Numerical Methods for Computational Science and Engineering

Resources

1. Gitlab: https://gitlab.math.ethz.ch/NumCSE/NumCSE
2. Eigen: https://eigen.tuxfamily.org/dox/index.html
3. LaTeX: https://oeis.org/wiki/List_of_LaTeX_mathematical_symbols
   (a) bold in vector: https://www.physicsread.com/latex-vector-bold/
   (b) LaTeX mathematical symbols

Table of contents

1 Computing with Matrices and Vectors

1.1 Fundamentals

Notations Vectors and Matrices are basic elements in numerical.

- Addressing vector components:
  - ✎ two notations: $\mathbf{x} = [x_1, \ldots, x_n]^\top \rightarrow x_i, \quad i = 1, \ldots, n$
    $\mathbf{x} \in \mathbb{K}^n \rightarrow (\mathbf{x})_i, \quad i = 1, \ldots, n$

- Addressing matrix entries & sub-matrices (✎ notations):

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

→ matrix entry/matrix element $(\mathbf{A})_{i,j} := a_{ij}$, $1 \le i \le n, 1 \le j \le m$,
→ i-th row, $1 \le i \le n$: $(\mathbf{A})_{i,:} := [a_{i,1}, \ldots, a_{i,m}]$,
→ j-th column, $1 \le j \le m$: $(\mathbf{A})_{:,j} := [a_{1,j}, \ldots, a_{n,j}]^\top$,
→ matrix block (sub-matrix) $(\mathbf{A})_{k:\ell,r:s} := \left[a_{ij}\right]_{\substack{i=k,\ldots,\ell \\ j=r,\ldots,s}}$, $1 \le k \le \ell \le n$, $1 \le r \le s \le m$.

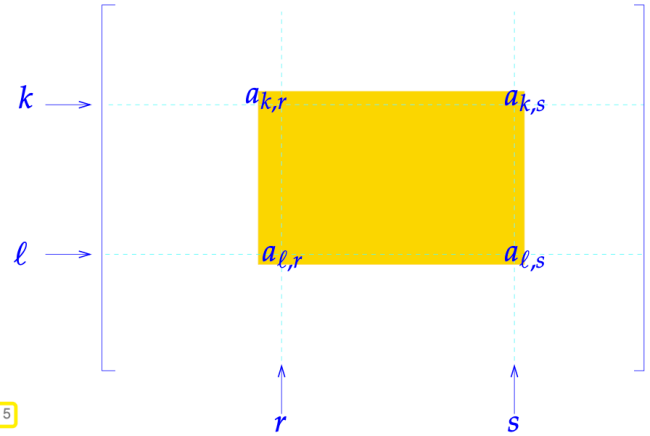


Fig. 5

The colon (:) range notation is inspired by MATLAB's/PYTHON's matrix addressing conventions. $(\mathbf{A})_{k:l,r:s}$ is a matrix of size $(l-k+1) \times (s-r+1)$.

Note that in PYTHON the : notation describes slightly different ranges: the end value is excluded.

1.2 Software and Libraries

1.2.1 Eigen

Eigen Where and how can we conduct these numerical methods.

- 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 : a special type of data type.

Define vector type and their using in Eigen

```
#include <Eigen/Dense>
```

```cpp
template <typename Scalar>
void eigentypedemo (unsigned int dim)
{
  using dynMat_t = Eigen::Matrix<Scalar, Eigen::Dynamic, Eigen::Dynamic>;
  using dynColVec_t = Eigen::Matrix<Scalar, Eigen::Dynamic, 1>;
  using dynRowVec_t = Eigen::Matrix<Scalar, 1, Eigen::Dynamic>;

  using index_t = typename dynMat_t::Index;
  using entry_t = typename dynMat_t::Scalar;

  dynColVec_t colvec(dim);
  dynRowVec_t rowvec(dim);

  //initialization through component access
  for(index_t i=0; i<colvec.size();++i) colvec[i]=(Scalar)i;
  for(index_t i=0; i<rowvec.size();++i) rowvec[i]=(Scalar)i/(i+1);
  colvec[0]=(Scalar)3.14;rowvec[dim-1]=(Scalar)2.718;

  dynMat vecprod = colvec*rowvec;  //matrix product
  const int nrows = vecprod.rows();
  const int nclos = vecprod.cols();
}
```

```
Eigen::Matrix  //Matrix class in Eigen namespace. It has six parameters, there of them
have default values. Defining these parameter can generate corresponding class.
//using dynMat_t = Eigen::Matrix<Scalar, Eigen::Dynamic, Eigen::Dynamic>;    create an
alias for easy access
//typedef Eigen::Matrix<Scalar, Eigen::Dynamic, Eigen::Dynamic> Matrix4x4;   create an
alias for easy access
```

Access submatrix

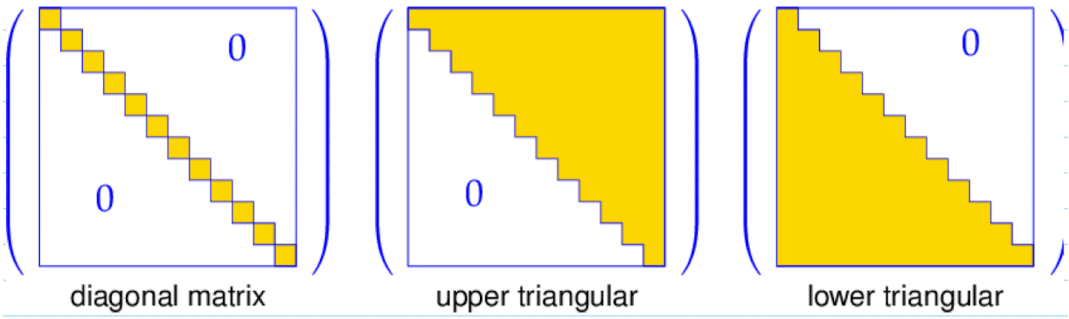
```cpp
#include <iostream>
#include <Eigen/Dense>
using namespace std;

template<typename MatType>
void blockAccess(Eigen::MatrixBase<MatType> &M)
{
  using index_t = typename Eigen::MatrixBase<MatType>::Index;
  using entry_t = typename Eigen::MatrixBase<MatType>::Scalar;
  const index_t nrows(M.rows()); // No. of rows
  const index_t ncols(M.cols()); // No. of columns
```

```
  cout << "Matrix M = " << endl << M << endl; // Print matrix
  // Block size half the size of the matrix
  index_t p = nrows/2,q = ncols/2;
  // Output submatrix with left upper entry at position (i,i)
  for(index_t i=0; i < min(p,q); i++)
    cout << "Block (" << i << ',' << i << ',' << p << ',' << q
         << ") = " << M.block(i,i,p,q) <<  endl;
  // l-value access: modify sub-matrix by adding a constant
  M.block(1,1,p,q) += Eigen::MatrixBase<MatType>::Constant(p,q,1.0);
  cout << "M = " << endl << M << endl;
  // r-value access: extract sub-matrix
  MatrixXd B = M.block(1,1,p,q);
  cout << "Isolated modified block = " << endl << B << endl;
  // Special sub-matrices
  cout << p << " top rows of m = " << M.topRows(p) << endl;
  cout << p << " bottom rows of m = " << M.bottomRows(p) << endl;
  cout << q << " left cols of m = " << M.leftCols(q) << endl;
  cout << q << " right cols of m = " << M.rightCols(p) << endl;
  // r-value access to upper triangular part
  const MatrixXd T = M.template triangularView<Upper>(); // \Label{ba:1}
  cout << "Upper triangular part = " << endl << T << endl;
  // l-value access to upper triangular part
  M.template triangularView<Lower>() *= -1.5; // \Label{ba:2}
  cout << "Matrix M = " << endl << M << endl;
}
```

```
    //using M.block(i,j,p,q) for matrix types -> (M)i+1:j+p,j+1:j+q
    //Eigen indexing from 0
```

- For index in Eigen, we must use `index_t`, rather than `int`.



- Ref: usage of using in c++; usage of typedef; usage of typename;

1.2.2 (Dense) Matrix Storage Formats

Motivation

- 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.

In this chapter, we only consider the storage format of dense matrix.

Two different storage format

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

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

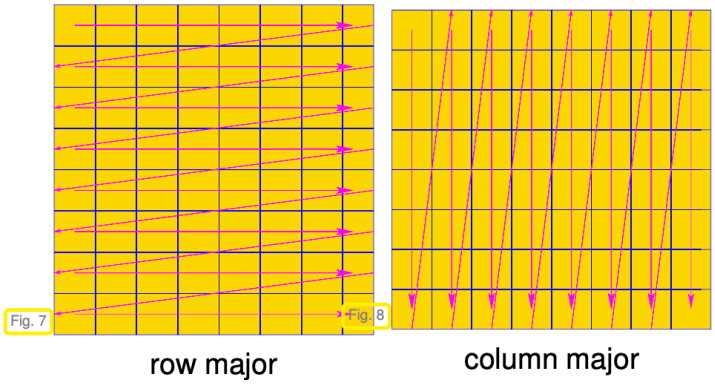
Access to entry $(\mathbf{A})_{ij}$ of $\mathbf{A} \in \mathbb{K}^{n,m}$, $i = 1,\ldots,n$, $j = 1,\ldots,m$:

row major:

$$(\mathbf{A})_{ij} \leftrightarrow \texttt{A\_arr(m*(i-1)+(j-1))}$$

column major:

$$(\mathbf{A})_{ij} \leftrightarrow \texttt{A\_arr(n*(j-1)+(i-1))}$$



- Due to the reason that the index is from 0, we need to subtract 1 when accessing these entries. Becareful~

Accessing entry in Eigen

We know that the entry in matrix object can be accessed by using `()`, eg `Eigen::MatrixXd mcn(3,3);` `mcn(0,0)=1;`. Apart from this, there are two different methods based on the storage format to access our entries.

1. `A(i)` ≡ reference for the i-th element of the array [closely related to the storage format (column-wise or raw-wise.]

```
using namespace Eigen;
void storageOrder(int nrows=6,int ncols=7)
{
  cout << "Different matrix storage layouts in Eigen" << endl;
  Matrix<double,Dynamic,Dynamic,ColMajor> mcm(nrows,ncols);
  Matrix<double,Dynamic,Dynamic,RowMajor> mrm(nrows,ncols);
  for (int l=1,i= 0; i< nrows; i++)
    for (int j= 0; j< ncols; j++,l++)
      mcm(i,j) = mrm(i,j) = l;

  cout << "Matrix mrm = " << endl << mrm << endl;
  cout << "mcm linear = ";
```

```cpp
    for (int l=0;l < mcm.size(); l++) cout << mcm(l) << ',';
    cout << endl;

    cout << "mrm linear = ";
    for (int l=0;l < mrm.size(); l++) cout << mrm(l) << ',';
    cout << endl;
}
```

```
Different matrix storage layouts in Eigen
Matrix mrm =
1 2 3
4 5 6
7 8 9
mcm linear = 1,4,7,2,5,8,3,6,9,
mrm linear = 1,2,3,4,5,6,7,8,9,
```

2. `A.data()` ≡ pointer [Documentation]

Example

```cpp
Matrix<int, 3, 4, ColMajor> Acolmajor;
Acolmajor << 8, 2, 2, 9,
             9, 1, 4, 4,
             3, 5, 4, 5;
cout << "The matrix A:" << endl;
cout << Acolmajor << endl << endl;

cout << "In memory (column-major):" << endl;
for (int i = 0; i < Acolmajor.size(); i++)
  cout << *(Acolmajor.data() + i) << "  ";
cout << endl << endl;

Matrix<int, 3, 4, RowMajor> Arowmajor = Acolmajor;
cout << "In memory (row-major):" << endl;
for (int i = 0; i < Arowmajor.size(); i++)
  cout << *(Arowmajor.data() + i) << "  ";
cout << endl;
```

Output

```
The matrix A:
8 2 2 9
9 1 4 4
3 5 4 5

In memory (column-major):
8  9  3  2  1  5  2  4  4  9  4  5

In memory (row-major):
8  2  2  9  9  1  4  4  3  5  4  5
```

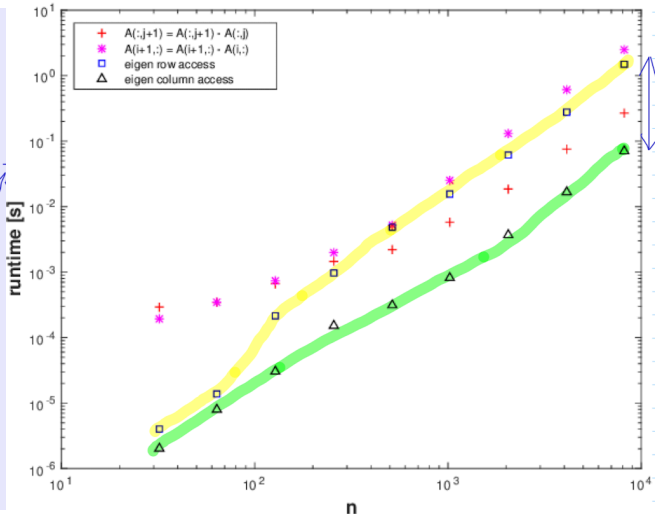
Data layout impacts efficiency

Although different layouts won't affect the access of these entries for users, it will impacts the efficiency.

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

```cpp
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);   -> declare matrix obj
    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;
}
```



- **Results:** In this code, we didn't use different storage format. we just consider the access speed for column-wise and row-wise. Result shows that row-wise (→incur cache missing) are much solver than col-wise access.
- **Conclusion:** elements of all columns are contiguously in memory. more efficient!

```
typedef Matrix< double, Dynamic, Dynamic >  Eigen::MatrixXd //Dynamic×Dynamic matrix
of type double.
```

- Eigen defines several typedef shortcuts for most common matrix and vector types. documentation

1.3 Computatioanl Effort

In numerical analysis, we often need to characterize the performance a prposed algorithm. The main idea is that calculating the number of elementary operations needed in this algorithm and regard it as the creteria. But we know that for different size of problems, the number of elementary operations is different. In order to build a universal creteria for judging an algorithm, the asymptotic complexity is proposed to chatacterize the problem tend to ∞.

