1 - F_2) + F_1^2(F_2 - F_0) + F_2^2(F_0 - F_1)} .$$

$(F_0 := F(x^{(k-2)}), F_1 := F(x^{(k-1)}), F_2 := F(x^{(k)}), x_0 := x^{(k-2)}, x_1 := x^{(k-1)}, x_2 := x^{(k)})$

Exp. 8.4.2.40 (Convergence of quadratic inverse interpolation)

$F(x) = xe^x - 1$, $x^{(0)} = 0$, $x^{(1)} = 2.5$, $x^{(2)} = 5$. → 3 initial guesses for 3-point method

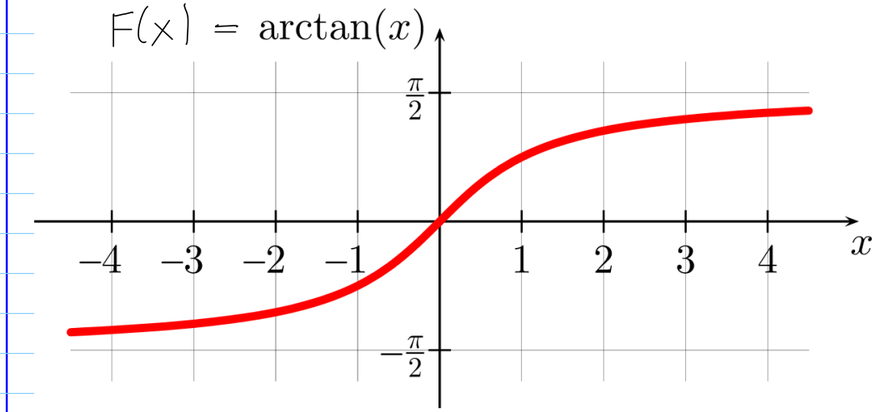
k	$x^{(k)}$	$F(x^{(k)})$	$e^{(k)} := x^{(k)} - x^*$	$\frac{\log\lvert e^{(k+1)}\rvert - \log\lvert e^{(k)}\rvert}{\log\lvert e^{(k)}\rvert - \log\lvert e^{(k-1)}\rvert}$
3	0.08520390058175	-0.90721814294134	-0.48193938982803	
4	0.16009252622586	-0.81211229637354	-0.40705076418392	3.33791154378839
5	0.79879381816390	0.77560534067946	0.23165052775411	2.28740488912208
6	0.63094636752843	0.18579323999999	0.06380307711864	1.82494667289715
7	0.56107750991028	-0.01667806436181	-0.00606578049951	1.87323264214217
8	0.56706941033107	-0.00020413476766	-0.00007388007872	1.79832936980454
9	0.56714331707092	0.00000007367067	0.00000002666114	1.84841261527097
10	0.56714329040980	0.00000000000003	0.00000000000001	

▷ Again: fractional order ≈ 1.8

NumCSE@ETHZ

A warning: Local cvg. only!

Ex 8.4.2.38 (Local convergence of the secant method)

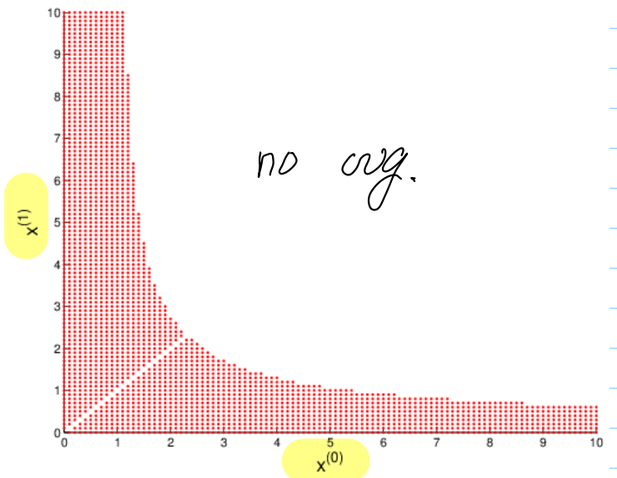


Model problem: find zero of

$$F(x) = \arctan(x)$$

· ≙ secant method converges for a pair $(x^{(0)}, x^{(1)}) \in \mathbb{R}^2_+$ of initial guesses. ▷

We observe that the secant method will converge only for initial guesses sufficiently close to 0 = local convergence → Def. 8.2.1.10



no cvg.

NumCSE@ETHZ

▷ Review questions 8.4.2.41

ETH Lecture 401-0663-00L Numerical Methods for CSE

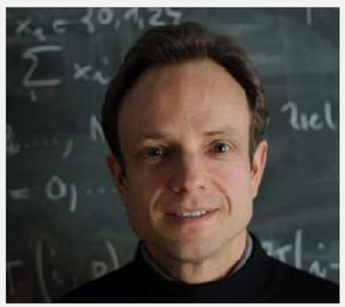
Course Video

Section 8.4.3: Asymptotic Efficiency of Iterative Methods for Zero Finding

Prof. R. Hiptmair, SAM, ETH Zurich

Date: December 2, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 8.2.1, Section 8.2.2, Section 8.4.2.3

Duration: 17 minutes

VIII Iterative Methods for Non-Linear Systems of Equations

8.4. Finding Zeros of Scalar Functions

Seek $x^* \in I$: $F(x^*) = 0$, $F : I \subset \mathbb{R} \to \mathbb{R}$

8.4.3. Asymptotic Efficiency of Iterative Methods for Zero Finding

$$\text{Efficiency} = \frac{\text{gain}}{\text{work}} \quad \begin{array}{l} \leftarrow \text{no. of correct digits} \\ \leftarrow \#F\text{-eval} + \#F'\text{-eval} \end{array}$$

For an iterative method ($e^{(k)} \hat{=}$ iteration error)

Gain : $|e^{(k)}| \leq \rho |e^{(0)}|$ for $\rho \ll 1$ (*)

work $\leftrightarrow$ minimal no. of steps it takes to achieve (*) $\hat{=} k(\rho)$

(2)

Notice: $|\log\rho|$ $\leftrightarrow$ Gain in no. of significant digits of $x^{(k)}$

Measure for efficiency:

$$\text{Efficiency} := \frac{\text{no. of digits gained}}{\text{total work required}} = \frac{|\log\rho|}{k(\rho)\cdot W}$$

$W \hat{=}$ # F-eval + # F'-eval per step

Focus: order-p iteration, $p > 1$

$$\left|e^{(k)}\right| \leq C\left|e^{(k-1)}\right|^p \leq C^{1+p}\left|e^{(k-2)}\right|^{p^2} \leq C^{1+p+p^2}\left|e^{(k-3)}\right|^{p^3} \leq \ldots$$

$$\leq C^{1+p^2+p^3+\cdots+p^{k-1}}\left|e^{(0)}\right|^{p^k} = C^{\frac{p^k-1}{p-1}}\left|e^{(0)}\right|^{p^k-1}\left|e^{(0)}\right|, \quad k \in \mathbb{N},$$

Assume < 1

$$|e^{(k)}| \leq \rho\,|e^{(0)}| \text{ , if } \left(C^{\frac{1}{1-p}}|e^{(0)}|\right)^{p^k-1} \leq \rho$$

$$k(\rho) \geq \frac{\log(1 + \frac{\log\rho}{\log L_0})}{\log p}, \quad L_0 := C^{1/p-1}\left|e^{(0)}\right| < 1 \tag{8.4.3.10}$$

§ 8.4.3.11 (Asymptotic efficiency)

$\rho \ll 1 \Rightarrow |\log\rho| \gg 1$

asymptotic efficiency for $\rho \ll 1$ (➜ $|\log\rho| \to \infty$):

$$\text{Efficiency}_{|\rho\ll 1} = \begin{cases} -\frac{\log C}{W} & \text{, if } p = 1\,, \\ \frac{\log p}{W}\cdot\frac{|\log\rho|}{\log(|\log\rho|)} & \text{, if } p > 1\,. \end{cases} \tag{8.4.3.12}$$

specific for the iteration: gauge for asymptotic efficience

§ 8.4.3.14 (Comparison of Newton's method and of the secant method)

$W_{\text{newton}} = 1$ F-eval + 1 F'-eval

$W_{\text{secant}} = 1$ F-eval

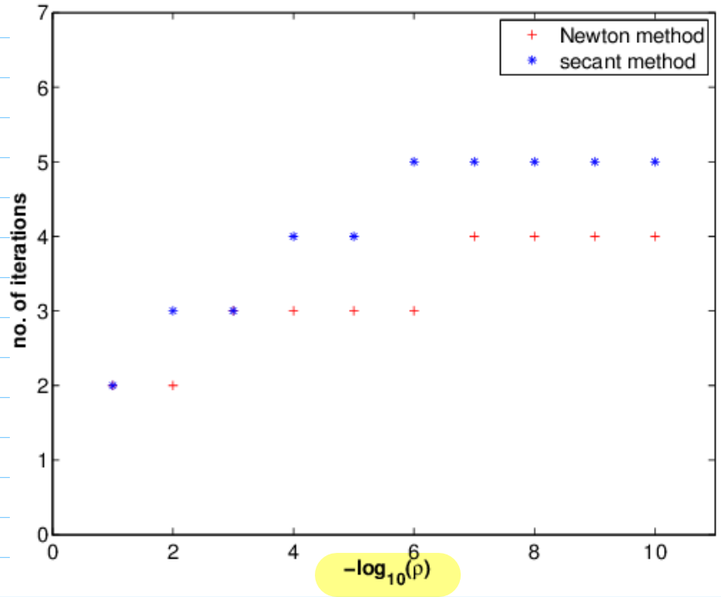
$W_{\text{Newton}} = 2W_{\text{secant}}$,

$p_{\text{Newton}} = 2$, $p_{\text{secant}} = 1.62$

$$\Rightarrow \frac{\log p_{\text{Newton}}}{W_{\text{Newton}}} : \frac{\log p_{\text{secant}}}{W_{\text{secant}}} = 0.71\,.$$

▷ Secant method more efficient

③



"Simulated iterations"

$C = 7$

$|e^{(0)}| = 0.1$

▷ Review questions 8.4.3.15

ETH Lecture 401-0663-00L Numerical Methods for CSE

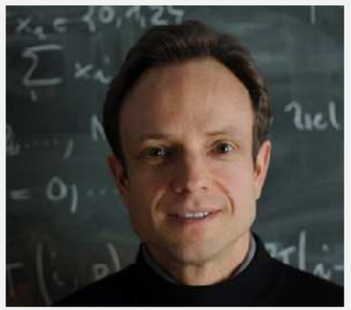
Course Video

Section 8.5.1: The Newton Iteration in $\mathbb{R}^n$ (I)

Prof. R. Hiptmair, SAM, ETH Zurich

Date: December 5, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- Multi-dimensional calculus: Jacobians

Dependencies. Section 8.4.2.1

Duration: 16 minutes

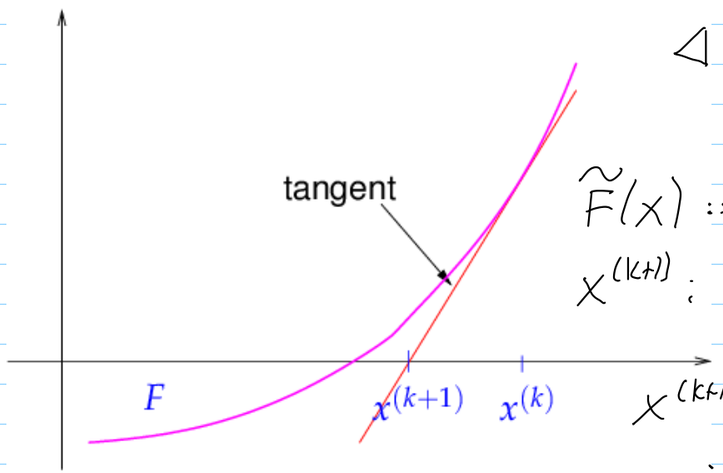
VIII Iterative Methods for Non-Linear Systems of Equations

8.5 Newton's Method in $\mathbb{R}^n$

NLSE: Find $x^* \in D \subset \mathbb{R}^n$: $F(x^*) = 0$

Assume $F : D \subset \mathbb{R}^n \longrightarrow \mathbb{R}^n$ continuously differentiable

8.5.1. The Newton Iteration



◁ Newton method in 1D

Model function

$\widetilde{F}(x) := F(x^{(k)}) + F'(x^{(k)})(x - x^{(k)})$

$x^{(k+1)}$: $\widetilde{F}(x^{(k+1)}) \stackrel{!}{=} 0$

$x^{(k+1)} = x^{(k)} - \frac{F(x^{(k)})}{F'(x^{(k)})}$

if $F'(x^{(k)}) \neq 0$

(2)

$\widetilde{F} \triangleq$ truncated Taylor exp. of F around $x^{(k)}$: a local linearization

▷ Same $\mathbb{R}^n$: $[\, F : D \subset \mathbb{R}^n \to \mathbb{R}^n \,]$

$\widetilde{F}(x) := F(x^{(k)}) + DF(x^{(k)})(x - x^{(k)})$

↑ Jacobian $\in \mathbb{R}^{n,n}$: regular!