Definition computational effort

Definition 1.4.0.1. Computational effort

The **computational effort**/**computational cost** required/incurred by a numerical code amounts to the number of elementary operations (additions,subtractions,multiplications,divisions,square roots) executed in a run.

- Traditional definition: Number of elementary operations $+, -, *, \sqrt{s}$

Fifty years ago counting elementary operations provided good predictions of runtimes, but nowadays this is no longer true.

"Computational effort $\not\sim$ 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
  - Pattern of memory access
  - Vectorization/ pipelining

Asymptotic complexity

Definition 1.4.1.1. (Asymptotic) complexity

The **asymptotic (computational) 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 ∞.

- *Problem size parameters* in numerical linear algebra usually are the lengths and dimensions of the vectors and matrices that an algorithm takes as inputs.
- *Worst case* indicates that the maximum effort over a set of admissible data is taken into account.
- **Notation:** "Landau-0". $Cost(n) = O(n^{\alpha}), \alpha > 0 \; for \; n \rightarrow \infty$
  - Meaning: Asymptic complexity predicts the dependence of runtime of problem size for

- large problems
- a concrete implementation

– e.g.: $cost(n) = O(n^2)$; then $n \to 2*n$ leads to $cost \to 4*cost$

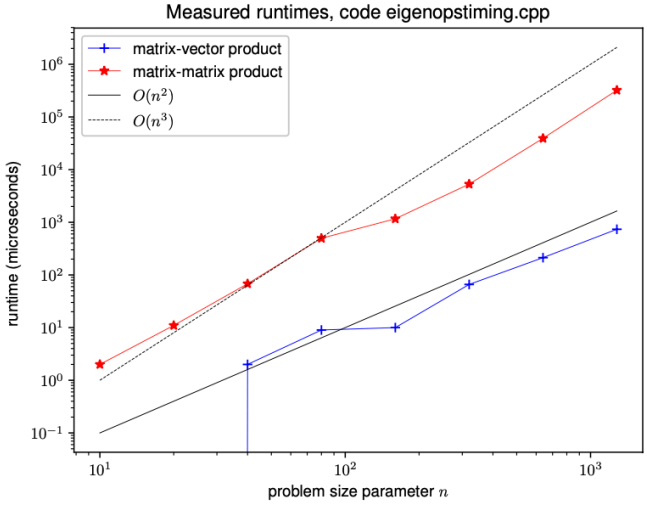
Computational cost of basic numerical LA operations

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	$n-1$	$O(n)$
tensor product	$(\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n) \mapsto \mathbf{x}\mathbf{y}^H$	nm	0	$O(mn)$
Matrix×vector	$(\mathbf{x} \in \mathbb{R}^n, \mathbf{A} \in \mathbb{R}^{m,n}) \mapsto \mathbf{A}\mathbf{x}$	mn	$(n-1)m$	$O(mn)$
matrix product(*)	$(\mathbf{A} \in \mathbb{R}^{m,n}, \mathbf{B} \in \mathbb{R}^{n,k}) \mapsto \mathbf{A}\mathbf{B}$	mnk	$mk(n-1)$	$O(mnk)$

- For matrix product, triple-loop implementation (faster implementation available $O(n^{2.36})$, if Remark 1.4.2.2 in lecture notes)

Experiments: Runtimes of elementary linear algebra operations in Eigen

implementation based on Eigen, m = n



- A doubly logarithmic plot.
  – [If $cost(n) = O(n^\alpha) \Rightarrow$ data points aligned in a doubly logarithmic plot, the slope is determined by α .]

Some tricks to improve complexity

In computations involving matrices and vectors complexity of algoritms can often be reduced by performing the operations in a particular order.

Example: Exploit associativity

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

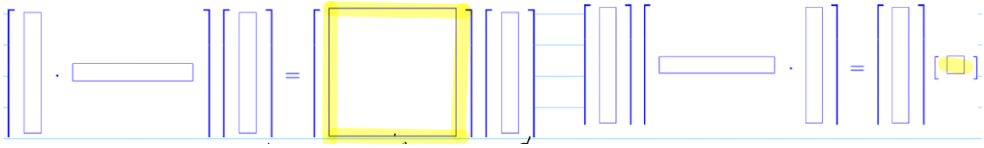
$$\mathbf{y} = \left(\mathbf{a}\mathbf{b}^\top\right)\mathbf{x}\,. \qquad (1.4.3.2) \qquad\qquad \mathbf{y} = \mathbf{a}\left(\mathbf{b}^\top\mathbf{x}\right)\,. \qquad (1.4.3.3)$$

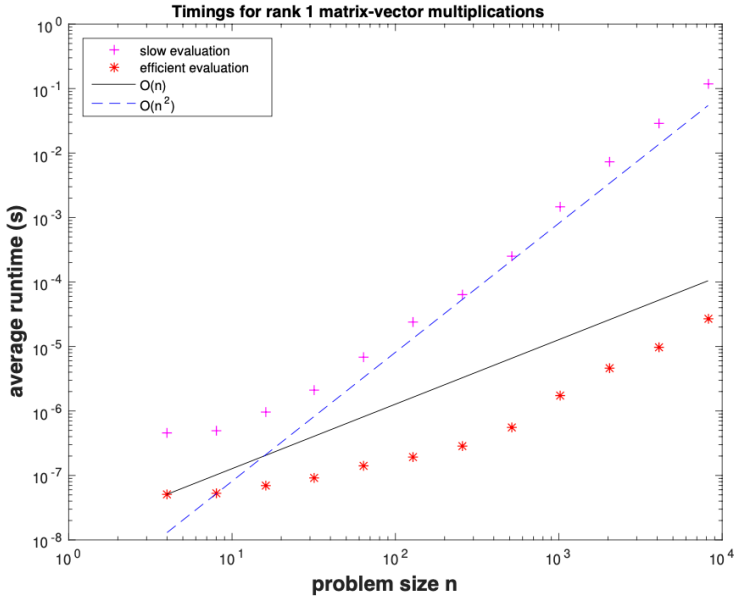
```
T = (a*b.transpose())*x;
```

➤ complexity $O(mn)$

```
t = a*b.dot(x);
```

➤ complexity $O(n+m)$ (“linear complexity”)





Example: Hidden summation

For $\mathbf{A},\ \mathbf{B} \in \mathbb{R}^{\mathbf{n},\mathbf{p}}, \mathbf{p} \ll \mathbf{n},\ \mathbf{x} \in \mathbb{R}^{\mathbf{n}}$, asymptotic complexity of the following code in Eigen:

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

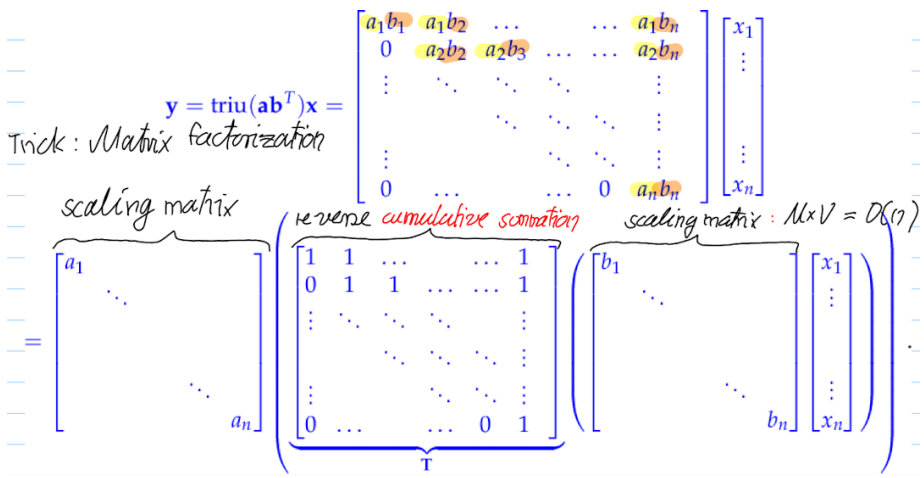
→ $O(n^2p)$ → $O(n^2)$ } → $O(n^2p)$

$$\mathbf{y} = \mathbf{triu}(\mathbf{A}\mathbf{B}^{\mathbf{T}})\mathbf{x} \tag{1}$$

- **triu** means upper triangular part

⇓ Can we do better?

For case $\mathbf{p} = \mathbf{1}$: $\mathbf{A},\ \mathbf{B}\ \leftrightarrow \mathbf{a},\ \mathbf{b}\ \in \mathbb{R}^{\mathbf{n}}$. $(\mathbf{a}\mathbf{b}^{\mathbf{T}})_{\mathbf{i},\mathbf{j}} = \mathbf{a}_\mathbf{i}\mathbf{b}_\mathbf{j}, \mathbf{1} \le \mathbf{i},\mathbf{j} \le \mathbf{n}$



Luckily, we have effecient algorithm for

$$\begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix} \mathbf{x} \tag{2}$$

- $cost(n) = O(n)$; Total cost $= O(n)$.

For case $\mathbf{p > 1}$:

Trick: $\mathbf{AB^T} = \sum_{i=1}^{p} \underbrace{(\mathbf{A})_{:,i}(\mathbf{B})_{:,i}^T}_{\mathbb{R}^{n,n}}$. Trip-operation is linear: $\text{triu}(\mathbf{AB^T} = \sum_{i=1}^{p} \text{triu}((\mathbf{A})_{:,i}(\mathbf{B})_{:,i}^T))$

$$\text{triu}(\mathbf{AB^T}) = \sum_{i=1}^{p} \text{triu}((\mathbf{A})_{:,i}(\mathbf{B})_{:,i}^T)\mathbf{x} \tag{3}$$

- total cost $O(np)$

Example: 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

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

↑ matrix-block notation

Task: given $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n,n}$, $\mathbf{x} \in \mathbb{R}^{n^2}$ we want to compute $\mathbf{y} = \underbrace{(\mathbf{A} \otimes \mathbf{B})}_{\in \mathbb{R}^{n^2,n^2}}\mathbf{x}$. [Naive implementation: cost $= O(n^4)$]

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

with partitioned vector $\mathbf{x} = [\mathbf{x}^1\ \mathbf{x}^2\ \dots\ \mathbf{x}^l]^T$, $\mathbf{x}^l \in \mathbb{R}^n$

Idea: precompute $\mathbf{Bx^l}$, $\mathbf{l = 1, \dots, n}$

$\mathbf{Z = B \cdot \underbrace{[x^1, \dots, x^n]}_{\in \mathbb{R}^{n,n}} = [Bx^1, \dots, Bx^n]}$ → requires $O(n^3)$ operations

$(\mathbf{A})_{1,1}\mathbf{Bx^1} + (\mathbf{A})_{1,2}\mathbf{Bx^2} + \cdots + (\mathbf{A})_{1,n}\mathbf{Bx^n} = \mathbf{Z}(\mathbf{A})_{1,:}^T \rightarrow \mathbf{Form : y = ZA^T \rightarrow cost\ O(n^3)}$

Results vector: $\mathbf{y} = [(\mathbf{y})_{:,1}\ \dots\ (\mathbf{y})_{:,n}]^T \in \mathbb{R}^{n^2}$ 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

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

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

1.4 Machine Arithmetic and Consequences

1.4.1 Experiment: Loss of Orthogonality

Let's use the following example to introduce the machine arithmetic.

Gram-Schmidt orthonormalization

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

1: $\mathbf{q}^1 := \frac{\mathbf{a}^1}{\|\mathbf{a}^1\|_2}$ *% 1st output vector*
2: **for** $j = 2,\dots,k$ **do**
{ *% Orthogonal projection*
3: $\mathbf{q}^j := \mathbf{a}^j$
4: **for** $\ell = 1,2,\dots,j-1$ **do** (GS)
5: $\{\ \mathbf{q}^j \leftarrow \mathbf{q}^j - \mathbf{a}^j \cdot \mathbf{q}^\ell \mathbf{q}^\ell\ \}$
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: $\{\mathbf{q}^1,\dots,\mathbf{q}^j\}$

✎ Notation: $\|\cdot\|_2 \triangleq$ Euclidean norm of a vector $\in \mathbb{K}^n$

In linear algebra we have learnt that, if it does not STOP prematurely, this algorithm will compute orthonormal vectors $\mathbf{q}^1,\dots,\mathbf{q}^k$ satisfying

$$\mathrm{Span}\{\mathbf{q}^1,\dots,\mathbf{q}^\ell\} = \mathrm{Span}\{\mathbf{a}^1,\dots,\mathbf{a}^\ell\}\,, \tag{1.5.1.2}$$

for all $\ell \in \{1,\dots,k\}$.

More precisely, if $\mathbf{a}^1,\dots,\mathbf{a}^\ell$, $\ell \le k$, are linearly independent, then the Gram-Schmidt algorithm will not terminate before the $\ell+1$-th step.

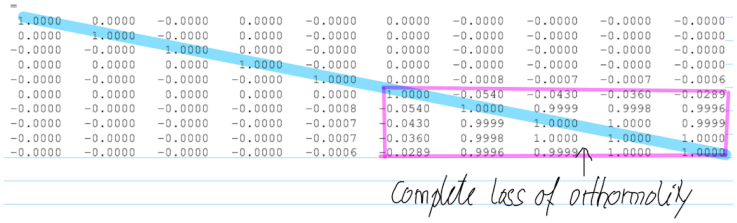
- $\mathbf{Q} := [\mathbf{q}^1\ \dots\ \mathbf{q}^j]$ If no stop $(j = k)$ then $\mathbf{Q}^T\mathbf{Q} = \mathbf{I} \Leftrightarrow \{\mathbf{q}^j\}_{j=1}^k$ is orthonormal
- $\mathrm{span}\{\mathbf{q}^1,\ \dots,\mathbf{q}^l\} = \mathrm{span}\{\mathbf{a}^1,\ \dots,\mathbf{a}^l\}$

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

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

We use above algorithm to run for Hilbert Matrix $\mathbf{A} = [\frac{1}{i+j-1}]_{i,j=1}^n \in \mathbb{R}^{n,n}, n = 10$

$\Rightarrow \mathbf{Q}^T\mathbf{Q} =$



- From the result, we can see that it is unreasonable. Let's explain it in following chapter.