$\widetilde{F}(x^{(k+1)}) = 0 \;:\; x^{(k+1)} = x^{(k)} - DF(x^{(k)})^{-1} F(x^{(k)})$

Idea (→ Section 8.4.2.1): local linearization:

Given $\mathbf{x}^{(k)} \in D$ ➢ $\mathbf{x}^{(k+1)}$ as zero of affine linear model function

$$F(\mathbf{x}) \approx \widetilde{F}_k(\mathbf{x}) := F(\mathbf{x}^{(k)}) + \mathrm{D}F(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)}) ,$$

$\mathrm{D}F(\mathbf{x}) \in \mathbb{R}^{n,n}$ = Jacobian, $\mathrm{D}F(\mathbf{x}) := \left[\frac{\partial F_j}{\partial x_k}(\mathbf{x})\right]_{j,k=1}^{n}$.

► **Newton iteration**: (generalizes (8.4.2.1) to $n > 1$)

$$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} - \mathrm{D}F(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)}) \quad , \quad [\text{ if } \mathrm{D}F(\mathbf{x}^{(k)}) \text{ regular }] \tag{8.5.1.6}$$

Terminology: $\underline{s} := -\mathrm{D}F(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)})$ is called the Newton correction

↑

solve $n \times n$ LSE: $DF(\underline{x}^{(k)})\underline{s} = -F(\underline{x}^{(k)})$

§ 8.5.1.8 (Newton's method in C++)

C++11-code 8.5.1.9: Newton's method in C++

```
template <typename FuncType, typename JacType, typename VecType>
VecType newton(FuncType &&F, JacType &&DFinv, VecType x, const double rtol,
               const double atol) {
  // Note that the vector x passes both the initial guess and also
  // contains the iterates
  VecType s(x.size()); // Vector for Newton corrections
  // Main loop
  do {
    s = DFinv(x, F(x)); // compute Newton correction
    x -= s;             // compute next iterate
  }
  // Newton correction based termination (relative and absolute)
  while ((s.norm() > rtol * x.norm()) && (s.norm() > atol));
  return x;
}
```

relative tolerance absolute tolerance

Rem. 8.5.1.11 (Affine invariance of Newton method)

Given F consider the Newton iteration

$G_A(x) = 0$, $G_A(x) = A \cdot F(x)$, $A \in \mathbb{R}^{n,n}$ regular

$[\, DG_A(x) = A \cdot DF(x) \,:$ product rule $]$

N.I.: $x^{(k+1)} = x^{(k)} - [A \cdot DF(x^{(k)})]^{-1} A\, F(x^{(k)})$

$= x^{(k)} - DF(x^{(k)})^{-1} \cancel{A^{-1}} A\, F(x^{(k)})$

↳ same as N.I. for F $\forall A$

▷ Stopping rules should be affine invariant

True for code 8.5.1.9 ?

▷ Review questions 8.5.1.46

NumCSE@ETHZ

ETH Lecture 401-0663-00L Numerical Methods for CSE

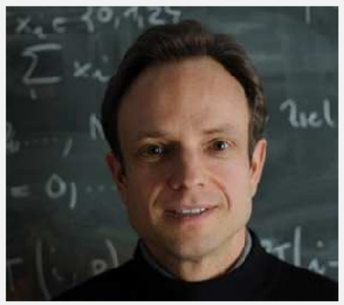
Course Video

Section 8.5.1: The Newton Iteration in $\mathbb{R}^n$ (II)

Prof. R. Hiptmair, SAM, ETH Zurich

Date: December 5, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 8.5.1 (Part I)

Duration: 30 minutes

VIII Iterative Methods for Non-Linear Systems of Equations

8.5 Newton's Method in $\mathbb{R}^n$

NLSE: Given C^1 $F: D \subset \mathbb{R}^n \longrightarrow \mathbb{R}^n$, $n \in \mathbb{N}$,

find $\underline{x}^* \in D$: $F(\underline{x}^*) = 0$

8.5.1. The Newton Iteration

Newton iteration: (generalizes (8.4.2.1) to $n > 1$)

Jacobian $\in \mathbb{R}^{n,n}$

$$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} - \mathrm{D}F(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)}) \quad , \quad [\text{if } \mathrm{D}F(\mathbf{x}^{(k)}) \text{ regular}] \tag{8.5.1.6}$$

Terminology: $-\mathrm{D}F(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)})$ is called the Newton correction $\underline{s} \in \mathbb{R}^n$

Newton correction by solving LSE

$$\mathrm{D}F(x^{(k)})\,\underline{s} = -F(x^{(k)}) \quad (\text{NCLSE})$$

§ 8.5.1.25 (A "quasi-linear" system of equations) → § 8.5.1.22

$$\mathbf{A}(\mathbf{x})\cdot\mathbf{x} = \mathbf{b} \quad , \quad \mathbf{b}\in\mathbb{R}^n, \quad \mathbf{A}:\mathbb{R}^n \to \mathbb{R}^{n,n} \text{ cont. diff.}$$

$$\Updownarrow$$

$$F(\mathbf{x}) = 0 \quad , \quad F(\mathbf{x}) := \mathbf{b} - \mathbf{A}(\mathbf{x})\mathbf{x} \quad , \quad DF(\mathbf{x}) ?$$

- $\cdot \hat{=}$ bilinear product : product rule

Product rule: $F : D \subset V \mapsto W$, $G : D \subset V \mapsto U$ sufficiently smooth, $b : W \times U \mapsto Z$ bilinear, *ie.*, linear in each argument:

$$T(\mathbf{x}) := b(F(\mathbf{x}), G(\mathbf{x})) \quad \Rightarrow \quad \mathrm{D}\,T(\mathbf{x})\mathbf{h} = b(\mathrm{D}\,F(\mathbf{x})\mathbf{h}, G(\mathbf{x})) + b(F(\mathbf{x}), \mathrm{D}\,G(\mathbf{x})\mathbf{h}) , \quad \mathbf{h}\in V, \mathbf{x}\in D . \tag{8.5.1.17}$$

Here: $b \hat{=} \cdot : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, $F(\mathbf{x}) \leftrightarrow \mathbf{A}(\mathbf{x})$, $G := \mathrm{Id}$

$$\triangleright \quad DF(\mathbf{x})\mathbf{h} = \underbrace{(DA(\mathbf{x})\mathbf{h})\cdot\mathbf{x}}_{\in\mathbb{R}^{n,n}} + \mathbf{A}(\mathbf{x})\cdot\mathbf{h}$$

$$\Downarrow$$

NCLSE

$$(DA(\mathbf{x}^{(k)})\mathbf{s})\mathbf{x}^{(k)} + \mathbf{A}(\mathbf{x}^{(k)})\mathbf{s} = \mathbf{A}(\mathbf{x}^{(k)})\mathbf{x}^{(k)} - \mathbf{b}$$

NumCSE@ETHZ

Ex. 8.5.1.30: (A special quasi-linear system of equations)

$\mathbf{b}\in\mathbb{R}^n$

$\mathbf{A}(\mathbf{x})\mathbf{x} = \mathbf{b}$, $\mathbf{A}(\mathbf{x}) :=$

$$\begin{bmatrix} \gamma(\mathbf{x}) & 1 & & & & \\ 1 & \gamma(\mathbf{x}) & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & \gamma(\mathbf{x}) & 1 \\ & & & & 1 & \gamma(\mathbf{x}) \end{bmatrix} \in \mathbb{R}^{n\times n}, \qquad \gamma(\mathbf{x}) = 3 + \|\mathbf{x}\|_2 \tag{8.5.1.31}$$

$$\Updownarrow$$

$F(\mathbf{x}) := \mathbf{A}(\mathbf{x})\mathbf{x} - \mathbf{b} = 0$

$$\triangleright \quad \mathbf{A}(\mathbf{x})\mathbf{x} = \mathbf{T}\mathbf{x} + \mathbf{x}\|\mathbf{x}\|_2 , \quad \mathbf{T} := \begin{bmatrix} 3 & 1 & & & & \\ 1 & 3 & 1 & & & \\ & \ddots & 3 & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & 3 & 1 \\ & & & & 1 & 3 \end{bmatrix}$$

$$F(\mathbf{x}) = \mathbf{T}\mathbf{x} + \mathbf{x}\|\mathbf{x}\|_2 - \mathbf{b}$$

$$DF(\mathbf{x})\mathbf{h} = D\{\underbrace{\mathbf{x}\mapsto \mathbf{T}\mathbf{x}}_{\text{linear}}\}(\mathbf{x})\mathbf{h} + D\{\mathbf{x}\to \mathbf{x}\cdot\|\mathbf{x}\|_2\}(\mathbf{x})\mathbf{h}$$

↑ product rule

Use $D\{\mathbf{x}\mapsto \|\mathbf{x}\|_2\}(\mathbf{x})\mathbf{h} = \frac{\mathbf{x}^T}{\|\mathbf{x}\|_2}\cdot\mathbf{h}$

$$\triangleright \quad DF(\mathbf{x})\mathbf{h} = \mathbf{T}\mathbf{h} + \mathbf{h}\cdot\|\mathbf{x}\|_2 + \mathbf{x}\cdot\frac{\mathbf{x}^T}{\|\mathbf{x}\|_2}\cdot\mathbf{h}$$

$$= \left(\mathbf{A}(\mathbf{x}) + \frac{\mathbf{x}\mathbf{x}^T}{\|\mathbf{x}\|_2}\right)\mathbf{h}$$

NumCSE@ETHZ

▷ NCLSE:

$$\underbrace{\left(A(x^{(k)}) + \frac{x^{(k)}x^{(k)\top}}{\|x^{(k)}\|_2}\right)}_{}\, s = b - A(x^{(k)})x^{(k)}$$

rank-1-modification of tridiagonal matrix;

Use SMW formula + QR for $O(n)$-solve

Ex. 8.5.1.38 (Matrix inversion by means of Newton's method)

$A \in \mathbb{R}^{n,n}$ regular

$F(A^{-1}) = 0$ for $\begin{cases} F : \mathbb{R}^{n,n}_* \longrightarrow \mathbb{R}^{n,n} \\ F(X) := A - X^{-1} \end{cases}$

NCLSE?

Ex. 8.5.1.35 (Derivative of matrix inversion)

$$\text{inv}: \begin{cases} \mathbb{R}^{n,n}_* \longrightarrow \mathbb{R}^{n,n} \\ X \longrightarrow X^{-1} \end{cases}, \quad \mathbb{R}^{n,n}_* \hat{=} \text{regular matrices}$$

D inv = ?