1.4.2 Machine Number + Roundoff Errors

Definition

The reason, why computers must fail to execute exact computations with real numbers is clear:

Computer = finite automaton ➢ can handle only *finitely many* numbers, not $\mathbb{R}$

machine numbers, set $\mathbb{M}$

Essential property: $\mathbb{M}$ is a finite, discrete subset of $\mathbb{R}$ (its numbers separated by gaps)

- Different from real number, $\mathbb{R}$. In computer/numerical analysis, we introduce machine numbers, $\mathbb{M}$.
- And computers compute in $\mathbb{M}$, Which will give rise to following outcomes due to its properties:
  - **Finite:** overflow/underflow
  - **Discrete in $\mathbb{R}$:** Roundoff errors (1.5.3). This is the essential properties in numercial analysis!
    - op $\equiv$ elementary arithmetic operation $\in \{+, -, *, /\}$
    - op: $\mathbb{M} \times \mathbb{M} \nrightarrow \mathbb{M}$
    - In order to avoid this problem, in real situation, hardware implementation replace op with $\widetilde{\text{op}}$. $\widetilde{\text{op}} \equiv \text{rd} \cdot \text{op}$ [Find the machine number nearest the solutions.]

      Definition 1.5.3.6. Correct rounding

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

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

    - The approximation $\widetilde{\text{op}}$ will give rise to the error. However, the good news is that: Relative error of $\widetilde{\text{op}}$ can be controlled.
      -

      $$\text{rel. err.} : \frac{|\widetilde{\text{op}}(x,y) - \text{op}(x,y)|}{|\text{op}(x,y)|} = \frac{|(\text{rd-1})\text{op}(x,y)|}{|\text{op}(x,y)|} \equiv \text{relative error of rd} \tag{4}$$

      - Guaranteed: $\underbrace{\max}_{x \in |\mathbb{M}|} \frac{|\text{rd}(x) - x|}{|x|} \equiv \text{EPS} \approx 2.2 \cdot 10^{-16}$ for double [Can be controlled to improve the precision of the computer, which shrink the distance between adjacent machine numbers]

        $|\mathbb{M}| = [\min\{|x|, x \in \mathbb{M}\}, \max\{|x|, x \in \mathbb{M}\}] \subset \mathbb{R} \Leftrightarrow \text{rd}(x) = x(1+\varepsilon), \text{ with} |\varepsilon| \leq \text{EPS}$

        we refer EPS as machine precision (of $\mathbb{M}$)

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 $|\delta| < \text{EPS}$.

- This means that all roundoff is controlled by the machine precision. We can use above formula to analyze the roundoff in our algorithm. Notice that the concrete algorithm will contain numerious op, the roundoff error will accumulate during the process. Advanced topics about roundoff analysis won't discuss in the lecture.

Remarks about 1.5.1

Let's come back to the chapter 1.5.1, we have two remarks about it.

Remarks: 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 }
```

$Q^TQ =$

```
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
```

- In this new code, we try a different method (line 11 12) to calculate the **Q** matrix. And the result is perfect.
- This tells us different algorithms have different sensitivity to roundoff.

Remarks: Translate the stopping rule in numerical codes

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

- Due to the roundoff error, the final result cannot end with $q^j = 0$. We must replace it with a relative smallness.

1.4.3 Cancellation

Why worry about EPS-sized errors? ⇒ Because of error amplification

Examples of cancellation

Examples: Zeros of a quadratic polynomial → based on discriminant formula

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

```
//!  C++ function computing the zeros of a quadratic polynomial
//!  ζ → ζ^2 + αζ + β by means of the familiar discriminant
//!  formula ζ_{1,2} = ½(−α ± √(α^2 − 4β)).  However
//!  this implementation is vulnerable to round-off!  The zeros are
//!  returned in a column vector
Vector2d zerosquadpol(double alpha, double beta) {
  Vector2d z;
  double D = std::pow(alpha, 2) - 4 * beta; // discriminant
  if (D < 0)
    throw "no real zeros";
  else {
    // The famous discriminant formula
    double wD = std::sqrt(D);
    z << (-alpha - wD) / 2, (-alpha + wD) / 2; //
  }
  return z;
}
```

Apply above algorithm to this: $p(\xi) = \xi^2 \underbrace{-(\gamma + 1/\gamma)}_{=\alpha}\xi + \underbrace{1}_{\equiv\beta} = (\xi - \gamma)(\xi - 1/\gamma)$

Plot of relative errors Def. 1.5.3.3 ▷

We observe that roundoff incurred during the computation of α and β leads to "wrong" roots.
For large γ the computed small root may be fairly inaccurate as regards its *relative error*, which can be several orders of magnitude larger than machine precision EPS.

The large root always enjoys a small relative error about the size of EPS.

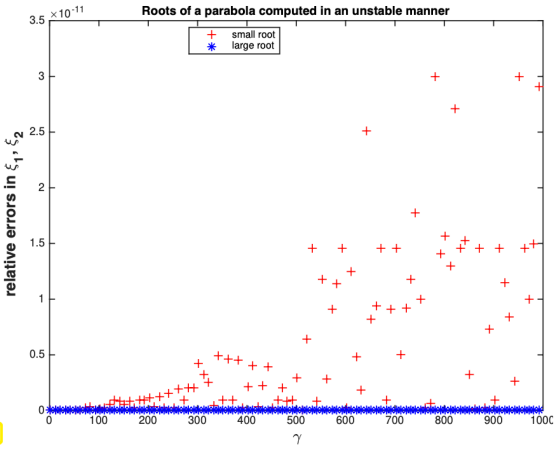
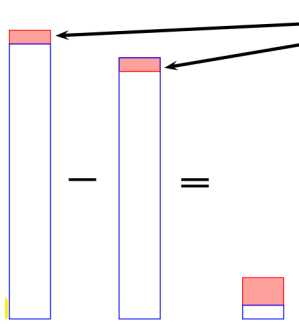


Fig. 21

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

In order to understand this phenomena, let's consider: Cancellation: Extreme amplification of relative error during the subtraction of numbers of equal size.



Cancellation

≙ Subtraction of almost equal numbers

(➤ extreme *amplification* of *relative* errors)

(◁ Roundoff error introduced by subtraction itself is negligible.)

- We look at the *exact* subtraction of two almost equal positive numbers both of which have small relative errors (red boxes) with respect to some desired exact value (indicated by blue boxes). The result of the subtraction will be small, but the errors may add up during the subtraction, ultimately constituting a large fraction of the result.

Another condition:

EXAMPLE 1.5.4.6 (Cancellation in decimal system) We consider two positive numbers x, y of about the same size afflicted with relative errors $\approx 10^{-7}$. This means that their seventh decimal digits are perturbed, here indicated by $*$. When we subtract the two numbers the perturbed digits are shifted to the left, resulting in a possible relative error of $\approx 10^{-3}$:

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

padded zeroes

- When discussing about cancellation, we often refer to relative errors. As above, the absolute error may be unchanged, but the relative will greatly imporve due to the nearly zero values after subtraction.

↓ Now, let's go back to the discriminant formula

When $\gamma \gg 1 \Rightarrow \alpha \gg 1$ *and* $\beta = 1 \Rightarrow \sqrt{D} \approx \alpha$ Cancellation will happen here: `(-alpha + wD)/2;`

Examples: Cancellation in different quotients

$f : \mathbb{R} \to \mathbb{R} : f'(x) = \frac{f(x+h)-f(x)}{h}$ *for* $h \ll 1 \Rightarrow$ *Cancellation in numerictor for* $h \approx 0$

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

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

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 $h \approx \sqrt{\text{EPS}}$ [This is the minimal relative error that can be achieved in this computer!]

Examples: Cancellation & Orthogonalization

Cancellation when computing orthogonal projection of vector **a** onto space spanned by vector **b** ▷

$$\mathbf{p} = \mathbf{a} - \frac{\mathbf{a}\cdot\mathbf{b}}{\mathbf{b}\cdot\mathbf{b}}\mathbf{b}\,.$$

If **a**, **b** point in almost the same direction, $\|\mathbf{p}\| \ll \|\mathbf{a}\|, \|\mathbf{b}\|$, so that a "tiny" vector **p** is obtained by subtracting two "long" vectors, which implies cancellation.

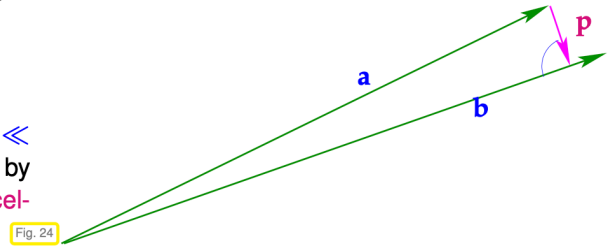


Fig. 24

- The reason for failure of naive Gram-Schmidt orthogonalisation

Avoiding cancellation

Example 1: Stable discriminant formula

$p(\xi) = \xi^2 + \alpha\xi + \beta$ with roots $\xi_1, \xi_2 \in \mathbb{R}$ vieta's theorem: $\xi_1 \cdot \xi_2 = \beta,\ |\xi_2| > |\xi_1|$

Idea: 1. Compute large root (no cancellation) 2. $\xi_1 = \beta/\xi_2$

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;
}
```

Example 2: Recasting Expression

- $\int_0^x \sin(t)dt = 1 - \cos(x) = 2\sin^2(x/2)$ $1 - \cos(x) \to$ cancellation for $x \approx 0$ $2\sin^2(x/2) \to$ no cancellation

$$\begin{aligned} y &= \sqrt{1+x^2} - \sqrt{1-x^2} \leftarrow \text{cancellation for } x \approx 0 \\ &= (\sqrt{1+x^2} - \sqrt{1-x^2})(\sqrt{1+x^2} + \sqrt{1-x^2})/(\sqrt{1+x^2} + \sqrt{1-x^2}) \\ &= 2x^2/(\sqrt{1+x^2} + \sqrt{1-x^2}) \to \text{no cancellation} \end{aligned} \tag{5}$$

- – NOTE: cancellation is harmless for $x \approx 1$

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

Example 3: Trading cancellation for approximation

$$I(a) = \int_0^1 e^{at}dt = \frac{e^a - 1}{a} \leftarrow \text{cancellation for } a = 1 \tag{6}$$

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

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

$$f(x_0 + h) = \sum_{k=0}^{m} \frac{f^{(k)}(x_0)}{k!} h^k + \underbrace{\frac{1}{(m+1)!} f^{(m+1)}(\xi) h^{m+1}}_{\text{remainder term, } \xi \in]x_0, x_0+h[} \tag{7}$$

Apply with $f(x) = e^x$, $x_0 = 0$, $h - a$ $\qquad$ $f^{(k)}(x) = e^x$

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

$\tilde{I}_m(a) = \sum_{k=0}^{m} \frac{1}{(k+1)!} a^k$:an approximation of $I(a)$

THEN, the key question is how to choose m? $\qquad$ Goal: Relative function error $\ll$ tol

LUCKILY, we can calculate the error by the remainder term:

$$\text{rel. err.} = \frac{\left| I(a) - \tilde{I}_m(a) \right|}{|I(a)|} = \frac{(e^a - 1)/a - \sum_{k=0}^{m} \frac{1}{(k+1)!} a^k}{(e^a - 1)/a}$$
$$\le \frac{1}{(m+1)!} \exp(\xi) a^m \le \frac{1}{(m+1)!} \exp(a) a^m .$$

For $a = 10^{-3}$ we get

m	1	2	3	4	5
	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
- Hence, keeping $m = 3$ terms is enough for achieving about 10 valid digits.

Relative error of unstable formula `(exp(a)-1.0)/a` and relative error, when using a Taylor expansion approximation for small a▷

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

Error computed by comparison with the PYTHON library function `numpy.expm1()` that provides a stable implementation of $\exp(x) - 1$.

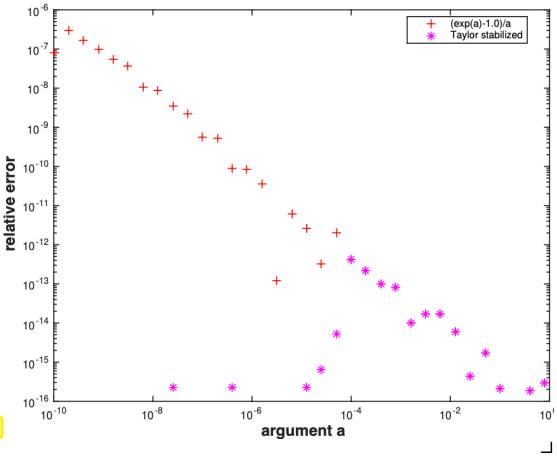


Fig. 31

1.4.4 Numerical Stability (of Algorithm)

Defintion of Problem

From the perspective of mathematics, problem can be defined as a function/mapping (Problem ≡ a function/mapping). Solving a problem is to find such function/mapping. In the view of numerical methods, we need to use numberical methods to build such function/mapping.

We have seen that a particular "problem" can be tackled by different "algorithms", which produce different results due to roundoff errors. This section will clarify what distinguishes a "good" algorithm from a rather abstract point of view.

A mathematical notion of "problem":

- data space X, usually $X \subset \mathbb{R}^n$
- result space Y, usually $Y \subset \mathbb{R}^m$
- mapping (problem function) $F : X \mapsto Y$

A problem is a well defined *function* that assigns to each datum a result.

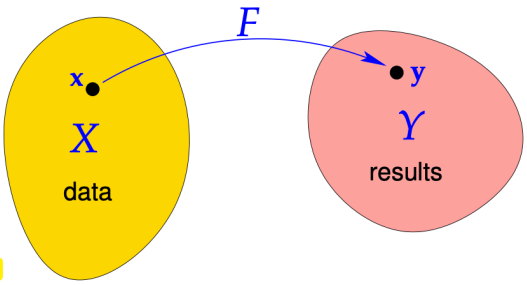


Fig. 32

- In this course, both the data space X and the result space Y will always be subsets of finite dimensional vecor spaces.

Example 1: Matrix × Vector multiplication

$F : (A, \mathbf{x}) \in X \to A\mathbf{x} \in \mathbf{y}$ $\qquad X = \mathbb{R}^{m,n} \times \mathbb{R}^n$ $\qquad \mathbf{y} = \mathbb{R}^m$

NEXT, we need to define norm such that we can measure the perturbation in the data space and the results space.

- On $\mathbb{R}^n$ we use vector norms, e.g. $\|\cdot\|_2 \equiv$ 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

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

Stable algorithm

Definition 1.5.5.19. Stable algorithm

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

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

- $\tilde{F} \equiv$ algorithm affected by roundoff. $\qquad w(x) \equiv$ #ops (number of algebraic operations)
-

Illustration of Def. 1.5.5.19 ▷
($\mathbf{y} \triangleq$ exact result for exact data $\mathbf{x}$)

Terminology:
Def. 1.5.5.19 introduces stability in the sense of backward error analysis

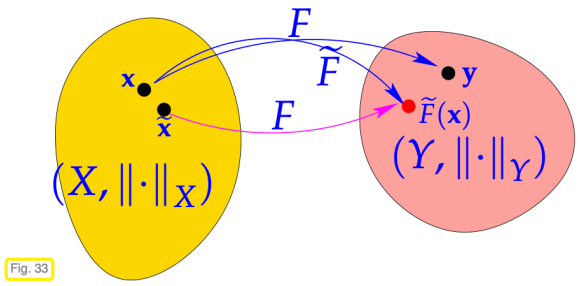


Fig. 33

- Note that "Exact ata hardly ever available": input $\pi, 1/3$
- Sloppily speaking, the impact of roundoff on a stable algorithm is of the same order of magnitude as the effect of the inevitable pertubations due to rounding the input data. For stable algorithms roundoff errors are “harmless”.

Example1: Testing stability of matrix×vector multiplication

Problem: $(A, \mathbf{x}) \in \mathbb{R}^{m,n} \times \mathbb{R}^n \to A\mathbf{x} \in \mathbb{R}^m$ $\quad \mathbf{y} = \tilde{F}((A, \mathbf{x}))$: computed result [Euclidean vector &matrix norms]

which pertubations $(\tilde{A}, \tilde{\mathbf{x}})$ of data $(A, (x))$ yield $\mathbf{y} = \tilde{A}\tilde{\mathbf{x}}$

$$
\begin{gathered}
\text{Possible choice: } \tilde{\mathbf{x}} = \mathbf{x}, \tilde{A} = A + \mathbf{z}\mathbf{x}^T, \ \mathbf{z} \equiv \frac{\mathbf{y} - A\mathbf{x}}{\|\mathbf{x}\|_2^2}) \in \mathbb{R}^m \\
\Rightarrow \tilde{A}\tilde{\mathbf{x}} = A\mathbf{x} + (\frac{\mathbf{y} - A\mathbf{x}}{\|\mathbf{x}\|_2^2})\mathbf{x}^T\mathbf{x} = A\mathbf{x} + \mathbf{y} - A\mathbf{x} = \mathbf{y} \\
F((\tilde{A}, \tilde{\mathbf{x}})) = \mathbf{y} \\
\|A - \tilde{A}\|_2 = \|\mathbf{z}\mathbf{x}^T\|_2 \leq \|\mathbf{x}\|_2\|\mathbf{z}\|_2 = \frac{\|\mathbf{y} - A\mathbf{x}\|_2}{\|\mathbf{x}\|_2} \\
\text{if} \|\mathbf{y} - A\mathbf{x}\|_2 \leq Cmn \cdot \text{EPS}\|\mathbf{x}\|_2\|A\|_2 \ (*) \\
\Rightarrow \|A - \tilde{A}\|_2 \leq Cmn \cdot \text{EPS}\|A\|_2 \\
\text{if } (*) \text{ holds } \forall \text{ inputs } (A, \mathbf{x}) \Rightarrow \tilde{F} \text{ is stable}
\end{gathered}
\tag{8}
$$

Example2: Problem with sensitive dependence on data:

$\|F(\mathbf{x}) - F(\tilde{\mathbf{x}})\|_{\mathbf{y}}$ can be large for tiny $\|\mathbf{x} - \tilde{\mathbf{x}}\|_{\mathbf{x}}$ Following is an example:

Example: The problem is the prediction of the position of the billard ball after ten bounces given the initial position, velocity, and spin.

It is well known, that tiny changes of the initial conditions can shift the final location of the ball to virtually any point on the table: the billard problem is chaotic.

Hence, a stable algorithm for its solution may just output a *fixed* or *random* position without even using the initial conditions!

- Random output is a stable algorithm!

2 Direct Methods for (Square) Linear Systems of Equations

2.1 Introduction - Linear Systems of Equations (LSE)

$$
\text{LSE: } A\mathbf{x} = \mathbf{b} \tag{9}
$$

- $A \in \mathbb{K}^{n,n}$: square system/cofficient/matrx
- $\mathbf{x} \in \mathbb{K}^n$: vector of unknowns
- $\mathbf{b}$: right-hand-side vector $\in \mathbb{K}^n$

#equation = #unknown = n $\to (A)_{i,:}^T\mathbf{x} = (\mathbf{b})_i, i = 1, \ldots, n$ [A scalar equation for each row of A]

AND this question can be simplified as the inverse problem of matrix×vector multiplication:

- Data/Input: $A, \mathbf{b}$
- Sought/Output: $\mathbf{x}$

2.2 Square LSE: Theory → Linear Algebra

2.2.1 Existence and Uniqueness of Solutions

When we discuss topic about equations (no matter PDE/ODE/LSE/FE), we need firstly discuss the existence and uniqueness of this set of equations:

Let's firstly review the concept of Invertible matrix:

Definition 2.2.1.1. Invertible matrix → [NS02, Sect. 2.3]

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

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

Conclusion: A invertible: $A\mathbf{x} = \mathbf{b} \Rightarrow \mathbf{x} = A^{-1}\mathbf{b}$ unique.

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

Criterias for judging the existence of invertible matrix:

- $A \in \mathbb{K}^{n,n}$ regular ⇔
  $\det A \neq 0 \Leftrightarrow$ cols. lin. indep. ⇔ rows lin. indep. ⇔ $N(A)$[nullspace of A] $= \{\mathbf{z} \in \mathbb{K} : A\mathbf{z} = \mathbf{0}\} = \{\mathbf{0}\}$

2.2.2 Sensitivity/Conditioning of LSE

Definition

Quantifies how small (relative) perturbation of data lead to changes of the output. This concept is likely to the numerical stability of the algorithm.

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

$$\|M\| = \underbrace{\sup_{\mathbf{x}\neq 0}}\frac{\|M\mathbf{x}\|}{\|\mathbf{x}\|} \Rightarrow \|M\mathbf{x}\| \leq \|M\|\|\mathbf{x}\| \qquad \|\mathbf{x}+\mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\| \Rightarrow \|\mathbf{x}-\mathbf{y}\| \geq \|\mathbf{x}\| - \|\mathbf{y}\|$$

- We relate the matrix norm with vector norm and discuss the inequalities of vector norm itself.

Examples: Sensitivity of linear mapping. A simple example used to explain above conceptes.

$A \in \mathbb{R}^{n,n}$ fixed & regular: input $\mathbf{x} \to \mathbf{y} = A\mathbf{x} \equiv$ output

$$\begin{aligned} A\mathbf{x} = \mathbf{y} \leftrightarrow \mathbf{x} = A^{-1}\mathbf{y} &\Rightarrow \|\mathbf{x}\| \leq \|A^{-1}\|\|\mathbf{y}\| \\ A(\not{\mathbf{x}} + \delta\mathbf{x}) = \not{\mathbf{y}} + \delta\mathbf{y} &\Rightarrow \|\delta\mathbf{y}\| \leq \|A\|\|\delta\mathbf{x}\| \\ \text{rel. pert.}\frac{\|\delta\mathbf{y}\|}{\mathbf{y}} \leq \frac{\|A\|\|\delta\mathbf{x}\|}{\|A^{-1}\|^{-1}\|\mathbf{x}\|} &= \underbrace{\|A\|\|A^{-1}\|}_{\text{sensitivity}}\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \end{aligned} \tag{10}$$

⋆ Sensitivity heavily depends on choice of norms.

Analysis of LSE

LET'S return back to the sensitivity/conditioning of LSE

$A \in \mathbb{R}^{n,n}$ regular: $A\mathbf{x} = \mathbf{b} \Rightarrow (A+\delta A)(\mathbf{x}+\delta\mathbf{x}) = \mathbf{b}+\delta\mathbf{b} \Rightarrow (A+\delta A)\delta\mathbf{x} = \delta\mathbf{b} - \delta A\mathbf{x}$

Our goal is to characterize: $\|\delta\mathbf{x}\|/\|\mathbf{x}\|$. During this process, we need to define norm, which is important.

Using following Tool (lemma):

Lemma 2.2.2.5. Perturbation lemma → [QSS00, Thm. 1.5]

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

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

$$\begin{gathered}
\overbrace{\|(I+B)\mathbf{x}\| \ge \|\mathbf{x}\| - \|B\mathbf{x}\| \ge (1-\|B\|)}^{>0}\|x\| \\
\cdot\mathbf{x} \neq 0 \Rightarrow (1+B)\mathbf{x} \neq 0 \Rightarrow N(1+B) = \{0\} \\
\cdot\|(1+B)^{-1}\| = \underbrace{\sup_{\mathbf{x}\neq 0}}\frac{\overbrace{\|(1+B)^{-1}\mathbf{x}\|}^{\mathbf{y}}}{\|x\|} = \underbrace{\sup_{\mathbf{y}\in\mathbb{R}^n,\mathbf{y}\neq 0}}\frac{\|y\|}{\|(1+B)\mathbf{y}\|} \le \underbrace{\sup_{\mathbf{y}\neq 0}}\frac{\|\mathbf{y}\|}{(1-\|B\|)\|\mathbf{y}\|} \\
\text{Apply to } A+\delta A = A(I+A^{-1}\delta A) \\
\|(A+\delta A)^{-1}\| = \|(1+A^{-1}\delta A)^{-1}A^{-1}\| \\
[\text{Lemma 2.2.2.5}] \ \le \|A^{-1}\|\frac{1}{1-\|A^{-1}\|\|\delta A\|} \\
\|\delta\mathbf{x}\| \le \frac{\|A^{-1}\|}{1-\|A^{-1}\|\|\delta A\|}\cdot(\|\delta\mathbf{b}\| + \|\delta A\|\|\mathbf{x}\|) \\
\|\delta\mathbf{x}\| \le \frac{\|A^{-1}\|\|A\|}{1-\|A^{-1}\|\|\delta A\|}\left(\frac{\|\delta b\|}{\|A\|\|\mathbf{x}\|} + \frac{\|\delta A\|}{\|A\|}\right)\|\mathbf{x}\|
\end{gathered} \tag{11}$$

Theorem 2.2.2.4. Conditioning of LSEs → [QSS00, Thm. 3.1], [GGK14, Thm 3.5]

If $\mathbf{A}$ *regular,* $\|\Delta\mathbf{A}\| < \left\|\mathbf{A}^{-1}\right\|^{-1}$ *and* (2.2.2.3)*, then*
(i) $\mathbf{A}+\Delta\mathbf{A}$ *is regular/invertible,*
(ii) If $\mathbf{A}\mathbf{x} = \mathbf{b}$, $(\mathbf{A}+\Delta\mathbf{A})\widetilde{\mathbf{x}} = \mathbf{b}+\Delta\mathbf{b}$, *then*

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

relative error of data — *relative perturbations*

- Specifical for conditiontiong of LSEs.

Unified Cretiria of condition for LSE (condition number)

Definition 2.2.2.7. Condition (number) of a matrix

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

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

Used to define the condition/sensitivity of such setting (LSE).