Idea: implicit differentiation

D: $X \cdot \text{inv}(X) = I$

product rule: $H \cdot \text{inv}(X) + X \cdot D\text{inv}(X)H = 0$, $H \in \mathbb{R}^{n,n}$

▷ $D\text{inv}(X)H = -X^{-1}HX^{-1}$

Ex. 8.5.1.38 cnt'd:

$DF(X)H = X^{-1}HX^{-1}$, $X \in \mathbb{R}^{n,n}_*$

NCLSE:

$DF(X^{(k)})S = (X^{(k)})^{-1}S(X^{(k)})^{-1} = -F(X^{(k)}) = (X^{(k)})^{-1} - A$

▷ $S = X^{(k)}((X^{(k)})^{-1} - A)X^{(k)} = X^{(k)} - X^{(k)}AX^{(k)}$

N.I.: $X^{(k+1)} = X^{(k)} + S = X^{(k)}(2I - AX^{(k)})$

(4)

Rem. 8.5.1.44 (Simplified Newton method)

≙ Newton method with "frozen Jacobian"

For $F(x) = 0$:

$$x^{(k+1)} = x^{(k)} - DF(x^{(0)})^{-1} F(x^{(k)}) \,, \quad k \in \mathbb{N}$$

Benefit: Reuse of LU-dec. of $DF(x^{(0)})$

C++ code 8.5.1.44: Efficient implementation of simplified Newton method

```
// C++ template for simplified Newton method
template <typename Func, typename Jac, typename Vec>
void simpnewton(Vec& x, Func F, Jac DF, double rtol, double atol)
{
  auto lu = DF(x).lu(); // do LU decomposition once!
  Vec s;                // Newton correction
  double ns,nx;         // auxiliary variables for termination control
  do {
    s = lu.solve(F(x));   --> O(n^2)
    x = x-s;            // new iterate
    ns = s.norm(); nx = x.norm();
  }
  // termination based on relative and absolute tolerance
  while((ns > rtol*nx) && (ns > atol));
}
```

Drawback: Slower cvg.

▷ Review questions 8.5.1.46

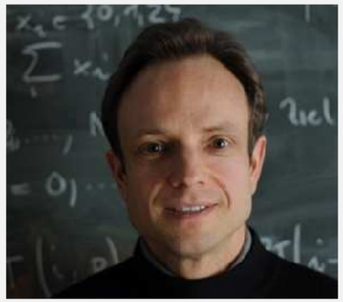
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
§ 8.5.1.15: Multi-dimensional Differentiation

Prof. R. Hiptmair, SAM, ETH Zurich

Date: December 5, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.
- Elementary derivatives
- Multi-dimensional calculus

Dependencies. Eq. (8.5.1.6)

Duration: 35 minutes

VIII Iterative Methods for Non-Linear Systems of Equations

8.5 Newton's Method in $\mathbb{R}^n$

NLSE: Given C^1 $F: D \subset \mathbb{R}^n \longrightarrow \mathbb{R}^n$, $n \in \mathbb{N}$,
find $x^* \in D$: $F(x^*) = 0$

8.5.1. The Newton Iteration

$$x^{(k+1)} := x^{(k)} - DF(x^{(k)})^{-1} F(x^{(k)}), \quad k \in \mathbb{N}_0$$

$DF(x^{(k)}) \in \mathbb{R}^{n,n} \hat{=}$ Jacobian

$F = [F_1, \ldots, F_n]$:

$$\mathrm{D}F(\mathbf{z}) = \left[\frac{\partial F_i}{\partial x_j}(\mathbf{z})\right]_{i,j=1}^{n} = \begin{bmatrix} \frac{\partial F_1}{\partial x_1}(\mathbf{z}) & \frac{\partial F_1}{\partial x_2}(\mathbf{z}) & \cdots & \cdots & \frac{\partial F_1}{\partial x_n}(\mathbf{z}) \\ \frac{\partial F_2}{\partial x_1}(\mathbf{z}) & & & & \frac{\partial F_2}{\partial x_n}(\mathbf{z}) \\ \vdots & & & & \vdots \\ \frac{\partial F_n}{\partial x_1}(\mathbf{z}) & \frac{\partial F_n}{\partial x_2}(\mathbf{z}) & \cdots & \cdots & \frac{\partial F_n}{\partial x_n}(\mathbf{z}) \end{bmatrix}. \tag{8.3.2.8}$$

(2)

Newton correction $\underline{s} := -DF(x^{(k)})^{-1} F(x^{(k)})$

$\Updownarrow$

LSE: $DF(x^{(k)})\,\underline{s} = -F(x^{(k)})$

§ 8.5.1.12 (Derivatives)

Derivation $\leftrightarrow$ local linearization

$F(\underline{x}+\underline{h}) = F(\underline{x}) + \underbrace{DF(x)\,\underline{h}}_{\text{A linear mapping } \mathbb{R}^n \to \mathbb{R}^n} + O(\|h\|^2)$ for $\underline{h} \to 0$

$\underline{h}$: perturbation $\in \mathbb{R}^n$

Definition 8.5.1.13. Derivative of functions between vector spaces

Let V, W be finite dimensional vector spaces and $F : D \subset V \mapsto W$ a sufficiently smooth mapping. The **derivative** (differential) $\mathrm{D}F(\mathbf{x})$ of F in $\mathbf{x} \in V$ is the *unique* linear mapping $\mathrm{D}F(\mathbf{x}) : V \mapsto W$ such that there is a $\delta > 0$ and a function $\epsilon : [0,\delta] \to \mathbb{R}^+$ satisfying $\lim_{\xi\to 0} \epsilon(\xi) = 0$ such that

$$\|F(\mathbf{x}+\mathbf{h}) - F(\mathbf{x}) - \mathrm{D}F(\mathbf{x})\mathbf{h}\| = \epsilon(\|\mathbf{h}\|) \quad \forall \mathbf{h} \in V,\ \|\mathbf{h}\| < \delta\,. \tag{8.5.1.14}$$

$\hookrightarrow O(\|\underline{h}\|^2)$

$V = W = \mathbb{R}$

$DF(x) \leftrightarrow F'(x) \in \mathbb{R} \leftrightarrow$ linear mapping $\begin{cases} \mathbb{R} \to \mathbb{R} \\ \xi \to F'(x)\,\xi \end{cases}$

§ 8.5.1.15 ("High-level" differentiation)

- $F : V \to W$ linear $\Rightarrow$ $DF = F$!

($F(x) = Ax$, $A \in \mathbb{R}^{m,n} \Rightarrow DF(\underline{x})\underline{h} = A\underline{h} = F(\underline{h})$)

- Chain rule: $F : V \to W$, $G : \mathcal{U} \to V$

$$F(G(\underline{x}+\underline{h})) = F(G(\underline{x}) + DG(\underline{x})\,\underline{h} + O(\|h\|^2))$$
$$= F(G(\underline{x})) + \underbrace{DF(G(\underline{x}))DG(\underline{x})\,\underline{h}}_{= D(F\circ G)(\underline{x})\,\underline{h}} + O(\|\underline{h}\|^2)$$

Chain rule: For $F : V \mapsto W$, $G : U \mapsto V$ sufficiently smooth

$$\mathrm{D}(F\circ G)(\mathbf{x})\mathbf{h} = \mathrm{D}F(G(\mathbf{x}))(\mathrm{D}G(\mathbf{x})\mathbf{h})\,,\quad \mathbf{h} \in V,\ \mathbf{x} \in D\,. \tag{8.5.1.16}$$

- Product rule:

  - $F : V \to W$, $G : V \to \mathcal{U}$
  - $b : W \times \mathcal{U} \to Z$ bilinear

$\Rightarrow$ $T : V \to Z$, $T(\underline{x}) := b(F(\underline{x}), G(\underline{x}))$

③

$(V = W = U = Z = \mathbb{R}: \quad b(x,y) = x \cdot y)$

$T(x) = F(x) \cdot G(x) \Rightarrow T'(x) = F'(x)G(x) + F(x)G'(x)$

As L.M.: $T'(x)h = (F'(x)h) \cdot G(x) + F(x) \cdot (G'(x)h)$

$$\begin{aligned} T(\mathbf{x}+\mathbf{h}) &= b(F(\mathbf{x}+\mathbf{h}), G(\mathbf{x}+\mathbf{h})) \\ &= b(F(\mathbf{x}) + \mathrm{D}F(\mathbf{x})\mathbf{h} + o(\|\mathbf{h}\|), G(\mathbf{x}) + \mathrm{D}G(\mathbf{x}) + o(\|\mathbf{h}\|)) \\ &\overset{(*)}{=} b(F(\mathbf{x}), G(\mathbf{x})) + \underbrace{b(\mathrm{D}F(\mathbf{x})\mathbf{h}, G(\mathbf{x})) + b(F(\mathbf{x}), \mathrm{D}G(\mathbf{x})\mathbf{h})}_{=\mathrm{D}T(\mathbf{x})\mathbf{h}} + o(\|\mathbf{h}\|) . \end{aligned}$$

$o(\|h\|^2)$

(*): b bilinear

Product rule: $F : D \subset V \mapsto W$, $G : D \subset V \mapsto U$ sufficiently smooth, $b : W \times U \mapsto Z$ bilinear, *ie.*, linear in each argument:

$$T(\mathbf{x}) := b(F(\mathbf{x}), G(\mathbf{x})) \Rightarrow \mathrm{D}T(\mathbf{x})\mathbf{h} = b(\mathrm{D}F(\mathbf{x})\mathbf{h}, G(\mathbf{x})) + b(F(\mathbf{x}), \mathrm{D}G(\mathbf{x})\mathbf{h}) , \quad \mathbf{h} \in V, \mathbf{x} \in D . \tag{8.5.1.17}$$

Special notion for $F : \mathbb{R}^n \to \mathbb{R}$:

Definition 8.5.1.18. Gradient and Hessian

For sufficiently smooth $F : D \subset \mathbb{R}^n \mapsto \mathbb{R}$ the **gradient** $\mathbf{grad}\, F : D \mapsto \mathbb{R}^n$, and the **Hessian** (matrix) $\mathrm{H}F(\mathbf{x}) : D \mapsto \mathbb{R}^{n,n}$ are defined as

$$\mathbf{grad}\, F(\mathbf{x})^T\mathbf{h} := \mathrm{D}F(\mathbf{x})\mathbf{h} , \quad \mathbf{h}_1^T \mathrm{H}F(\mathbf{x})\mathbf{h}_2 := \mathrm{D}(\mathrm{D}F(\mathbf{x})(\mathbf{h}_1))(\mathbf{h}_2) , \quad \forall \mathbf{h}, \mathbf{h}_1, \mathbf{h}_2 \in \mathbb{R}^n .$$

$\mathbf{grad}\, F(\mathbf{x})^T\mathbf{h}$: L.M. $\mathbb{R}^n \to \mathbb{R}$

$\mathbf{grad}\, F(\mathbf{x})$: $\hat{=}\ \mathrm{D}(\mathrm{D}F) \hat{=}$ L.M. $\mathbb{R}^n \to \{$L.M. $\mathbb{R}^n \to \mathbb{R}\}$ $\hat{=}$ bilinear mapping $\mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$

Ex 8.5.1.19 (Derivative of a bilinear form)

$\Psi : \mathbb{R}^n \to \mathbb{R}$, $\Psi(x) := x^T A x$, $A \in \mathbb{R}^{n,n}$

Product rule: $F : D \subset V \mapsto W$, $G : D \subset V \mapsto U$ sufficiently smooth, $b : W \times U \mapsto Z$ bilinear, *ie.*, linear in each argument:

$$T(\mathbf{x}) := b(F(\mathbf{x}), G(\mathbf{x})) \Rightarrow \mathrm{D}T(\mathbf{x})\mathbf{h} = b(\mathrm{D}F(\mathbf{x})\mathbf{h}, G(\mathbf{x})) + b(F(\mathbf{x}), \mathrm{D}G(\mathbf{x})\mathbf{h}) , \quad \mathbf{h} \in V, \mathbf{x} \in D . \tag{8.5.1.17}$$