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

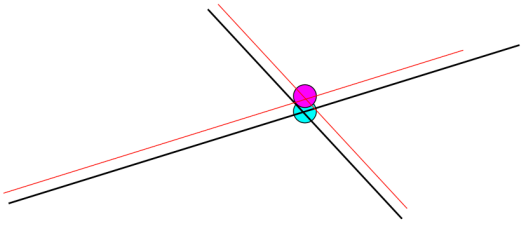
Example: Visualizing sensitivity: intersection of lines in 2D

Remember the Hessian normal form of a straight line in the plane. We are given the Hessian normal forms of two lines L_1 and L_2 and want to compute the coordinate vector $\mathbf{x}\in\mathbb{R}^2$ of the point in which they intersect:

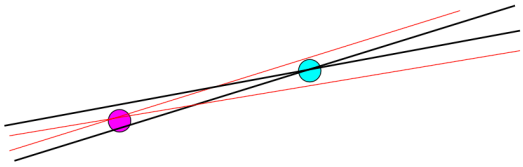
$$L_i=\{\mathbf{x}\in\mathbb{R}^2:\ \mathbf{x}^T\mathbf{n}_i=d_i\}\ ,\quad \mathbf{n}_i\in\mathbb{R}^2,\ d_i\in\mathbb{R}\ ,\quad i=1,2\ .$$

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

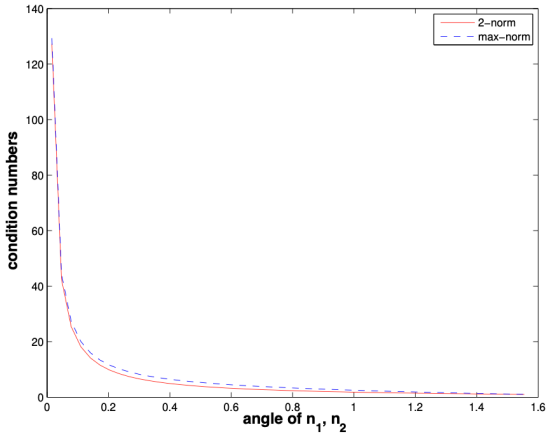
- $\mathbf{x}$ represents coordinates of $L_1\cap L_2$.
- This question can also be viewed as LSE.



nearly orthogonal intersection: well-conditioned



glancing intersection: ill-conditioned



- n_1 "almost linearly dependent" of n_2 $\qquad\qquad \operatorname{cond}(A)\to\infty$

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"

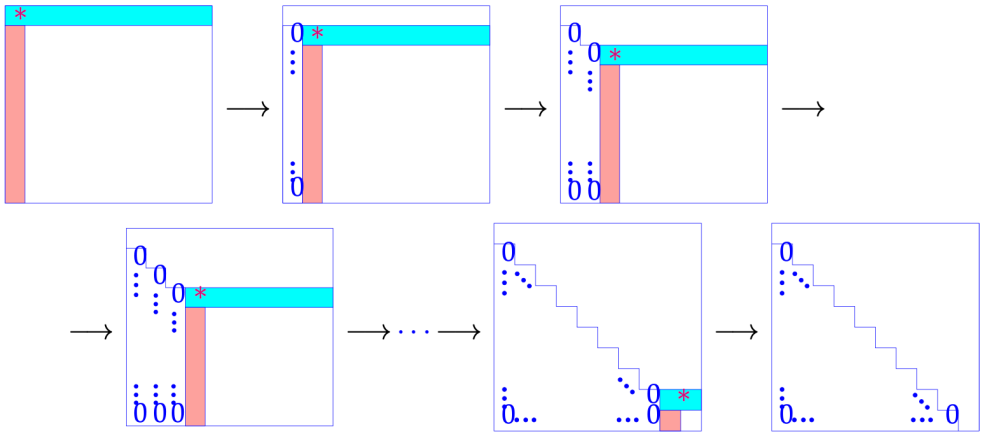
2.3 Gaussian Elimination (GE)

2.3.1 Basic Algorithm (→LA)

Definition of GE

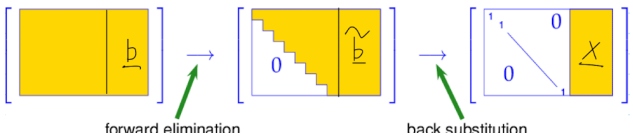
Successive row transformation of LSE $A\mathbf{x} = \mathbf{b}$ to convert it to triangular form

For A:



$*$ ≙ the pivot entry (necessarily $\neq 0$, which we assume here), = pivot row

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



- forward elimination $\rightarrow O(n^3)$ back substitution $\rightarrow O(n^2)$

Block GE

LSE:

$$A\mathbf{x} = \mathbf{b},\ A \in \mathbb{R}^{n,n}, A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} A_{1,1} \in \mathbb{R}^{k,k} \tag{12}$$

Assume: $A_{1,1}$ is regular .

Recall: blockwise $M \times M$:

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

Now: GE like for 2×2 LSE [except for commutativity]

$$\left[\begin{array}{cc|c} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{b}_1 \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{b}_2 \end{array}\right] \xrightarrow{❶} \left[\begin{array}{cc|c} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{b}_1 \\ 0 & \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12} & \mathbf{b}_2 - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{b}_1 \end{array}\right]$$

$$\xrightarrow{❷} \left[\begin{array}{cc|c} \mathbf{I} & 0 & \mathbf{A}_{11}^{-1}(\mathbf{b}_1 - \mathbf{A}_{12}\mathbf{S}^{-1}\mathbf{b}_S) \\ 0 & \mathbf{I} & \mathbf{S}^{-1}\mathbf{b}_S \end{array}\right];,$$

- 1→froward elimination 2→back substitution
- $S \equiv A_{2,2} - A_{2,1}A_{11}^{-1}A_{1,2}$: Schur complement.

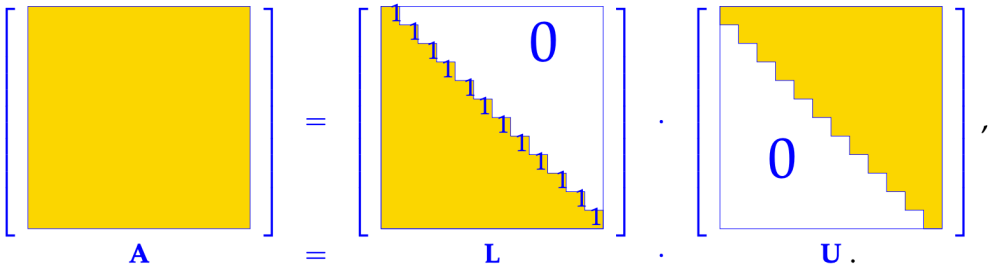
Block LU-decompoition

$$\underbrace{\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & 0 \\ \mathbf{A}_{21}\mathbf{A}_{11}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ 0 & \mathbf{S} \end{bmatrix}}_{\text{block LU-factorization}}, \quad \text{with Schur complement } \boxed{\mathbf{S} := \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}}. \tag{2.3.2.20}$$

2.3.2 LU-Decomposition

Definition of LU-decompoition

[Equivalent to GE] $\Rightarrow \text{cost}(LU) = O(n^3)$



- normalized lower trianglular × 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 $\mathbf{A} = \mathbf{LU}$.

Solving LSE via LU

LSE:

$$A\mathbf{x} = \mathbf{b} \Leftrightarrow L(U\mathbf{x}) = \mathbf{b} \Rightarrow L\mathbf{z} = \mathbf{b}, U\mathbf{x} = \mathbf{z} \tag{13}$$

$\mathbf{Ax} = \mathbf{b}$:
① LU-decomposition $\mathbf{A} = \mathbf{LU}$, #elementary operations $= \frac{1}{3}n(n-1)(n+1)$
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 $= \frac{1}{2}n(n+1)$

- Three-stage procedure of solving $n \times n$ linear systems of equations.
- $1 \to O(n^3)$ $2,3 \to O(n^2)$

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

GE/LU-dec. in Eigen

EIGEN supplies a rich suite of functions for matrix decompositions and solving LSEs. The default solver is Gaussian elimination with partial pivoting, accessible through the methods `lu()` and `solve()` of dense matrix types:

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

LSE multiple r.h.s [contained in columns of B]: $A\mathbf{x} = B \in \mathbb{R}^{n,l}$ Solve a set of LSEs.

Given: system/coefficient matrix $\mathbf{A} \in \mathbb{K}^{n,n}$ regular $\leftrightarrow$ A ($n \times n$ EIGEN matrix)
right hand side vectors $\mathbf{B} \in \mathbb{K}^{n,\ell}$ $\leftrightarrow$ B ($n \times \ell$ EIGEN matrix)
(corresponds to multiple right hand sides, *cf.* Code 2.3.1.10)

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)`

Summarizing the detailed information given in § 2.3.2.15:

cost(`X = A.lu().solve(B)`) = $= O(n^3 + ln^2)$ for $n, l \to \infty$

> Never contemplate implementing a general solver for linear systems of equations!
>
> If possible, use algorithms from numerical libraries! (→ Exp. 2.3.1.7)

Many sequential solutions of LSE

As we have seen above, EIGEN provides functions that return decompositions of matrices.

Based on the precomputed decompositions, a linear system of equations with coefficient matrix $A \in \mathbb{K}^{n,n}$ can be solved with asymptotic computational effort $O(n^2)$.

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

- A rationale for LU
- A high-efficient way for computing serial solutions of LSE.

2.4 Exploiting Structure when Solving Linear Systems

By "structure" of a linear system we mean prior knowledge that

- either certain entries of the system matrix vanish,
- or the system matrix is generated by a particular formula.

Trangular Linear Systems

Triangular linear systems are linear systems of equations whose system matrix is a **triangular matrix**.

Above chapter tells us that (dense) triangular linear systems can be solved by backward/forward elimination with $O(n^2)$ asymptotic computational effort ($n \equiv$ number of unknowns) compared to an asymptotic complexity of $O(n^3)$ for solving a generic (dense) linear system of equations.

This is the simplest case where exploiting special structure of the system matrix leads to faster algorithms for the solution of a special class of linear systems.

Block Elimination

Let's recall Block GE:

For $k, \ell \in \mathbb{N}$ consider the block partitioned square $n \times n$, $n := k + \ell$, linear system

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

$$\begin{aligned} \mathbf{x}_1 &= A_{11}^{-1}(\mathbf{b}_2 - A_{12}\mathbf{x}_2) \text{ from 1st row} \\ \text{into 2nd. row } &\underbrace{(A_{22} - A_{21}A_{11}^{-1}A_{12})\mathbf{x}_2}_{\text{Schur complement matrix} \in \mathbb{R}^{\ell,\ell}} = \mathbf{b}_2 - A_{21}A_{11}^{-1}\mathbf{b}_1 \end{aligned} \tag{14}$$

We can notice that block elimination will be useful, if $A_{11}\mathbf{x} = \mathbf{c}$ can be solved "easily".

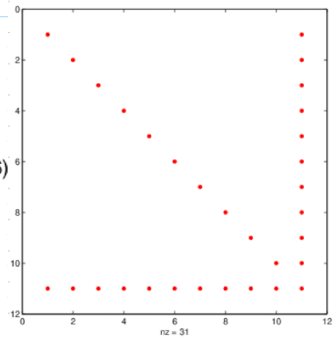
$$\underbrace{\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & 0 \\ \mathbf{A}_{21}\mathbf{A}_{11}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ 0 & \mathbf{S} \end{bmatrix}}_{\text{block LU-factorization}}, \quad \text{with Schur complement } \boxed{\mathbf{S} := \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}}. \tag{2.3.2.20}$$

Let's consider a special situation:

LSE with arrow matrix:

set up as follows:

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



- If we use general GE to solve this question:

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 = $O(n^3)$

cost $= O(n^3)$

- If we use block GE

$$\mathbf{A}\mathbf{x} = \left[\begin{array}{c|c} \mathbf{D} & \mathbf{c} \\ \hline \mathbf{b}^\top & \alpha \end{array}\right] \left[\begin{array}{c} \mathbf{x}_1 \\ \hline \xi \end{array}\right] = \mathbf{y} := \left[\begin{array}{c} \mathbf{y}_1 \\ \hline \eta \end{array}\right], \tag{2.6.0.7}$$

$$\begin{aligned} \xi &= \frac{\eta - \mathbf{b}^T \mathbf{D}^{-1}\mathbf{y}_1}{\alpha - \mathbf{b}^\top \mathbf{D}^{-1}\mathbf{c}}, \\ \mathbf{x}_1 &= \mathbf{D}^{-1}(\mathbf{y}_1 - \xi\mathbf{c}) . \end{aligned} \tag{2.6.0.8}$$

→ $\neq 0$ required!

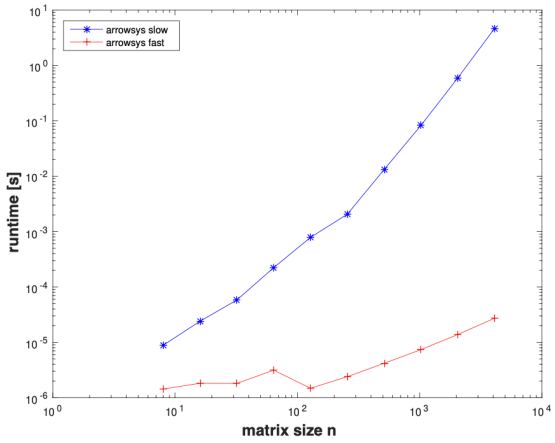
cost $= O(n)$

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();
}
```

→ sale test == 0.0

Comparision between above two methods:



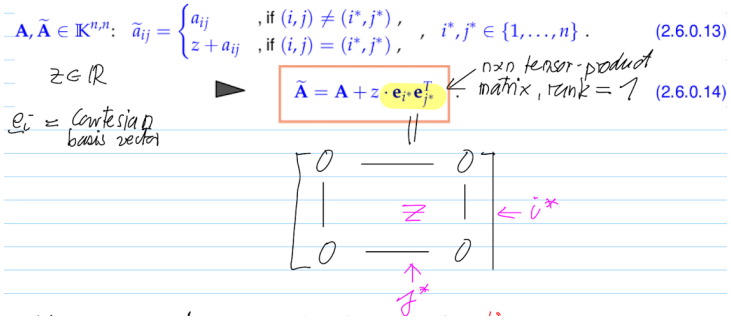
Fast code more vulnerable to round-off: instability.

Solving LSE subject to low-rank modification of system matrix

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

- A special structure
- LU-dec. available

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



MORE general: rank-1 modification.

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

Trick: Block elimination

$$\begin{bmatrix} A & \mathbf{u} \\ \mathbf{v}^T & -1 \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{x}} \\ \xi \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{b}} \\ 0 \end{bmatrix} \tag{15}$$

Block row $\mathbf{II} \Rightarrow \xi = \mathbf{v}^T\tilde{\mathbf{x}} \overset{\mathbf{I}}{\Rightarrow} \underbrace{(A + \mathbf{u}\mathbf{v}^T)}_{\tilde{A}}\tilde{\mathbf{x}} = \mathbf{b}$

Block row $\mathbf{I} \Rightarrow \tilde{\mathbf{x}} = A^{-1}(\mathbf{b} - \xi \cdot \mathbf{u}) \overset{\mathbf{II}}{\Rightarrow} \xi = \mathbf{v}^T A^{-1}(\mathbf{b} - \xi\mathbf{u}) \Rightarrow \xi = \frac{\mathbf{v}^T A^{-1}\mathbf{b}}{1 + \mathbf{v}^T A^{-1}\mathbf{u}}$

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

→ have to solve two LSE $A\mathbf{x} = \dots$

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

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

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.

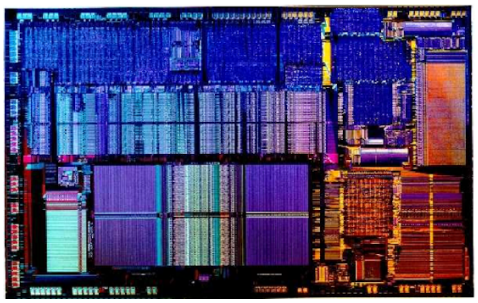
General rank k-matrix

2.5 Sparse Linear Systems

Sparse matrix:

- "Almost all" entries $= 0$
- for which it is worth while exploiting this fact

Example: Sparse LSE in circuit simulation:



- Every is connected to only a few other nodes. we can imagine that graph is also a sparse matrix in computer.
- $\rightarrow$ LSE $A\mathbf{x} = \mathbf{b}$ $\quad \mathrm{nnz}(A) \approx \#nodes$
- Sparse LSE arise in models of large models of large networks

2.5.1 Sparse Matrix Storage Formats

Motivation/Purposes

For large sparse matrix, we want to find a efficient storage way which don't affect its algebraic operation and access.

- Memory $\sim \mathrm{nnz}(A)$ (number of nonzero entries)
- $\mathrm{cost}(A \times \mathrm{vector}) \sim \mathrm{nnz}(A)$ elementary operation.
- provide easily accessible information about location of non-zero entries.

COO/triplet format

$\rightarrow$ 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>;
```

or we can use `std::vector<Eigen::Triplet<double>> triplets;`

C++-code 2.7.1.2: Matrix×vector product $\mathbf{y} = \mathbf{A}\mathbf{x}+\mathbf{y}$ in triplet format

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

Allowed: repeated index pairs $\rightarrow$ values summed up

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

Compressed row storage(CRS)

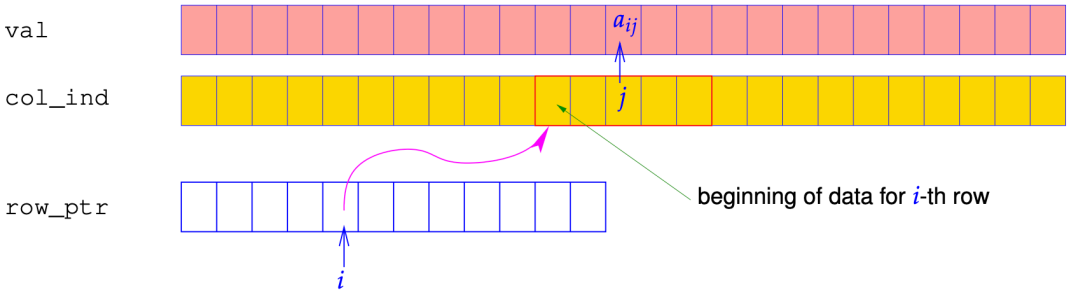
$A \in \mathbb{K}^{n,n}$: stores non-zeros entries, row-oriented

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

As above we write $\mathrm{nnz}(\mathbf{A}) \triangleq$ (number of nonzeros) of $\mathbf{A}$

Access to matrix entry $a_{ij} \neq 0$, $1 \leq i,j \leq n$ ("mathematical indexing")

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



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

`val`-vector:

10	-2	3	9	3	7	8	7	3…9	13	4	2	-1

`col_ind`-array:

1	5	1	2	6	2	3	4	1…5	6	2	5	6

`row_ptr`-array:

1	3	6	9	13	17	20

- Cores of such method is that we need to identify the original location of entries in `val` vector. We use:
  - `col_ind` to identify the number of columns
  - `row_ptr`: compress data in this dimension by defining row pointer to allocate the corresponding row. If we have n rows, then the size of the `row_ptr` is $n+1$. And the value define the accumulated number in each row. For example:
    - First row: $1 \leq a < 3$ Second row: $3 \leq a < 6$ ... for `val` vector
- CRS: Non-zero entries reside in contiguous memory!

2.5.2 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 we don't know how to initialize `Eigen::SparseMatrix<double>` `asp(n,n)` quickly.

Set random entry ⇒ massive data movement necessary

Idea:

1. 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

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

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

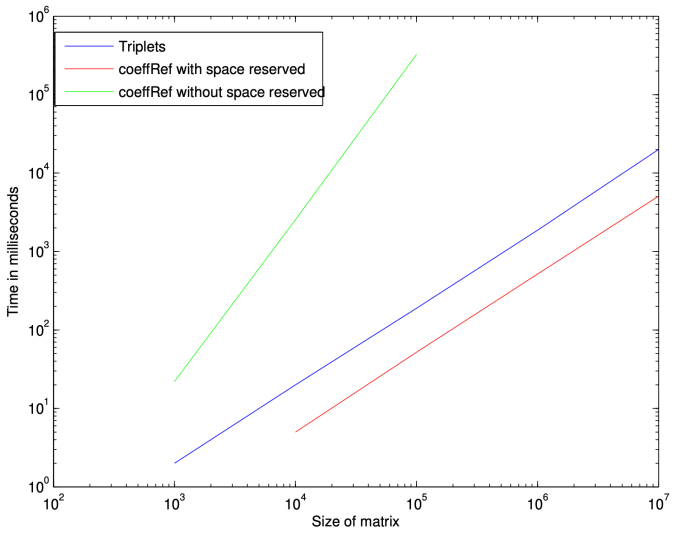
```
unsigned int rows,cols,max_no_nnz_per_row;
.....
SparseMatrix<double, RowMajor> mat(rows,cols);
mat.reserve(RowVectorXi::Constant(cols,max_no_nnz_per_row)); --> Allocation of enough space
// do many (incremental) initializations
for ( ) {
  mat.insert(i,j) = value_ij;
  mat.coeffRef(i,j) += increment_ij;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

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

↳ squeeze out zeros

Initialization of Sparse Matrix

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



Very interesting point! Time for initialization of sparse matrix will change with different methods

2.5.3 Direct Solution of Sparse Linear Systems of Equations

Introduction

Assume: System matrix in sparse matrix format→tells location of zero entries→can be exploited by sparse elimination techniques.

Eigen:

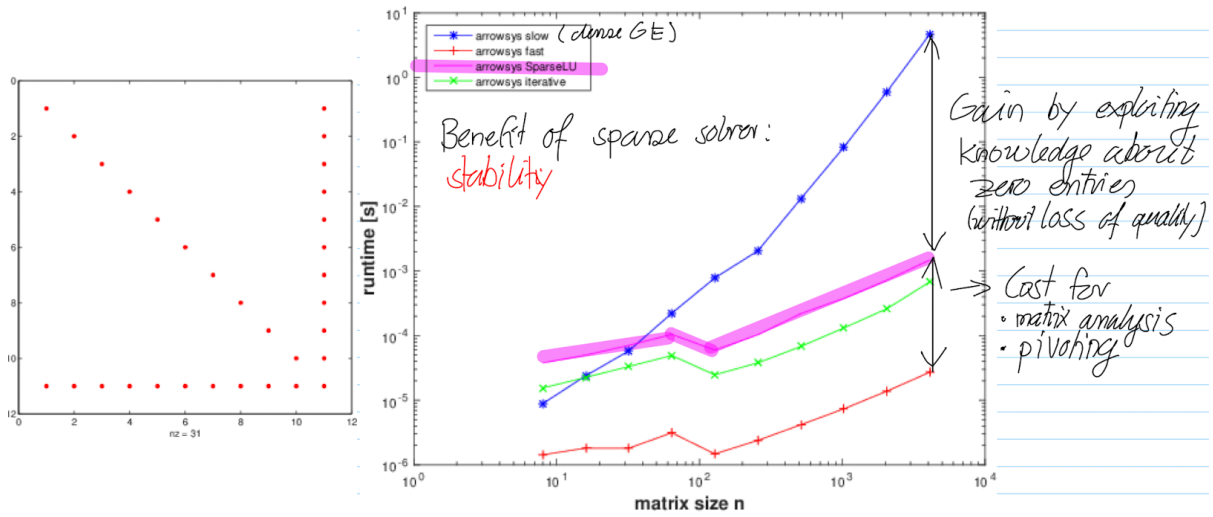
C++-code 2.7.3.1: Function for solving a sparse LSE with EIGEN → GITLAB

```
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

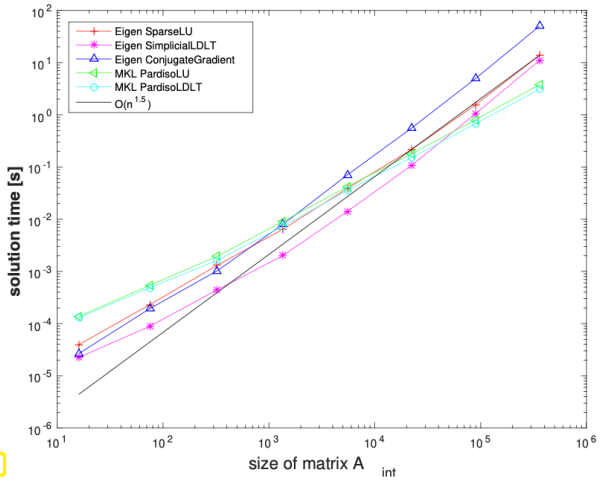
```
//An example in the homework
C = buildC(A); //reshape Lypunov funtion
b = Eigen::Map<Eigen::MatrixXd>(I.data(),n*n,1);
Eigen::SparseLU<Eigen::SparseMatrix<double>> solver;
solver.compute(C);
vecX = solver.solve(b);
X = Eigen::Map<Eigen::MatrixXd>(vecX.data(),n,n);
```

Sparse elimination for arrow matrix



Sparse elimination for combinatorial graph laplacian

We consider a sequence of planar triangulations created by successive regular refinement (→ Def. 2.7.2.16) of the planar triangulation of Fig. 55, see Ex. 2.7.2.5. We use different EIGEN and MKL sparse solver for the linear system of equations (2.7.2.15) associated with each mesh.



- We observe an empirical asymptotic complexity (→ Def. 1.4.1.1) of $O(n^{1.5})$, way better than the asymptotic complexity of $O(n^3)$ expected for Gaussian elimination in the case of dense matrices.

In practice: cost(sparse solution of $A\mathbf{x} = \mathbf{b}$) $= O((\mathrm{nnz}(A))^{\alpha})$ $\alpha \approx 1.5 - 2.5$

> 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.

- Never ever even think about implementing a general sparse elimination solver by yourself!

3 Direct Methods for Linear Least Squares Problems

In this chapter, we will firstly introduce the definition of linear least squares problems. Then, we will discuss

Analytical Solutions:

- **Normal Equations:** For matrix with full-rank condition.
- **Moore-Penrose pseudo-inverse:** For the solution of general LSE.

Orthogonal Transformations Methods:

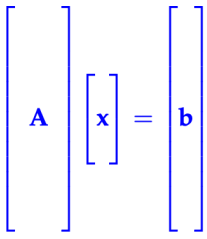
- **QR factorization:** For matrix with full-rank condition. Transfer the original matrix to triangular matrix which is easily to solve. Subsequently, the problem will be transferred into the standard LSE.
- **SVD factorization:** For the solution of general LSE.

3.1 Overdetermined Linear Systems of Equations: Examples

Definition

Overdetermined LSE:

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



- Overdetermined LSE → Always be unsolved; If it is solvable, then $[A]_{1:m,n}$ is enough for solving these problem with n variables.
- Potential Question background: This situation often happens when (1) we have a lot of data points and want to use a linear model to approximate these data. (2) some data is generated by a specific linear model. However, due to the pertubation of this system, PDE not fully satisfied by these data.

Examples of such question

Example 1: Linear parameter estimation in 1D

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

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

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

However: measurement errors have occurred. And this question is not a LSE problem but a regression problem. Here, we will consider linear regression. And in mathematics, linear regression can be viewed as the overdetermined linear systems of equations. [linear regression is just apparent, the core of it is overdetermined LSE.]

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

↑ will be perturbed ⇒ no solutions

Example 2: Linear regression: linear parameter estimation

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

Measurements: $(x_i, y_i) \in \mathbb{R}^n \times \mathbb{R},\ i = 1, \ldots, m$

Theoretically, $y_i = \mathbf{a}^T \mathbf{x}_i + \beta$ (Overdetermined if $m > n + 1$)

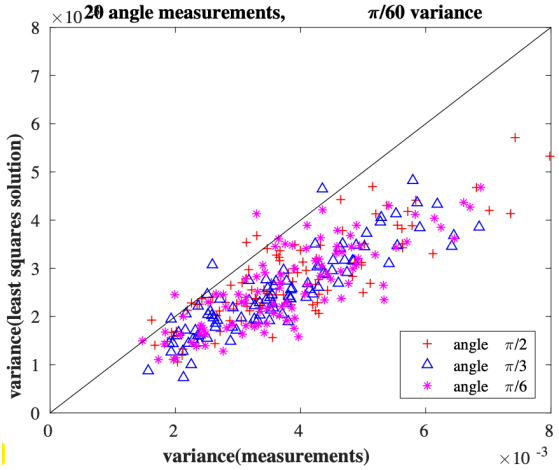
- Multi-variable linear regression. Similarly, y maybe pertubation.