Now: $F, G = Id : \mathbb{R}^n \to \mathbb{R}^n$, $b(x,y) = x^T A y$

$D\Psi(x)h = h^T A x + x^T A h = (x^T A^T + x^T A)h$

$\Rightarrow$ $\mathrm{grad}\,\Psi(x) = (A^T + A)x \in \mathbb{R}^n$

"Pedestrian way"

$$\Psi(\mathbf{x}) = \sum_{k=1}^{n}\sum_{j=1}^{n}(\mathbf{A})_{k,j}x_k x_j = (\mathbf{A})_{i,i}x_i^2 + \sum_{\substack{j=1\\ j\neq i}}^{n}(\mathbf{A})_{i,j}x_i x_j + \sum_{\substack{k=1\\ k\neq i}}^{n}(\mathbf{A})_{k,i}x_k x_i + \sum_{\substack{k=1\\ k\neq i}}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n}(\mathbf{A})_{k,j}x_k x_j ,$$

$$\begin{aligned} \frac{\partial \Psi(\mathbf{x})}{\partial x_i}(\mathbf{x}) &= 2(\mathbf{A})_{i,i}x_i + \sum_{\substack{j=1\\ j\neq i}}^{n}(\mathbf{A})_{i,j}x_j + \sum_{\substack{k=1\\ k\neq i}}^{n}(\mathbf{A})_{k,i}x_k = \sum_{j=1}^{n}(\mathbf{A})_{i,j}x_j + \sum_{k=1}^{n}(\mathbf{A})_{k,i}x_k \\ &= (\mathbf{A}\mathbf{x} + \mathbf{A}^\top\mathbf{x})_i , \quad i = 1, \ldots, n . \end{aligned}$$

Ex 8.5.1.20 (Derivative of Euclidean norm)

$F : \mathbb{R}^n \setminus \{0\} \to \mathbb{R}, \quad F(\mathbf{x}) = \|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$

$F = G \circ H, \quad G(\xi) := \sqrt{\xi}, \quad H(\mathbf{x}) = \mathbf{x}^T\mathbf{x}$

$DG(\xi)\eta = \frac{1}{2\sqrt{\xi}}\eta, \quad DH(\mathbf{x})\mathbf{h} = 2\mathbf{x}^T\mathbf{h}$

Chain rule: For $F : V \mapsto W$, $G : U \mapsto V$ sufficiently smooth

$$D(F \circ G)(\mathbf{x})\mathbf{h} = DF(G(\mathbf{x}))(DG(\mathbf{x})\mathbf{h}), \quad \mathbf{h} \in V, \mathbf{x} \in D. \tag{8.5.1.16}$$

Here: $F \leftarrow G, \quad G \leftarrow H$

$$\Rightarrow DF(\mathbf{x})\mathbf{h} = DG(H(\mathbf{x}))DH(\mathbf{x})\mathbf{h}$$

$$= \frac{1}{2\sqrt{\mathbf{x}^T\mathbf{x}}} \cdot 2\mathbf{x}^T\mathbf{h}$$

$$\operatorname{grad} F(\mathbf{x}) = \frac{\mathbf{x}}{\|\mathbf{x}\|_2}, \quad \mathbf{x} \neq 0$$

§ 8.5.1.22 (Newton iteration via product rule)

We seek solutions of $F(\mathbf{x}) = \mathbf{0}$ with $F(\mathbf{x}) := b(G(\mathbf{x}), H(\mathbf{x}))$, where

- V, W are some vector spaces (finite- or even infinite-dimensional),
- $G : D \to V$, $H : D \to W$, $D \subset \mathbb{R}^n$, are continuously differentiable in the sense of Def. 8.5.1.13,
- $b : V \times W \mapsto \mathbb{R}^n$ is bilinear (linear in each argument).

[$F : D \subset \mathbb{R}^n \longrightarrow \mathbb{R}^n$]

LSE for Newton correction: $DF(\mathbf{x})\mathbf{s} = -F(\mathbf{x})$

Product rule:

$$DF(\mathbf{x})\mathbf{h} = b(DG(\mathbf{x})\mathbf{h}, H(\mathbf{x})) + b(G(\mathbf{x}), DH(\mathbf{x})\mathbf{h}), \quad \mathbf{h} \in \mathbb{R}^n. \tag{8.5.1.23}$$

$$\Downarrow \text{ P.R.}$$

$$\underbrace{b(DG(\mathbf{x}^{(k)})\mathbf{s}, H(\mathbf{x}^{(k)})) + b(G(\mathbf{x}^{(k)}), DH(\mathbf{x}^{(k)})\mathbf{s})}_{\hat{=}\, M\mathbf{s},\ M \in \mathbb{R}^{n,n}} = \underbrace{-b(G(\mathbf{x}^{(k)}), H(\mathbf{x}^{(k)}))}_{\in \mathbb{R}^n}. \tag{8.5.1.24}$$

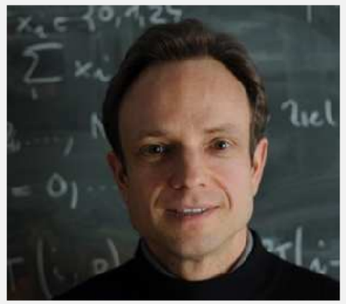
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 8.5.2: Convergence of Newton's Method

Prof. R. Hiptmair, SAM, ETH Zurich

Date: December 5, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 8.5.1

Duration: 16 minutes

VIII Iterative Methods for Non-Linear Systems of Equations

8.5 Newton's Method in $\mathbb{R}^n$

NLSE: Given C^1 $F: D \subset \mathbb{R}^n \longrightarrow \mathbb{R}^n$, $n \in \mathbb{N}$,

find $\underline{x}^* \in D$: $F(\underline{x}^*) = 0$

Newton iteration: (generalizes (8.4.2.1) to $n > 1$)

$$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} - DF(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)}) \quad , \quad [\text{if } DF(\mathbf{x}^{(k)}) \text{ regular}] \tag{8.5.1.6}$$

Terminology: $-DF(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)})$ is called the Newton correction

NumCSE@ETHZ

8.5.2. Convergence of Newton's Method

$n = 1 \quad \Rightarrow \quad$ local quadratic cvg.

Definition 8.2.2.10. Order of convergence → , ,

A **convergent** sequence $\mathbf{x}^{(k)}$, $k = 0,1,2,\ldots$, in $\mathbb{R}^n$ with limit $\mathbf{x}^* \in \mathbb{R}^n$ converges with order p, $p \geq 1$, if

$$\exists C > 0: \quad \left\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\right\| \leq C\left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\|^p \quad \forall k \in \mathbb{N}_0 \,, \tag{8.2.2.11}$$

and, in addition, $C < 1$ in the case $p = 1$ (linear convergence → Def. 8.2.2.1).

N.I. $\Leftrightarrow$ fixed-point iteration in $\mathbb{R}^n$

$$\underline{x}^{(k+1)} = \Phi(\underline{x}^{(k)}), \quad \Phi(x) = x - DF(x)^{-1} \cdot F(x)$$

[$F(\underline{x}^*) = 0 \Leftrightarrow \Phi(\underline{x}^*) = \underline{x}^*$]

$$\|\underline{x}^{(k+1)} - \underline{x}^*\| = \|\Phi(\underline{x}^{(k+1)}) - \Phi(\underline{x}^*)\|$$

Taylor, $F \in C^2$ $= \|D\Phi(\underline{x}^*)(\underline{x}^{(k)} - \underline{x}^*) + O(\|\underline{x}^{(k)} - \underline{x}^*\|^2)$

By product rule:

$$D\Phi(\underline{x})\underline{h} = \underline{h} - D\{\underline{x} \mapsto DF(\underline{x})^{-1}\underline{h} \cdot F(\underline{x}) + DF(\underline{x})^{-1} DF(\underline{x})\underline{h}$$

$$D\Phi(\underline{x}^*)\underline{h} = \underline{h} - D\{\underline{x} \mapsto DF(\underline{x})^{-1}\underline{h} \cdot \underbrace{F(\underline{x}^*)}_{=0} + DF(\underline{x}^*)^{-1} DF(\underline{x}^*)\underline{h}$$

$$= 0 \quad \forall \underline{h}$$

▷ $\|\underline{x}^{(k+1)} - \underline{x}^*\| = O(\|\underline{x}^{(k)} - \underline{x}^*\|^2)$

$\hat{=}$ local quadratic cvg.

Ex. 8.5.2.3 (Convergence of Newton iteration for matrix inversion → Ex. 8.5.1.38)

$A \in \mathbb{R}^{n,n}$ regular:

$$\mathbf{X}^{(k+1)} = \mathbf{X}^{(k)} - \left(\mathbf{X}^{(k)}\mathbf{A}\mathbf{X}^{(k)} - \mathbf{X}^{(k)}\right) = \mathbf{X}^{(k)}\left(2\mathbf{I} - \mathbf{A}\mathbf{X}^{(k)}\right) . \tag{8.5.1.42}$$

▷ Recursion for error $E^{(k)} := X^{(k)} - A^{-1} \in \mathbb{R}^{n,n}$

$$\begin{aligned}
\mathbf{E}^{(k+1)} &= \mathbf{X}^{(k+1)} - \mathbf{A}^{-1} \\
&\overset{(8.5.1.42)}{=} \mathbf{X}^{(k)}\left(2\mathbf{I} - \mathbf{A}\mathbf{X}^{(k)}\right) - \mathbf{A}^{-1} \\
&= (\mathbf{E}^{(k)} + \mathbf{A}^{-1})\left(2\mathbf{I} - \mathbf{A}(\mathbf{E}^{(k)} + \mathbf{A}^{-1})\right) - \mathbf{A}^{-1} \\
&= (\mathbf{E}^{(k)} + \mathbf{A}^{-1})(\mathbf{I} - \mathbf{A}\mathbf{E}^{(k)}) - \mathbf{A}^{-1} = -\mathbf{E}^{(k)}\mathbf{A}\mathbf{E}^{(k)} .
\end{aligned}$$

$\|AB\| \leq \|A\|\,\|B\|$:

$$\blacktriangleright \quad \left\|\mathbf{E}^{(k+1)}\right\| \leq \left\|\mathbf{E}^{(k)}\right\|^2 \|\mathbf{A}\| . \rightarrow \text{local quadratic cvg.} \tag{8.5.2.4}$$

N.I. convergent:

$$\|E^{(0)}A\| = \|X^{(0)}A - I\| < 1$$

▷ Review questions 8.5.2.8

NumCSE@ETHZ

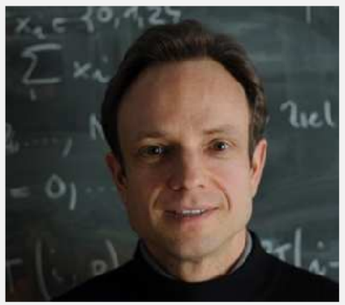
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 8.5.3: Termination of Newton Iteration

Prof. R. Hiptmair, SAM, ETH Zurich

Date: December 5, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 8.5.1

Duration: 12 minutes

VIII Iterative Methods for Non-Linear Systems of Equations

8.5 Newton's Method in $\mathbb{R}^n$

NLSE: Given C^1 $F: D \subset \mathbb{R}^n \longrightarrow \mathbb{R}^n$, $n \in \mathbb{N}$,
find $x^* \in D$: $F(x^*) = 0$

▶ **Newton iteration**: (generalizes (8.4.2.1) to $n > 1$)

$$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} - \mathrm{D}F(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)}) \quad , \quad [\text{if } \mathrm{D}F(\mathbf{x}^{(k)}) \text{ regular}] \tag{8.5.1.6}$$

Terminology: $-\mathrm{D}F(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)})$ is called the Newton correction

▷ Local quadratic convergence to $x^* \in D$, $F(x^*) = 0$, if $DF(x^*)$ regular

8.5.3. Termination of Newton Iteration

Goal: $\|x^{(k)} - x^*\|$ "small"

Quad cvg. $\Rightarrow$ If N.I. converges, asymptotically

$$\|x^{(k+1)} - x^*\| \lesssim \|x^{(k)} - x^*\|$$

$$\Rightarrow \quad \|x^{(k)} - x^*\| \approx \|x^{(k)} - x^{(k+1)}\|$$

STOP, as soon as $\left\|\Delta \mathbf{x}^{(k)}\right\| \leq \tau_{\text{rel}} \left\|\mathbf{x}^{(k)}\right\|$ **or** $\left\|\Delta \mathbf{x}^{(k)}\right\| \leq \tau_{\text{abs}}$, (8.5.3.2)

with Newton correction $\Delta \mathbf{x}^{(k)} := \mathrm{D}F(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)})$.

→ $x^{(k)}$ is already accurate enough

needlessly computed $x^{(k+1)}$: significant additional cost*

→ very few steps needed

* cost = $O(n^3)$ per step

§ 8.5.3.4 (Termination of Newton iteration based on simplified Newton correction)

Idea: Replace $\mathrm{D}F(\mathbf{x}^{(k)})$ with $\mathrm{D}F(\mathbf{x}^{(k-1)})$ in any correction based termination criterion.

Rationale: LU-decomposition of $\mathrm{D}F(\mathbf{x}^{(k-1)})$ is already available ➤ less effort.

Terminology: $\Delta \bar{\mathbf{x}}^{(k)} := \mathrm{D}F(\mathbf{x}^{(k-1)})^{-1} F(\mathbf{x}^{(k)}) \triangleq$ simplified Newton correction

Economical correction based termination criterion for Newton's method:

STOP, as soon as $\left\|\Delta \bar{\mathbf{x}}^{(k)}\right\| \leq \tau_{\text{rel}} \left\|\mathbf{x}^{(k)}\right\|$ **or** $\left\|\Delta \bar{\mathbf{x}}^{(k)}\right\| \leq \tau_{\text{abs}}$, (8.5.3.5)

with simplfied Newton correction $\Delta \bar{\mathbf{x}}^{(k)} := \mathrm{D}F(\mathbf{x}^{(k-1)})^{-1} F(\mathbf{x}^{(k)})$.

Affine invariant: does not change when $F \longrightarrow AF$, $A \in \mathbb{R}^{n,n}$ regular

C++11 code 8.5.3.6: Generic Newton iteration with termination criterion (8.5.3.5)

```
template <typename FuncType, typename JacType, typename VecType>
void newton_stc(const FuncType &F, const JacType &DF, VecType &x, double rtol,
                double atol) {
  using scalar_t = typename VecType::Scalar;
  scalar_t sn;
  do {
    auto jacfac = DF(x).lu(); // LU-factorize Jacobian ]   -> O(n^3)
    x -= jacfac.solve(F(x));  // Compute next iterate      -> O(n^2)
    // Compute norm of simplified Newton correction
    sn = jacfac.solve(F(x)).norm();                        -> O(n^2)
  }
  // Termination based on simplified Newton correction
  while ((sn > rtol * x.norm()) && (sn > atol));
}
```

▷ Review questions 8.5.3.9

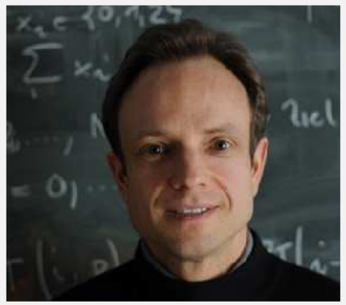
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 8.5.4: Damped Newton Method

Prof. R. Hiptmair, SAM, ETH Zurich

Date: December 5, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 8.5.1, Section 8.5.2

Duration: 20 minutes

Note: Possible minor *mismatch of video and tablet notes*!

[Corrections and updates can be incorporated into tablet notes only]

VIII Iterative Methods for Non-Linear Systems of Equations

8.5 Newton's Method in $\mathbb{R}^n$

NLSE: Given C^1 $F: D \subset \mathbb{R}^n \longrightarrow \mathbb{R}^n$, $n \in \mathbb{N}$,

find $x^* \in D$: $F(x^*) = 0$

Newton iteration: (generalizes (8.4.2.1) to $n > 1$)

$$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} - \mathrm{D}F(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)}) \quad , \quad [\text{if } \mathrm{D}F(\mathbf{x}^{(k)}) \text{ regular}] \tag{8.5.1.6}$$

Terminology: $-\mathrm{D}F(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)})$ is called the Newton correction

Summary: Newton's method

quadratic local cvg.

converges *asymptotically* very fast: doubling of number of significant digits in each step

often a very small region of convergence, which requires an initial guess rather close to the solution.

} local cvg.

8.5.4. Damped Newton Method

Goal: Enlarge domain of cvg.

Ex. 8.5.4.1 (Local convergence of Newton's method)

$n = 1$:

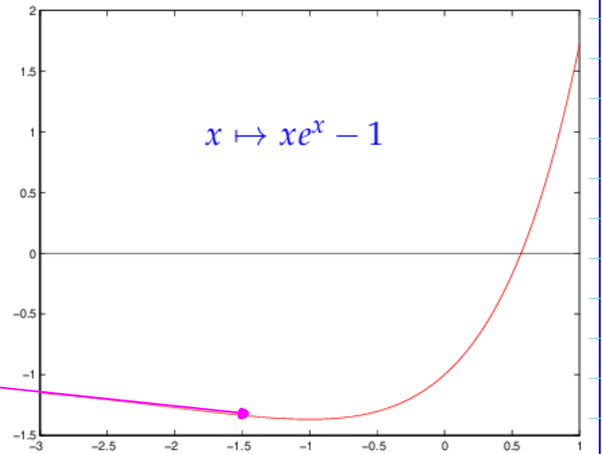
$F(x) = xe^x - 1 \quad \Rightarrow \quad F'(-1) = 0$

$x^{(0)} < -1 \quad \Rightarrow \quad x^{(k)} \to -\infty$,

$x^{(0)} > -1 \quad \Rightarrow \quad x^{(k)} \to x^*$,

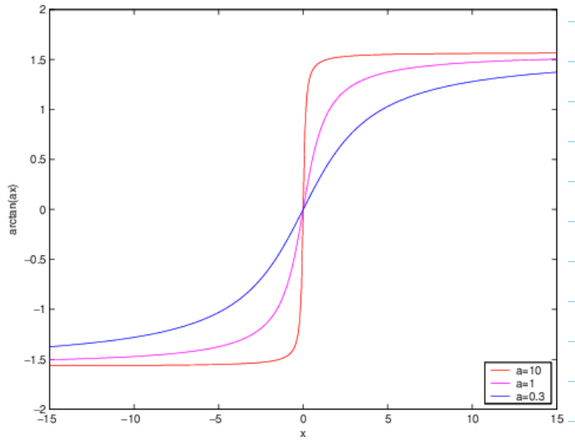
↳ Newton correct in the wrong direction

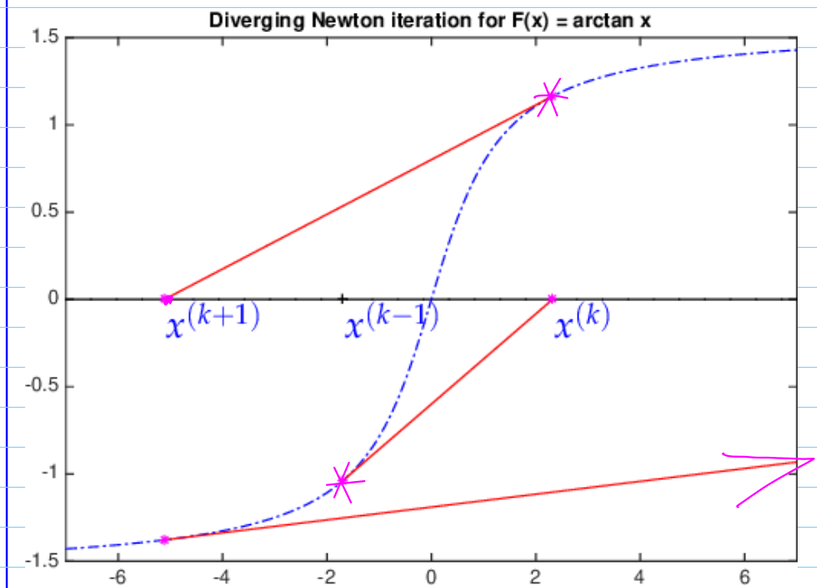
▷ no remedy



$F(x) = \arctan(ax)$, $a > 0, x \in \mathbb{R}$

with zero $x^* = 0$.





"Overshooting" Newton correction

we observe "overshooting" of Newton correction

Idea: damping of Newton correction:

With $\lambda^{(k)} > 0$: $\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} - \lambda^{(k)} D F(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)})$. (8.5.4.2)

Terminology: $\lambda^{(k)}$ = damping factor, $\lambda \in]0,1]$.

choosen by heuristics

↳ $x^{(k)}, \lambda^{(k)}$ not affected when replacing $F \longrightarrow AF$, $A \in \mathbb{R}^{n,n}$ regular

Affine invariant damping strategy

Choice of damping factor: affine invariant natural monotonicity test (NMT) :

choose "maximal" $0 < \lambda^{(k)} \leq 1$: $\left\| \Delta\bar{\mathbf{x}}(\lambda^{(k)}) \right\|_2 \leq (1 - \frac{\lambda^{(k)}}{2}) \left\| \Delta\mathbf{x}^{(k)} \right\|_2$ (8.5.4.4)

where $\Delta\mathbf{x}^{(k)} := D F(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)})$ → current Newton correction ,

$\Delta\bar{\mathbf{x}}(\lambda^{(k)}) := D F(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)} - \lambda^{(k)} \Delta\mathbf{x}^{(k)})$ → tentative simplified Newton correction .

↳ tentative next iterate

If quad. cvg. $\Rightarrow \|\Delta\bar{x}^{(k)}\| \leq \frac{1}{2} \|\Delta x^{(k)}\|$

If NMT fails $\Rightarrow \lambda \longleftarrow \lambda/2$ & retry

passed $\Rightarrow$ • (8.5.4.2), $k \longleftarrow k+1$

• $\lambda \longleftarrow 2\lambda$ in next step

A reject-accept approach

C++ code 8.5.4.5: Generic damped Newton method based on natural monotonicity test

```
template <typename FuncType,typename JacType,typename VecType>
void dampnewton(const FuncType &F,const JacType &DF,
                VecType &x,double rtol,double atol)
{
  using index_t = typename VecType::Index;
  using scalar_t = typename VecType::Scalar;
  const index_t n = x.size(); // No.  of unknowns
  const scalar_t lmin = 1E-3; // Minimal damping factor
  scalar_t lambda = 1.0; // Initial and actual damping factor
  VecType s(n),st(n); // Newton corrections
  VecType xn(n);      // Tentative new iterate
  scalar_t sn,stn;    // Norms of Newton corrections

  do {
    auto jacfac = DF(x).lu(); // LU-factorize Jacobian
    s = jacfac.solve(F(x));   // Newton correction
    sn = s.norm();            // Norm of Newton correction
    lambda *= 2.0;
    do {
      lambda /= 2; // Reduce damping factor
      if (lambda < lmin) throw "No convergence: lambda -> 0";
      xn = x-lambda*s;          // Tentative next iterate
      st = jacfac.solve(F(xn)); // Simplified Newton correction
      stn = st.norm();
    }
    while (stn > (1-lambda/2)*sn); // Natural monotonicity test
    x = xn; // Now:  xn accepted as new iterate
    lambda = std::min(2.0*lambda,1.0); // Try to mitigate damping
  }
  // Termination based on simplified Newton correction
  while ((stn > rtol*x.norm()) && (stn > atol));
}
```

reject

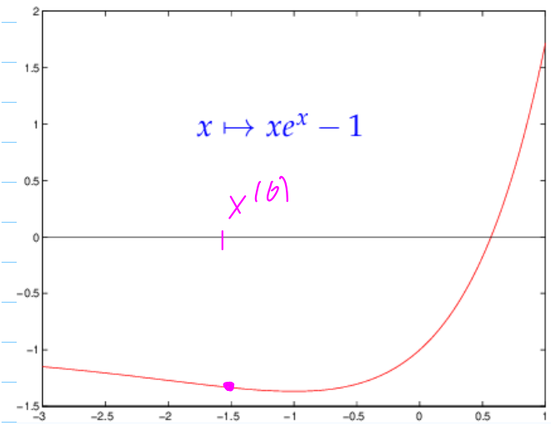
Exp. 8.5.4.6 (Damped Newton method)