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

Exampel 3: Measuring the angles of a triangle.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} \widetilde{\alpha} \\ \widetilde{\beta} \\ \widetilde{\gamma} \\ \pi \end{bmatrix}. \tag{3.0.1.7}$$

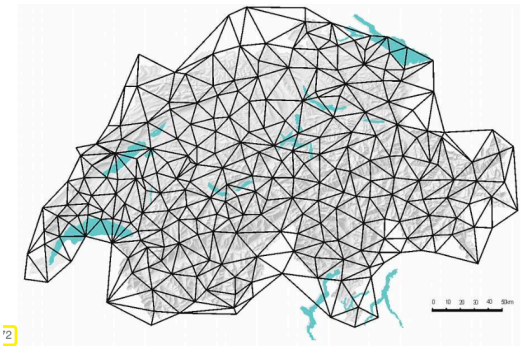
- $\widetilde{\alpha}, \widetilde{\beta}, \widetilde{\gamma} \equiv$ measured angles. $\quad \sum \text{angles} = \pi$
- The tenet of data science: You cannot afford not to use any piece of information available.



Here we just report the results of a numerical experiment:
We consider the triangle with angles $\pi/2$, $\pi/3$, and $\pi/6$. Synthetic "measurement errors" are introduced by adding a normally distributed random perturbation with mean 0 and standard deviation $\pi/60$ to the exact values of the angles, yielding $\widetilde{\alpha}$, $\widetilde{\beta}$, and $\widetilde{\gamma}$.
For 20 "measurements" we compute the variance of the raw angles and that of the estimates obtained by solving (3.0.1.7) in least squares sense (→ Section 3.1.1). These variances are plotted for many different "runs".

Example 4: Angles in a triangulation

Measured: angles



In Ex. 2.7.2.5 we learned about the concept and data structures for planar triangulations → Def. 2.7.2.6. Such triangulations have been and continue to be of fundament importance for geodesy. In particular before distances could be measured accurately by means of lasers, triangulations were indispensable, because angles could already be determined with high precision. *C.F. Gauss* pioneered both the use of triangulations in geodesy and the use of the least squares method to deal with measurement errors → Wikipedia.

- each angle is supposed to be equal to its measured value,
- the sum of interior angles is π for every triangle, → + #triag. Extra equations ⇒OD LSE
- the sum of the angles at an interior node is 2π. → + #int. Nodes extra equations.

Example 5: Point locations from distances

Consider n points located on the real axis at unknown locations $x_i \in \mathbb{R}, i = 1,\ldots,n$. At least we know that $x_i < x_{i+1}$, $i = 1,\ldots,n-1$.

We *measure* the $m := \binom{n}{2} = \frac{1}{2}n(n-1)$ pairwise distances $d_{ij} := |x_i - x_j|$, $i,j \in \{1,\ldots,n\}$, $i \neq j$.
They are connected to the point positions by the overdetermined linear system of equations

$$\begin{array}{c} x_i - x_j = d_{ij}\,, \\ 1 \le j < i \le n\,. \\ \updownarrow \\ \mathbf{A}\mathbf{x} = \mathbf{b} \end{array} \leftrightarrow \begin{bmatrix} -1 & 1 & 0 & \ldots & & \ldots & 0 \\ -1 & 0 & 1 & 0 & & & \\ \vdots & & \ddots & \ddots & & & \\ \vdots & & & \ddots & \ddots & & \\ -1 & \ldots & & & & 0 & 1 \\ 0 & -1 & 1 & & & & 0 \\ \vdots & & & & & & \vdots \\ \vdots & & & & & & \vdots \\ 0 & \ldots & & & & -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_{12} \\ d_{13} \\ \vdots \\ \vdots \\ d_{1n} \\ d_{23} \\ \vdots \\ \vdots \\ d_{n-1,n} \end{bmatrix} \tag{3.0.1.10}$$

Note that we can never expect a unique solution for $\mathbf{x} \in \mathbb{R}^n$, because adding a multiple of $[1,1,\ldots,1]^\top$ to any solution will again yield a solution, because $\mathbf{A}$ has a non-trivial kernel: $\mathcal{N}(\mathbf{A}) = [1,1,\ldots,1]^\top$.
Non-uniqueness can be cured by setting $x_1 := 0$, thus removing one component of $\mathbf{x}$.

If the measurements were perfect, we could then find $x_2, \ldots, x_n$ from $d_{i-1,i}$, $i = 2, \ldots, n$ by solving a standard (square) linear system of equations. However, as in above analysis, using much more information through the overdetermined system helps curb measurement errors.

3.2 Least Squares Solution Concepts

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

Notation for important subspaces associated with a matrix $\mathbf{A} \in \mathbb{K}^{m,n}$

- Image/range: $\mathcal{R}(A) \equiv \{\mathbf{Ax}, \mathbf{x} \in \mathbb{K}^n\} \subset \mathbb{K}^m$,
- Kernel/nullspace: $\mathcal{N}(A) \equiv \{\mathbf{x} \in \mathbb{K} : \mathbf{Ax} = \mathbf{0}\}$.

A is not a square matrix, which means that the number of equations is not consistent with that of variables. if $m \geq n$, this means that the number of equations is larger than that of unknown variables. This is what we call the overdetermined linear system of equations.

3.2.1 Least Squares Solutions: Definition

After downing the question background, we want to figure out the potential soution of such problem.

Idea: A LSQ sol. of $A\mathbf{x} = \mathbf{b}$ is a vector $x \in \mathbb{R}^n$ that makes the residual $r \equiv \mathbf{b} - A\mathbf{x}$ 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

$$\mathbf{x} \in \operatorname*{argmin}_{\mathbf{y} \in \mathbb{R}^n} \|\mathbf{Ay} - \mathbf{b}\|_2^2 ,$$

$$\Updownarrow$$

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

- LSQ need not be unique. Expression: $\equiv \operatorname{lsq}(A, b)$
- The solution is based on the definition of the norm. [Because norm will define the distance in the Euclidean space, and the goal of lsq is to find the shortest distance based on such norm.]
- Here, **A** can be viewed as the coordinate matrix of the data, and **y** can viewed as the value matrix of the data.

A true generalization: $\mathbf{A} \in \mathbb{R}^{n,n}$ regular $\Rightarrow \operatorname{lsq}(\mathbf{A},\mathbf{b}) \equiv \mathbf{A}^{-1}\mathbf{b}$

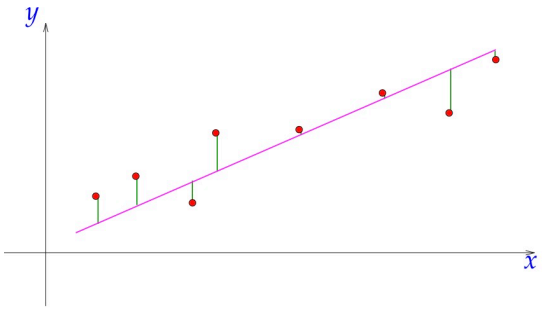
Example: 1D linear regression

OD-LSE:

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

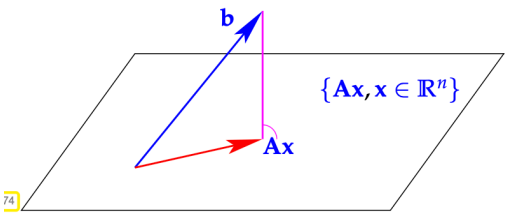
↑ will be perturbed ⇒ no solutions

LSQ sol.: $[\alpha\ \beta] \in \underbrace{\operatorname{argmin}}_{\tilde{\alpha},\tilde{\beta}} \sum_{i=1}^{m} (y_i - \tilde{\alpha}x_i - \tilde{\beta})^2$



- Minimize the sum of squares of vertical distances of the data point from the regression line.

Geometric interpretation of least squares solution



◁ For a least squares solution $\mathbf{x} \in \mathbb{R}^n$ the vector $\mathbf{Ax} \in \mathbb{R}^m$ is the unique orthogonal projection of $\mathbf{b}$ onto

$$\mathcal{R}(\mathbf{A}) = \operatorname{Span}\{(\mathbf{A})_{:,1}, \ldots, (\mathbf{A})_{:,n}\},$$

because the orthogonal projection provides the nearest (w.r.t. the Euclidean distance) point to $\mathbf{b}$ in the subspace (hyperplane) $\mathcal{R}(\mathbf{A})$.

- From this geometric consideration we conclude that $\operatorname{lsq}(\mathbf{A}, \mathbf{b})$ is the space of solutions of $\mathbf{Ax} = \mathbf{b}^*$, where $\mathbf{b}^*$ is the orthogonal projection of $\mathbf{b}$ onto $\mathcal{R}(\mathbf{A})$. Since the set of solutions of a linear system of equations invariably is an affine space, this argument teaches that $\operatorname{lsq}(\mathbf{A}, \mathbf{b})$ is an affine subspace of $\mathbb{R}^n$!

Geometric intuition generates the following insights: Least squares solutions always exist

Theorem 3.1.1.9. Existence of least squares solutions

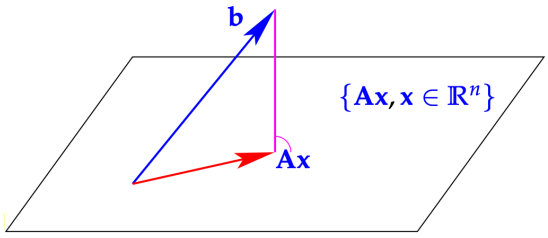
For any $\mathbf{A} \in \mathbb{R}^{m,n}$, $\mathbf{b} \in \mathbb{R}^m$ a least squares solution of $\mathbf{Ax} = \mathbf{b}$ (→ Def. 3.1.1.1) exists.

Proof. The function $F : \mathbb{R}^n \to \mathbb{R}$, $F(\mathbf{x}) := \|\mathbf{b} - \mathbf{Ax}\|_2^2$ is continuous, bounded from below by 0 and $F(\mathbf{x}) \to \infty$ for $\|\mathbf{x}\| \to \infty$. Hence, there must be an $\mathbf{x}* \in \mathbb{R}^n$ for which it attains its minimum. □

3.2.2 Normal Equations

In this chapter, the normal equation not only gives the actual solution of least squares problem → normal equation (a n×n LSE), but identify when the $\operatorname{lsq}(\mathbf{A}, \mathbf{b})$ ($\mathbf{A}$ is a tall matrix) will have the unique solution → full-ranke condition (FRC).

Setting: OD-LSE $\mathbf{Ax} = \mathbf{b}$, $\mathbf{A} \in \mathbb{R}^{m,n}, \mathbf{b} \in \mathbb{R}^m$, $m \geq n$ $\qquad$ $x^* \in \operatorname{lsq}(\mathbf{A}, \mathbf{b})$



↓ NEXT, we will obtain the **normal equation** from (1) geometric view of points; (2) algebric view of point.

Geometirc points

$$\begin{aligned} &\text{residual } r \equiv \mathbf{b} - \mathbf{A}\mathbf{x}^* \perp \mathcal{R}(\mathbf{A}) = \mathbf{A}\mathbf{y} : \mathbf{y} \in \mathbb{R}^n \\ &(\mathbf{A}\mathbf{y})^T(\mathbf{b} - \mathbf{A}\mathbf{x}^*) = 0 \; \forall \mathbf{y} \in \mathbb{R}^n \\ &\Rightarrow \mathbf{A}^T(\mathbf{b} - \mathbf{A}\mathbf{x}^*) = 0 \end{aligned} \tag{16}$$

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* $\mathbf{A}\mathbf{x} = \mathbf{b}$, $\mathbf{A} \in \mathbb{R}^{m,n}$, $\mathbf{b} \in \mathbb{R}^m$, if and only if *it solves the normal equations (NEQ)*

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

- $\mathbf{x} \in \mathrm{lsq}(\mathbf{A}, \mathbf{b}) \Leftrightarrow \mathbf{A}^\mathbf{T}\mathbf{A}\mathbf{x} = \mathbf{A}^\mathbf{T}\mathbf{b}$ ($n \times n$ Square LSE)

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

$$\Leftrightarrow \quad \begin{bmatrix} \mathbf{A}^\top\mathbf{A} \end{bmatrix} \begin{bmatrix} \mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{A}^\top \end{bmatrix} \begin{bmatrix} \mathbf{b} \end{bmatrix} .$$

Algebric points

Define: $J : \mathbb{R}^n \to \mathbb{R}$ $\qquad J(\mathbf{y}) \equiv \|\mathbf{b} - \mathbf{A}\mathbf{y}\|_2^2$

$$\begin{aligned} J(\mathbf{y}) &= \mathbf{y}^\top \mathbf{A}^\top \mathbf{A}\mathbf{y} - 2\mathbf{b}^\top \mathbf{A}\mathbf{y} + \mathbf{b}^\top \mathbf{b} \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} (\mathbf{A}^\top\mathbf{A})_{ij} y_i y_j - 2\sum_{i=1}^{m}\sum_{j=1}^{n} b_i(\mathbf{A})_{ij} y_j + \sum_{i=1}^{m} b_i^2 , \end{aligned} \qquad \mathbf{y} = [y_1, \ldots, y_n]^\top \in \mathbb{R}^n .$$

- Multivariant polynomial in $\mathbf{y}$: J smooth, C^∞
- $x^* \in \underbrace{\operatorname{argmin}}_{x \in \mathbb{R}^n} J(\mathbf{x}) \Rightarrow \underbrace{\mathbf{grad}\, J(\mathbf{x}^*)}_{\mathbb{R}^n \to \mathbb{R}^n} = 0$
- $\operatorname{grad} J(\mathbf{y}) = [\frac{\partial J}{\partial y_i}(\mathbf{y})]_{i=1}^n = 2\mathbf{A}^\mathbf{T}\mathbf{A}\mathbf{y} - 2\mathbf{A}^\mathbf{T}\mathbf{b}$

Example: NEQ for linear regression

3.2.3 Moore-Penrose Pseudoinverse

Normal equation is the full characterication of least squares problems. But we know that when the full-rank condition is not satisfied, the solution of LSQ is not unique. Here, we will give a more general criterion for linear systems of equations (including n by n and m by n).