$F(x) = \arctan(x)$,

- $x^{(0)} = 20$
- $q = \frac{1}{2}$
- `LMIN = 0.001`

k	$\lambda^{(k)}$	$x^{(k)}$	$F(x^{(k)})$
1	0.03125	0.94199967624205	0.75554074974604
2	0.06250	0.85287592931991	0.70616132170387
3	0.12500	0.70039827977515	0.61099321623952
4	0.25000	0.47271811131169	0.44158487422833
5	0.50000	0.20258686348037	0.19988168667351
6	1.00000	-0.00549825489514	-0.00549819949059
7	1.00000	0.00000011081045	0.00000011081045
8	1.00000	-0.00000000000001	-0.00000000000001

Exp. 8.5.4.7 (Failure of damped Newton method)



k	$\lambda^{(k)}$	$x^{(k)}$	$F(x^{(k)})$
1	0.25000	-4.4908445351690	-1.0503476286303
2	0.06250	-6.1682249558799	-1.0129221310944
3	0.01562	-7.6300006580712	-1.0037055902301
4	0.00390	-8.8476436930246	-1.0012715832278
5	0.00195	-10.5815494437311	-1.0002685596314

Bailed out because of `lambda < LMIN`!

$x^{(0)} = -1.5$

▷ Review questions 8.5.4.8

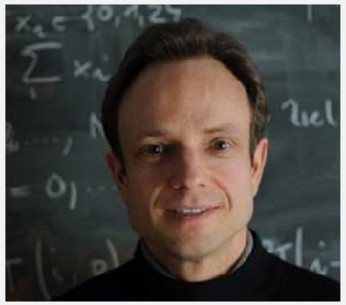
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 8.6: Quasi-Newton Method

Prof. R. Hiptmair, SAM, ETH Zurich

Date: December 6, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 8.5.1, Section 2.6

Duration: 27 minutes

Updated version compared to video

VIII Iterative Methods for Non-Linear Systems of Equations

8.6 Quasi-Newton Method

NLSE: $\underline{x}^* \in D : F(\underline{x}^*) = 0$

$F : D \subset \mathbb{R}^n \longrightarrow \mathbb{R}^n$

Drawback of Newton method: DF required

Goal: Derivative-free iteration of NLSE

Achieved for $n = 1$: Secant method, §4.2.3

$$x^{(k+1)} = x^{(k)} - \frac{x^{(k)} - x^{(k-1)}}{F(x^{(k)}) - F(x^{(k-1)})} F(x^{(k)})$$

N.I.: $x^{(k+1)} = x^{(k)} - \frac{1}{F'(x^{(k)})} F(x^{(k)})$

(2)

$n = 1$:

$DF(x^{(k)}) \longrightarrow \frac{F(x^{(k)}) - F(x^{(k-1)})}{x^{(k)} - x^{(k-1)}} =: J_k$

$n > 1$:

Replace $DF(x^{(k)})$ w/ "different quotient approximation" $J_k \in \mathbb{R}^{n,n}$

Idea: rewrite (8.6.0.1) as a **secant condition** for an approximation $\mathbf{J}_k \approx \mathrm{D}F(\mathbf{x}^{(k)})$, $\mathbf{x}^{(k)} \hat{=}$ the current iterate:

(SC)

$$\mathbf{J}_k(\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}) = F(\mathbf{x}^{(k)}) - F(\mathbf{x}^{(k-1)}) \,. \tag{8.6.0.2}$$

Iteration: $\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} - \mathbf{J}_k^{-1} F(\mathbf{x}^{(k)})$. (8.6.0.3)

does not fix J_k : Reuse J_{k-1} : $J_k \equiv J_{k-1}$ on "non-secant directions"

Broyden's conditions: $\mathbf{J}_k \mathbf{z} = \mathbf{J}_{k-1}\mathbf{z} \quad \forall \mathbf{z}: \mathbf{z} \perp (\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)})$. (BC) (8.6.0.4)

$$\mathbf{J}_k := \mathbf{J}_{k-1} + \frac{F(\mathbf{x}^{(k)})(\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)})^\top}{\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|_2^2} \tag{8.6.0.5}$$

(BC) + (SC) unique determine J_k

Final form of **Broyden's quasi-Newton method** for solving $F(\mathbf{x}) = 0$:

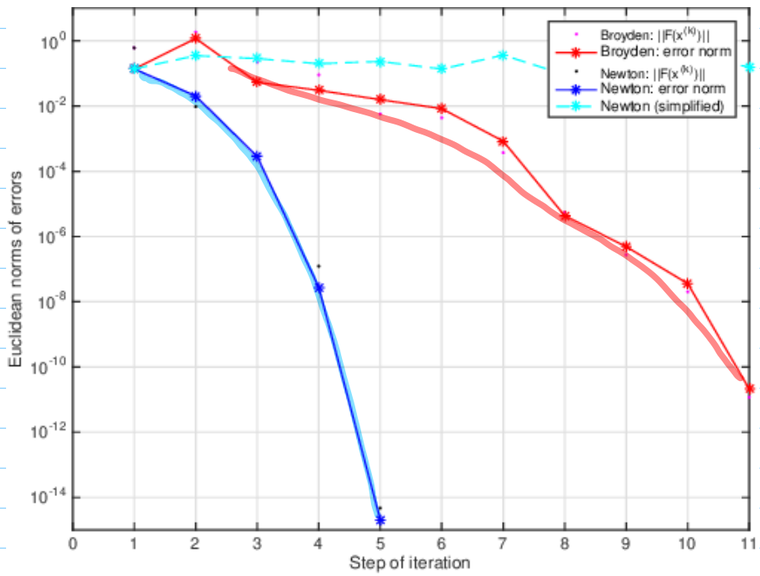
$$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)} \,, \quad \Delta\mathbf{x}^{(k)} := -\mathbf{J}_k^{-1} F(\mathbf{x}^{(k)}) \,,$$
$$\mathbf{J}_{k+1} := \mathbf{J}_k + \frac{F(\mathbf{x}^{(k+1)})(\Delta\mathbf{x}^{(k)})^\top}{\|\Delta\mathbf{x}^{(k)}\|_2^2} \,. \quad \approx DF(x^{(k)}) \tag{8.6.0.6}$$

req. : initial guess J_0 : $J_0 := DF(x^{(0)})$

Exp. 8.6.0.10 (Convergence of Broyden's method)

$$F(\mathbf{x}) = \begin{bmatrix} x_1^2 - x_2^4 \\ x_1 - x_2^3 \end{bmatrix} \,, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2 \quad \text{with solution} \quad F\left(\begin{bmatrix} 1 \\ 1 \end{bmatrix}\right) = 0 \,, \tag{8.5.2.2}$$

and take $\mathbf{x}^{(0)} = [0.7, 0.7]^T$. As starting value for the matrix iteration we use $\mathbf{J}_0 = \mathrm{D}F(\mathbf{x}^{(0)})$.



◁ Lin-log error plot

▷ No asymptotic quadratic cvg.

§ 8.6.0.15 (Implementation of Broyden's quasi-Newton method)

Final form of **Broyden's quasi-Newton method** for solving $F(\mathbf{x}) = 0$:

$$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)} \quad , \quad \Delta\mathbf{x}^{(k)} := -\mathbf{J}_k^{-1}F(\mathbf{x}^{(k)}) ,$$
$$\mathbf{J}_{k+1} := \mathbf{J}_k + \frac{F(\mathbf{x}^{(k+1)})(\Delta\mathbf{x}^{(k)})^\top}{\|\Delta\mathbf{x}^{(k)}\|_2^2} . \quad \hat{=} \text{ rank-1-modification} \tag{8.6.0.6}$$

Lemma 2.6.0.21. Sherman-Morrison-Woodbury formula

For regular $\mathbf{A} \in \mathbb{K}^{n,n}$, and $\mathbf{U}, \mathbf{V} \in \mathbb{K}^{n,k}$, $n, k \in \mathbb{N}$, $k \le n$, holds

$$(\mathbf{A} + \mathbf{U}\mathbf{V}^H)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{I} + \mathbf{V}^H\mathbf{A}^{-1}\mathbf{U})^{-1}\mathbf{V}^H\mathbf{A}^{-1} ,$$

if $\mathbf{I} + \mathbf{V}^H\mathbf{A}^{-1}\mathbf{U}$ *is regular.*

Use for $k = 1$:

$$(\mathbf{A} + \mathbf{u}\mathbf{v}^\top)^{-1} = \left(\mathbf{I} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^\top}{1 + \mathbf{v}^\top\mathbf{A}^{-1}\mathbf{u}}\right)\mathbf{A}^{-1} , \quad 1 + \mathbf{v}^\top\mathbf{A}^{-1}\mathbf{u} \neq 0 ,$$

with $\mathbf{u} := F(x^{(k+1)})$, $\mathbf{v} := \dfrac{\Delta x^{(k)}}{\|\Delta x^{(k)}\|_2^2}$

▷ Update formula for $\mathbf{J}_k^{-1}$

$$\mathbf{J}_{k+1}^{-1} = \left(\mathbf{I} - \frac{\mathbf{J}_k^{-1}F(\mathbf{x}^{(k+1)})(\Delta\mathbf{x}^{(k)})^T}{\|\Delta\mathbf{x}^{(k)}\|_2^2 + (\Delta\mathbf{x}^{(k)})^\top\mathbf{J}_k^{-1}F(\mathbf{x}^{(k+1)})}\right)\mathbf{J}_k^{-1} = \left(\mathbf{I} - \frac{\Delta\overline{\mathbf{x}}^{(k+1)}(\Delta\mathbf{x}^{(k)})^T}{\|\Delta\mathbf{x}^{(k)}\|_2^2 + (\Delta\mathbf{x}^{(k)})^\top\Delta\overline{\mathbf{x}}^{(k+1)}}\right)\mathbf{J}_k^{-1} , \tag{8.6.0.16}$$

↑ simplified quasi-Newton correction (→ useful for termination & damping)

Req. : $|\Delta x^{(k)\top}\cdot\Delta\overline{x}^{(k+1)}| < \|\Delta x^{(k)}\|_2^2$

$$\Delta\mathbf{x}^{(k)} = -\mathbf{J}_k^{-1}F(\mathbf{x}^{(k)}) = -\prod_{\ell=0}^{k-1}\left(\mathbf{I} - \frac{\Delta\overline{\mathbf{x}}^{(\ell+1)}(\Delta\mathbf{x}^{(\ell)})^T}{\|\Delta\mathbf{x}^{(\ell)}\|_2^2 + (\Delta\mathbf{x}^{(\ell)})^\top\Delta\overline{\mathbf{x}}^{(\ell+1)}}\right)\mathbf{J}_0^{-1}F(\mathbf{x}^{(k)}) .$$

↳ LU-factorize once : $O(n^3)$

$\mathbf{t}_0 \in \mathbb{R}^n$: $\mathbf{J}_0\mathbf{t}_0 = F(\mathbf{x}^{(k)})$, → 2 triangular solves $O(n^2)$

$$\mathbf{t}_{\ell+1} := \mathbf{t}_\ell - \Delta\overline{\mathbf{x}}^{(\ell+1)}\frac{(\Delta\mathbf{x}^{(\ell)})^T\mathbf{t}_\ell}{\|\mathbf{x}^{(\ell)}\|_2^2 + (\Delta\mathbf{x}^{(\ell)})^\top\Delta\overline{\mathbf{x}}^{(\ell+1)}} , \quad \ell = 0, \dots, k-1 , \quad \to O(kn) \tag{8.6.0.18}$$

$\Delta\mathbf{x}^{(k)} := -\mathbf{t}_k$,

Total cost $= O(n^3 + Nn^2 + N^2n)$ (N steps)

NumCSE@ETHZ

▷ Review questions 8.6.0.22

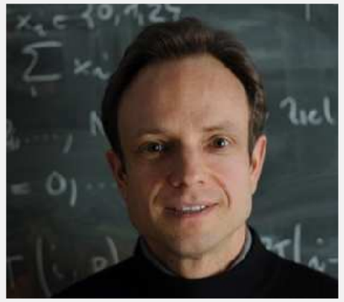
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 8.7: Non-linear Least Squares

Prof. R. Hiptmair, SAM, ETH Zurich

Date: December 6, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 3.1.1

Duration: 11 minutes

VIII Iterative Methods for Non-Linear Systems of Equations

8.7 Non-Linear Least Squares

LSE : $A\underline{x} = \underline{b}$, $A \in \mathbb{R}^{m,n}$

overdetermined $\leftarrow$ $m > n$

NLSE $F(\underline{x}) = 0$, $F : D \subset \mathbb{R}^n \longrightarrow \mathbb{R}^m$

overdetermined $\leftarrow$ $m > n$

▷ LSE : generalized concept of solutions

Definition 3.1.1.1. Least squares solution

For given $\mathbf{A} \in \mathbb{R}^{m,n}$, $\mathbf{b} \in \mathbb{R}^m$ the vector $\mathbf{x} \in \mathbb{R}^n$ is a least squares solution of the linear system of equations $\mathbf{Ax} = \mathbf{b}$, if

$$\mathbf{x} \in \operatorname*{argmin}_{\mathbf{y} \in \mathbb{R}^n} \|\mathbf{Ay} - \mathbf{b}\|_2^2 ,$$

$$\Updownarrow$$

$$\|\mathbf{Ax} - \mathbf{b}\|_2^2 = \min_{\mathbf{y} \in \mathbb{R}^n} \|\mathbf{Ay} - \mathbf{b}\|_2^2 = \min_{y_1,\ldots,y_n \in \mathbb{R}} \sum_{i=1}^{m} \left(\sum_{j=1}^{n} (\mathbf{A})_{i,j} y_j - (\mathbf{b})_i \right)^2 .$$

(2)

Definition 8.7.0.6. Non-linear least-squares solution

Given $F : D \subset \mathbb{R}^n \mapsto \mathbb{R}^m$, $m, n \in \mathbb{N}$, $m > n$, we call $\mathbf{x}^*$ a **non-linear least squares solution** of $F(\mathbf{x}) = \mathbf{0}$, if

$$\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}\in D} \|F(\mathbf{x})\|_2^2 .$$

Non-linear least squares problem

Given: $F : D \subset \mathbb{R}^n \mapsto \mathbb{R}^m$, $m, n \in \mathbb{N}$, $m > n$.

Find: $\mathbf{x}^* \in D$: $\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}\in D} \Phi(\mathbf{x})$, $\Phi(\mathbf{x}) := \frac{1}{2}\|F(\mathbf{x})\|_2^2$. (8.7.0.7)

Ex. 8.7.0.1 (Least squares data fitting)

Recall: Linear least-squares fitting (→ 5.7)

General Linear least squares fitting problem:

Given:
- data points $(\mathbf{t}_i, \mathbf{y}_i) \in \mathbb{R}^k \times \mathbb{R}^d$, $i = 1, \ldots, m$
- basis functions $\mathbf{b}_j : D \subset \mathbb{R}^k \mapsto \mathbb{R}$, $j = 1, \ldots, n$, $n < m$

Sought: coefficients $x_j \in \mathbb{R}$, $j = 1, \ldots, n$, such that

coefficients

$$\mathbf{x} := [x_1, \ldots, x_n]^\top := \operatorname{argmin}_{z_j \in \mathbb{R}^d} \sum_{i=1}^{m} \left\| \sum_{j=1}^{n} z_j \mathbf{b}_j(\mathbf{t}_i) - \mathbf{y}_i \right\|_2^2 . \quad (5.7.0.9)$$

Linear combination

Non-linear least squares fitting problem in 1D

Given:
- data points (t_i, y_i), $i = 1, \ldots, m$
- (symbolic formula) for parameterized function $f(x_1, \ldots, x_n; \cdot) : D \subset \mathbb{R} \mapsto \mathbb{R}$, $n < m$

Sought: parameter values $x_1^*, \ldots, x_n^* \in \mathbb{R}$ such that

$$(x_1^*, \ldots, x_n^*) = \operatorname{argmin}_{\mathbf{x}\in\mathbb{R}^n} \sum_{i=1}^{m} |f(x_1, \ldots, x_n; t_i) - y_i|^2 . \quad (8.7.0.3)$$

Linear case: $f(x_1, \ldots, x_n, t) = \sum_{j=1}^{n} x_j b_j(t)$

$\leftrightarrow$ overdet. NLSE $F(\underline{x}) = 0$

$$F(\mathbf{x}) := \begin{bmatrix} f(x_1, \ldots, x_n, t_1) - y_1 \\ \vdots \\ f(x_1, \ldots, x_n, t_m) - y_m \end{bmatrix} , \quad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} . \quad (8.7.0.4)$$

Fitting Gaussians: ($n = 3$)

$f(x_1, x_2, x_3, t) := x_1 e^{-x_2(t - x_3)}$

NumCSE@ETHZ

▷ Review questions 8.7.0.10

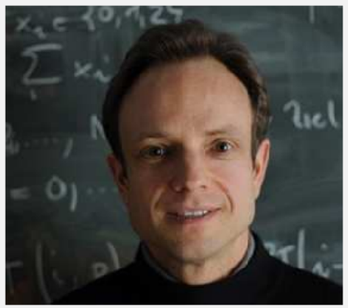
ETH Lecture 401-0663-00L Numerical Methods for CSE

Course Video
Section 8.7.1: Non-linear Least Squares: (Damped) Newton Method

Prof. R. Hiptmair, SAM, ETH Zurich

Date: December 6, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.
- Multi-dimensional differentiation

Dependencies. Section 8.5.1

Duration: 19 minutes

VIII Iterative Methods for Non-Linear Systems of Equations

8.7 Non-Linear Least Squares

Non-linear least squares problem

Given: $F : D \subset \mathbb{R}^n \mapsto \mathbb{R}^m$, $m, n \in \mathbb{N}$, $m > n$.

Find: $\mathbf{x}^* \in D$: $\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}\in D} \Phi(\mathbf{x})$, $\Phi(\mathbf{x}) := \frac{1}{2}\|F(\mathbf{x})\|_2^2$. (8.7.0.7)

Special case: Linear least-squares problem

$F(\underline{x}) = A\underline{x} - \underline{b} \Rightarrow \Phi(\underline{x}) = \|A\underline{x} - \underline{b}\|_2^2$

$A \in \mathbb{R}^{m,n}$, $\underline{b} \in \mathbb{R}^m$

8.7.1 (Damped) Newton Method

Assume : $F \in C^2$

$[\Phi : \mathbb{R}^n \to \mathbb{R}]$

$\Phi(\mathbf{x}^*) = \min \Rightarrow \underbrace{\mathbf{grad}\,\Phi(\mathbf{x}) = 0}_{n \times n \text{ NLSE}}$, where $\mathbf{grad}\,\Phi(\mathbf{x}) := \left[\frac{\partial \Phi}{\partial x_1}(\mathbf{x}), \ldots, \frac{\partial \Phi}{\partial x_n}(\mathbf{x})\right]^T \in \mathbb{R}^n$.

necessary condition

Newton iteration for $G(x) = 0$: $x^{(k+1)} = x^{(k)} - DG(x^{(k)})^{-1} G(x^{(k)})$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathrm{H}\Phi(\mathbf{x}^{(k)})^{-1}\,\mathbf{grad}\,\Phi(\mathbf{x}^{(k)}), \tag{8.7.1.2}$$

where $\mathrm{H}\Phi(\mathbf{x}) \in \mathbb{R}^{n,n}$ is the **Hessian matrix** of Φ

$$\mathrm{H}\Phi(\mathbf{x}) = \left[\frac{\partial^2 \Phi}{\partial x_i \partial x_j}\right]_{i,j=1}^{n} \in \mathbb{R}^{n,n}.$$

- grad Φ : $\Phi(x) = \frac{1}{2} F(x)^T \cdot F(x)$

Product rule: $F : D \subset V \mapsto W$, $G : D \subset V \mapsto U$ sufficiently smooth, $b : W \times U \mapsto Z$ bilinear, *ie.*, linear in each argument:

$$T(\mathbf{x}) := b(\mathrm{H}(\mathbf{x}), G(\mathbf{x})) \Rightarrow \mathrm{D}\,T(\mathbf{x})\mathbf{h} = b(\mathrm{DH}(\mathbf{x})\mathbf{h}, G(\mathbf{x})) + b(H(\mathbf{x}), \mathrm{D}\,G(\mathbf{x})\mathbf{h}), \quad \mathbf{h} \in V, \mathbf{x} \in D. \tag{8.5.1.17}$$

NumCSE@ETHZ

with $H = G := F$, $b(x, y) = \frac{1}{2} x^T \cdot y$

$D\,D\Phi(x)h = \frac{1}{2}(DF(x)h)^T F(x) + \frac{1}{2} F(x)^T (DF(x)h)$

↑ Jacobian

$= F(x)^T DF(x)\,h$

Definition 8.5.1.18. Gradient and Hessian

For sufficiently smooth $F : D \subset \mathbb{R}^n \mapsto \mathbb{R}$ the **gradient** $\mathbf{grad}\,F : D \mapsto \mathbb{R}^n$, and the **Hessian** (matrix) $\mathrm{H}F(\mathbf{x}) : D \mapsto \mathbb{R}^{n,n}$ are defined as

$\mathbf{grad}\,F(\mathbf{x})^T\mathbf{h} := \mathrm{D}F(\mathbf{x})\mathbf{h}$, $\mathbf{h}_1^T \mathrm{H}F(\mathbf{x})\mathbf{h}_2 := \mathrm{D}(\mathrm{D}F(\mathbf{x})(\mathbf{h}_1))(\mathbf{h}_2)$, $\forall \mathbf{h}, \mathbf{h}_1, \mathbf{h}_2 \in \mathbb{R}^n$.

$\Rightarrow$ $\mathrm{grad}\,\Phi(x) = DF(x)^T F(x)$

- $H\Phi$:

$\leftrightarrow$ $D\{x \in D \subset \mathbb{R}^n \longrightarrow F(x) DF(x) h_1 \in \mathbb{R}\}$

Again: product rule

$D\{x \longmapsto F(x)^T \cdot DF(x) h_1\}(x) h_2$

$= (DF(x)h_2)^T (DF(x)h_1) + F(x)^T \cdot D(DF(x)h_1)h_2$

$\to \in \mathbb{R}^{n,n}$ Jacobian

$$\mathrm{H}\Phi(\mathbf{x}) := \mathrm{D}(\mathbf{grad}\,\Phi)(\mathbf{x}) = \mathrm{D}F(\mathbf{x})^T\,\mathrm{D}F(\mathbf{x}) + \sum_{j=1}^{m} F_j(\mathbf{x})\,\mathrm{D}^2F_j(\mathbf{x})\,,$$

$$\Updownarrow \qquad (8.7.1.5)$$

$$(\mathrm{H}\Phi(\mathbf{x}))_{i,k} = \sum_{j=1}^{n}\left\{\frac{\partial^2 F_j}{\partial x_i \partial x_k}(\mathbf{x})F_j(\mathbf{x}) + \frac{\partial F_j}{\partial x_k}(\mathbf{x})\frac{\partial F_j}{\partial x_i}(\mathbf{x})\right\}.$$

▷ LSE for Newton correction $\underline{s} \in \mathbb{R}^n$

$$\underbrace{\left(\mathrm{D}F(\mathbf{x}^{(k)})^T\,\mathrm{D}F(\mathbf{x}^{(k)}) + \sum_{j=1}^{m} F_j(\mathbf{x}^{(k)})\,\mathrm{D}^2F_j(\mathbf{x}^{(k)})\right)}_{=\mathrm{H}\Phi(\mathbf{x}^{(k)})}\mathbf{s} = -\underbrace{\mathrm{D}F(\mathbf{x}^{(k)})^T F(\mathbf{x}^{(k)})}_{=\mathbf{grad}\,\Phi(\mathbf{x}^{(k)})}\,. \qquad (8.7.1.6)$$

Newton method $F(x) = Ax - b$

$A \in \mathbb{R}^{m,n}$, $b \in \mathbb{R}^m$?

⟶ Normal equation method!

NumCSE@ETHZ

Rem 8.7.1.7. (Newton method and minimization of quadratic functional)

Task: $\Phi(x) \to \min$, $\Phi : \mathbb{R}^n \to \mathbb{R}$

Idea: Taylor approximation at $\underline{x}^{(k)}$

$$\Phi(\mathbf{x}) \approx Q(\mathbf{s}) := \Phi(\mathbf{x}^{(k)}) + \mathbf{grad}\,\Phi(\mathbf{x}^{(k)})^T\mathbf{s} + \frac{1}{2}\mathbf{s}^T\mathrm{H}\Phi(\mathbf{x}^{(k)})\mathbf{s}\,, \qquad (8.7.1.8)$$

$$\mathbf{grad}\,Q(\mathbf{s}) = 0 \;\Leftrightarrow\; \mathrm{H}\Phi(\mathbf{x}^{(k)})\mathbf{s} + \mathbf{grad}\,\Phi(\mathbf{x}^{(k)}) = 0 \;\Leftrightarrow\; (8.7.1.4)\,.$$

▷ Review question 8.7.0.10
Review question 8.7.1.9

ETH Lecture 401-0663-00L Numerical Methods for CSE

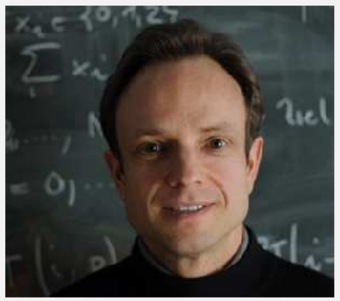
Course Video

Section 8.7.2 , Section 8.7.3: (Trust-region) Gauss-Newton Method

Prof. R. Hiptmair, SAM, ETH Zurich

Date: December 6, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 3.3.4

Duration: 22 minutes

VIII Iterative Methods for Non-Linear Systems of Equations

8.7 Non-Linear Least Squares

Non-linear least squares problem $\Leftrightarrow$ $F(x) = 0$

Given: $F : D \subset \mathbb{R}^n \mapsto \mathbb{R}^m$, $m, n \in \mathbb{N}$, $m > n$.

Find: $\mathbf{x}^* \in D$: $\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}\in D} \Phi(\mathbf{x})$, $\Phi(\mathbf{x}) := \frac{1}{2}\|F(\mathbf{x})\|_2^2$. (8.7.0.7)

8.7.2. Gauss-Newton Method

Idea: *Local linearization* of F: $F(\mathbf{x}) \approx F(\mathbf{y}) + \mathrm{D}F(\mathbf{y})(\mathbf{x}-\mathbf{y})$

➢ Leads to a sequence of *linear* least squares problems

⇓

iterative method

(2)

$\operatorname{argmin}_{x\in\mathbb{R}^n} \|F(x)\|_2$ is approximated by $\underbrace{\operatorname{argmin}_{x\in\mathbb{R}^n} \|F(x_0) + DF(x_0)(x - x_0)\|_2}_{(\spadesuit)}$,

$(\spadesuit) \Leftrightarrow \operatorname{argmin}_{x\in\mathbb{R}^n} \|Ax - b\|$ with $A := DF(x_0) \in \mathbb{R}^{m,n}$, [Jacobian]
$b := -F(x_0) + DF(x_0)x_0 \in \mathbb{R}^m$.

↑ linear least-squares problem

Gauss-Newton iteration (under full-rank assumption (8.7.0.9))

Initial guess $x^{(0)} \in D$

$$x^{(k+1)} := x^{(k)} - s \quad , \quad s := \operatorname{argmin}_{x\in\mathbb{R}^n} \left\| F(x^{(k)}) - DF(x^{(k)})x \right\|_2 . \tag{8.7.2.1}$$

Gauss-Newton correction

a linear least squares problem!

C++ code 8.7.2.2: Template for Gauss-Newton method

```
template <class Function, class Jacobian>
VectorXd gn(const Eigen::VectorXd &init, Function &&F, Jacobian &&J,
            double rtol = 1.0E-6, double atol = 1.0E-8) {
  Eigen::VectorXd x = init; // Vector for iterates x^(k)
  // Vector for Gauss-Newton correction s
  Eigen::VectorXd s = J(x).householderQr().solve(F(x)); //
  x = x - s;
  // A posteriori termination based on absolute and relative tolerances
  while ((s.norm() > rtol * x.norm()) && (s.norm() > atol)) { //
    s = J(x).householderQr().solve(F(x)); // --> linear LSQ
    x = x - s;
  }
  return x;
}
```

correction based

Rem. 8.7.2.3 (Gauss-Newton vs. Newton)

+ 2nd-derivative free
+ Larger domain of cvg.
+ local quadratic cvg.

Ex. 8.7.2.4 (Non-linear fitting of data)

Fitting functions $f(x_1, x_2, x_3, t) = x_1 + x_2 e^{-x_3 t}$

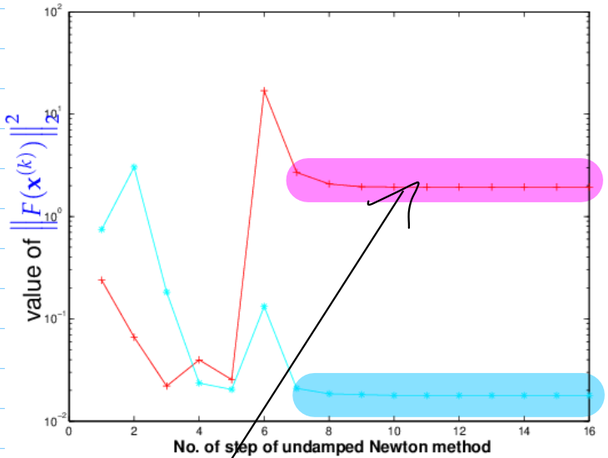
▷ Overdetermined NLSE $F(x) = 0$

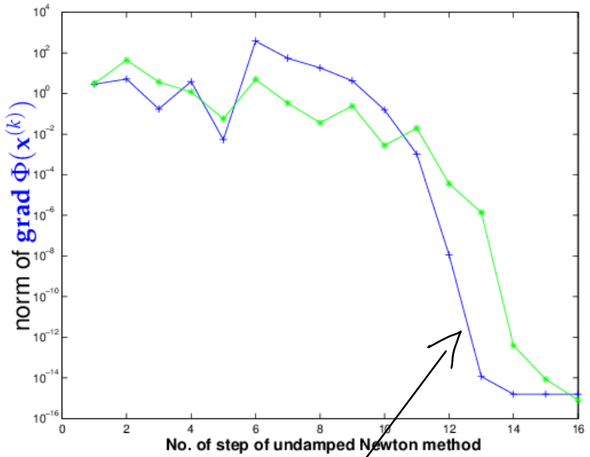
$$F : \mathbb{R}^3 \mapsto \mathbb{R}^m \quad , \quad F(x) := \begin{bmatrix} x_1 + x_2 \exp(-x_3 t_1) - y_1 \\ \vdots \\ x_1 + x_2 \exp(-x_3 t_m) - y_m \end{bmatrix} .$$

· 21 data points, "random"

- initial value $(1.8, 1.8, 0.1)^T$ (red curves, blue curves)
- initial value $(1.5, 1.5, 0.1)^T$ (cyan curves, green curves)

- Newton method for grad $\Phi(x) = 0$

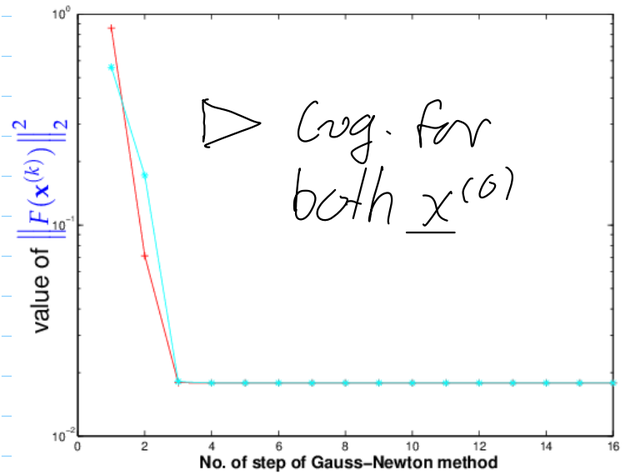




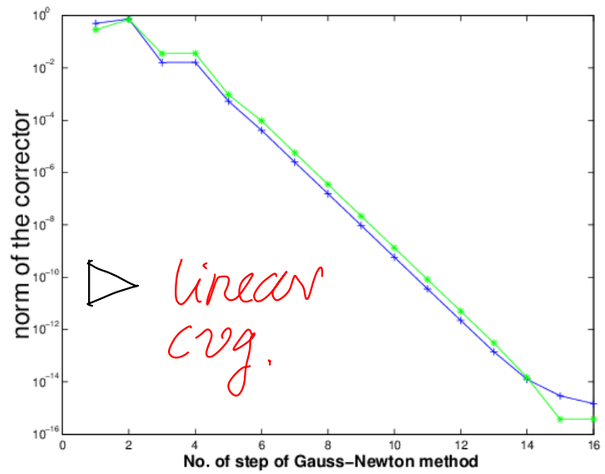
Newton it. stuck in local min.

quadratic cvg.

- Gauss-Newton method



▷ cvg. for both $x^{(0)}$



▷ linear cvg.

8.7.3. Trust Region Method

≙ Implicitly damped Gauss-Newton method

Idea: damping of the Gauss-Newton correction in (8.7.2.1) using a **penalty term**

instead of $\left\| F(\mathbf{x}^{(k)}) + \mathrm{D}F(\mathbf{x}^{(k)})\mathbf{s} \right\|_2^2$ minimize $\left\| F(\mathbf{x}^{(k)}) + \mathrm{D}F(\mathbf{x}^{(k)})\mathbf{s} \right\|_2^2 + \lambda \|\mathbf{s}\|_2^2$,

where $\lambda > 0$ is a penalty parameter.

$$\begin{cases} x^{(k+1)} = x^{(k)} - s \\ \mathbf{s} := \operatorname{argmin}_{\mathbf{z}\in\mathbb{R}^n} \left\| F(\mathbf{x}^{(k)}) + \mathrm{D}F(\mathbf{x}^{(k)})\mathbf{z} \right\|^2 + \lambda \|\mathbf{z}\|_2^2 \end{cases}$$

↳ Extended LSQ problem

Normal equations:

$$\left(\mathrm{D}F(\mathbf{x}^{(k)})^T \mathrm{D}F(\mathbf{x}^{(k)}) + \lambda \mathbf{I} \right) \mathbf{s} = -\mathrm{D}F(\mathbf{x}^{(k)}) F(\mathbf{x}^{(k)}) . \qquad (8.7.3.2)$$

$\lambda = ?$ Heuristics, $\lambda \to 0$ as $x^{(k)} \to x^*$

NumCSE@ETHZ

▷ Review questions 8.7.3.3.