3.3 Normal Equation Methods

When we used normal equation to solve the LSQ, Usually we will face two obstacles: 1. roundoff errors → loss of information in the computation of $\mathbf{A}^\mathbf{T}\mathbf{A}$ due to the EPS; 2. loss of sparsity → for large m, n
$\mathbf{A}$ sparse $\nRightarrow$ $\mathbf{A}^\mathbf{T}\mathbf{A}$ sparse.

3.4 Orthogonal Transformation Methods

In above section, we have introduced the **normal equation** for solving the overdertimined LSE. In this section, we will introduce another class of methods for solving LSQ inspired by Gaussian elimination.

3.4.1 Idea

3.4.2 Orthogonal Matrices

3.4.3 QR-Decomposition

3.4.3.1 Theory

In the solution of LSE, we used the Gaussian Elimination to introduce the LU decomposition. In the solution of overdetermined LSE (tall matrix), we used Gram-Schmidt orthonormalization to introduce the QR decomposition.

3.4.3.2 Computation of QR-Decomposition

? Not clearly understand

Motivation: Gram-Schmidt algorithm is unstable, affected by roundoff.

Idea: (similar to GE→row transformation) find simple unitary/orthogonal (row) transformations rendering certain matrix elements zero:

- House-holder reflection
- Given Rotation

The $\mathbf{Q}$ is stored in the encode form so that the matrix × vector can be computed quickly.

Unitary/orthogonal matrices are a little different from matrices with orthonormal columns. Unitary/orthogonal matrices must be n by n, because it is related to the regular/invertible. orthonormal vectors is defined as follows: $(\mathbf{q^i})^\mathbf{T}\mathbf{q^j} = \delta_{ij}$ or $\mathbf{U^H U = I_n}$ ($\mathbf{U}$ Orthonormal columns).

- In original definition, orthonormal transformation is by multiplying a series of unitary/orthnormal matrices without changing its norm.

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

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

3.4.3.3 QR-Based Solver for Linear Least Squares Problems

3.5 Singular Value Decomposition (SVD)

In this chapter, firstly, we will review the basic theory of singular value decomposition (SVD), then we will discuss its application as follows: (1) General LSE solution (no matter any size and rank). (2) SVD-Based Optimization and Approximation.

3.5.1 SVD: Definition and Theory

Another factorization based on orthogonal transformations. QR-upper triangulr matrix; SVD-diagonal matrix

3.5.2 SVD in Eigen

Computing following components:

1. Computing SVDs $\Rightarrow \mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$
2. Computing rank (concerned with roundoff)
   (a) numerical rank: judging singular values one by one
   (b) Using `rank()` in Eigen `A.jacobiSvd().setThreshold(tol).rank();`

```
using index_t = MatrixXd::Index;
Eigen::JacobiSVD<MatrixXd> svd(A,Eigen::ComputeFullV);
index_t r = svd.setThreshold(tol).rank();
```

3. Computing orthonormal basis (ONB) of Nullspace and Rangespac

3.5.3 Solving General Least-Squares Problems by SVD

Giving the general solution of Linear System based on psedo-inverse matrix by using the SVD directly.

We can given the Moore-Penrose Pseudoinverse directly by SVD decomposition.

3.5.4 SVD-Based Optimization and Approximation

3.5.4.1 Norm-Constrained Extrema of Quadratic Forms

The key point for the orthogonal transformation is that it don't change the norm of the vector. This will give us many inspiration of orthogonal transformation in the min/max problem.

3.5.4.2 Best Low-Rank Approximation

SVD can be used for best low-rank approximation. That is we can use small number of memory to preserve the most characteristics of the matrix.

3.5.4.3 Principal Component Analysis (PCA)

PCA can be used for (1) Trend detection; (2)Data classification.

4 Data Interpolation and Data Fitting in 1D

4.1 Abstract Interpolation (AI)

Tasks: Data Interpolation

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* $f : I \mapsto \mathbb{R}$

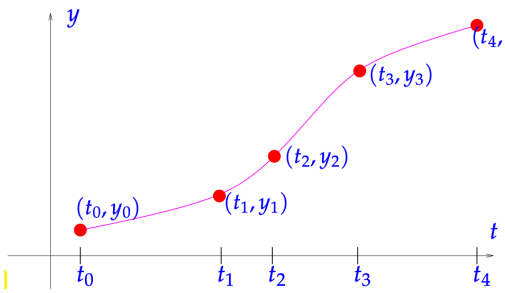
- satisfying the $n+1$ **interpolation conditions** (IC)

$$f(t_i) = y_i, \quad i = 0, \ldots, n \,. \tag{5.1.0.2}$$

- and belonging to a set V of eligible functions.

The function f we find is called the **interpolant** of the given data set $\{(t_i, y_i)\}_{i=0}^{n}$.

- we have $n+1$ points; Set V of eligible functions define smoothness requirements etc.
- Key points of this definition we need to understand: (a) satisfying **Interpolation conditions**; (b) The interpolated function comes from a set V of eligible functions. We refer it as Interpolant.



Parlance: The numbers $t_i \in \mathbb{R}$ are called **nodes**, the $y_i \in \mathbb{R}$ are the (data) **values**.

◁ • ≙ data points $(t_i, y_i) \in \mathbb{R}^2$

t_i ≙ nodes

y_i ≙ values

Graph of interpolant f passes through data points.

Examples: Constitutive relations from measurements

In this context: t, y ≙ two state variables of a physical system, where t determines y: a functional dependence $y = y(t)$ is assumed.

Examples: t and y could be

t	y
voltage U	current I
pressure p	density ρ
magnetic field H	magnetic flux B
...	...

Known: several *accurate* $(*)$ measurements

$$(t_i, y_i) \,, \quad i = 1, \ldots, m$$

- Measurements → accurate ⇔ IC; Functional relation $y = f(t)$ needed for numerical.

Functions in a code

Motivation: What does it mean to "represent" or "make available" a function $f : I \subset \mathbb{R} \to \mathbb{R}$ in a computer code?

- A general "mathematical" function $f : I \subset \mathbb{R} \to \mathbb{R}^d$, I an interval, contains an "infinite amount of information".

Rather, in the context of numerical methods, "function" should be read as "subroutine", a piece of code that can, for any $x \in I$, compute $f(x)$ in finite time. Even this has to be qualified, because we can only pass machine numbers $x \in I \bigcap M$ and, of course, in most cases, $f(x)$ will be an approximation. In a C++ code a simple real valued function can be incarnated through a function object of a type as given in Code 5.1.0.10.

C++-code 5.1.0.10: C++ data type representing a real-valued function ➙ GITLAB

```
class Function {
  private:
    // various internal data describing f
  public:
    // Constructor:  expects information for specifying the function
    Function(/* ... */);
    // Evaluation operator
    double operator () (double t) const;
};
```

If a constitutive relationship for a circuit element is needed in a C++ simulation code (→ Ex. Above), the following specialized **Function** class could be used to represent it. It demonstrates the concrete object oriented implementation of an interpolant.

C++-code 5.1.0.12: C++ class representing an interpolant in 1D ➙ GITLAB

```

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;
};
```

Two main components have to be designed and implemented:

1. The constructor, which is in charge of "setup", e.g. building and solving a linear system of equations.
2. The evaluation operator **operator** (), e.g., implemented as evaluation of a linear combination

d

1. Understand interpolation operator and linear interpolation operator.
   (a) Before we discussed about interpolation operator, the nodes are given. The mapping is between values in this nodes (n+1) and a resulting function (continuous function).
   (b) Linear means that the operator satisfies $\mathbf{I}(\alpha\mathbf{y} + \beta\mathbf{z}) = \alpha\mathbf{I}(\mathbf{y}) + \beta\mathbf{I}(\mathbf{z}),\ \forall \mathbf{y}, \mathbf{z} \in \mathbb{R}^{n+1}, \alpha, \beta \in \mathbb{R}$
2. And we introduce a type of interpolation operator: **Piecewise linear interpolation**
   (a) cardinal basisi: "tent functions"

4.2 Global Polynomial Interpolation

(Global) polynomial interpolation, that is, interpolation into spaces of functions spanned by polynomials up to a certain degree, is the simplest interpolation scheme and of great importance as building block for more complex algorithms.

In this lecture note, I won't discussed another typical methods-piecewise interpolation including trigonometric interpolation (generic periodic function), piecewise interpolation, etc.

Note that, interpolation operator a mapping from data

4.2.1 Uni-Variate Polynomials

4.2.2 Polynomial Interpolation: Theory

4.2.3 Polynomial Interpolation: Algorithms

Not clearly about the single evaluation.

4.2.3.1 Extrapolation to Zero

Inspiration: We found that that taylor expansion of a function without a remainder term is a polynomial. We can sample some points in this function, and interpolate it. Theoretically, we can approximate the function with any precision with more sampled points.

Motivation: Compute the limit $\lim_{h\to 0}\psi(h)$ with prescribed accuracy, though the evaluation of the function $\psi = \psi(h)$ (maybe given in procedural form only) for very small arguments $\|h\| \ll 1$ is difficult, usually because of numerical instability.

Advantage: Not only can it achieve high precision with relative small evaluation, but we can use it to achieve error control.

4.2.3.2 Newton Basis and Divided Differences

Motivation:

1. Evaluation of data with different orders (another reason is that we may not be known when we receive information about the first interpolation points) simulataneously without sacrificing efficiency. However, the drawback of the lagrange basis or barycentric formula is that adding another data point affects *all* basis polynomials/all precomputed values!
2. In order to solve the leading coefficient of the interpolating polynomials with Newton basis, we have to conduct the elimination for a triangular linear system. Can we find more efficient algorithm?

Techniques:

1. we have already discussed monomials basis, polynomial basis, cardinal basis, but we need to use an "update-frendly" basis of $\mathcal{P}_n$ →Newton basis; Evaluation: we use Horner-like scheme
2. Efficient algorithm for determining leading coefficient: divided differences. (based on the A.N. -like scheme)

The coefficients of newton basis is unknown and needed to be solved by triangular linear system equations.

Results:

Advantages:

4.2.4 Polynomial Interpolation: Sensitivity

Motivation

In chapter 1.4.4, we have known that the sensitivity/conditioning of an algorithm/problem/mapping should be evaluated such that we can utilize its properly.

Sensitivity (of a problem) $\longleftrightarrow$ amplification of pertubations of data

- In chapter 1.4.4, problem means a mapping: {Data (space)} $\rightarrow$ {Result (space)}

FOR linear interpolation problem (LIP):

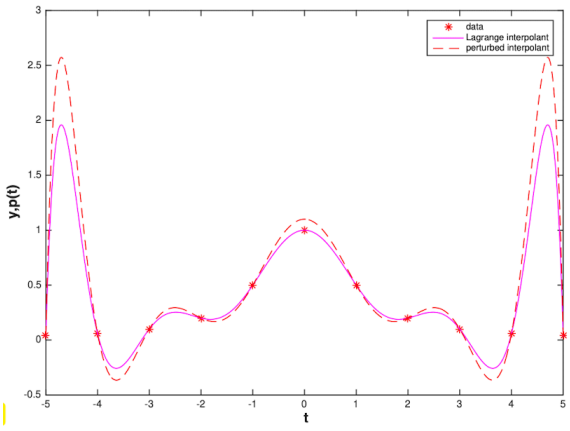
- Data space: data points $(\mathbb{R} \times \mathbb{R})^{n+1}$ or data values $\mathbb{R}^{n+1}$
- Result space: $\mathcal{P}_n$ (a function space, polynomial with certain order=#points)

It will be difficult to gauge the sensitivity of resulting polynomial to interpolation nodes, we won't discuss about it here. Simply, we will discuss the situation where the interpolated nodes are fixed (equidistant nodes).

Our focus is on interpolation operator, more specifically linear interpolation opeartor.

$$I_{\mathcal{T}} : \begin{cases} \mathbb{R}^{n+1} & \to \mathcal{P}_n \,, \\ (y_0, \ldots, y_n)^T & \mapsto \text{interpolating polynomial } p \,. \end{cases} \tag{5.2.2.9}$$

Example of oscillating polynomial interpolant



- Plotted is the interpolant for n=10 and the interpolant for which the data value at t=0 has been pertubated by 0.1. (large variaton at t=4.5)

☞ 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".

In fuzzy terms, what we have observed is "high sensitivity" of polynomial interpolation with respect to perturbations in the data values: small perturbations in the data can cause big variations of the polynomial interpolants in certain points, which is clearly undesirable.

Quantatively quantify these pertubations

We need tools to quantify pertubations, we need to define norms.

- On data space $\mathbb{R}^{n+1}$: $\|\mathbf{y}\|_2$, $\|\mathbf{y}\|_\infty$
- On result space $\mathcal{P}_n \equiv$ a function space on $I \subset \mathbb{R}$

$$\text{supremum norm} \quad \|f\|_{L^\infty(I)} := \sup\{|f(t)|: t \in I\}\,, \tag{5.2.4.5}$$

$$L^2\text{-norm} \quad \|f\|^2_{L^2(I)} := \int_I |f(t)|^2\,\mathrm{d}t\,, \tag{5.2.4.6}$$

$$L^1\text{-norm} \quad \|f\|_{L^1(I)} := \int_I |f(t)|\,\mathrm{d}t\,. \tag{5.2.4.7}$$

(5.2.4.6) and (5.2.4.7) can be viewed as the function version of $\|\mathbf{y}\|_2$, $\|\mathbf{y}\|_1$, respectively.

we defined the norm in the function space (from the perspective of operator learning, this is the concept of the mapping between finite- and infinite-dimensional space) For linear interpolation operator, the input space is finite $\mathbb{R}^n$, but the resulting space is infinite (a continuous function space).

Sensitivity of linear (problem) maps

$\mathbf{I}_\mathcal{T}(\mathbf{y}+\delta\mathbf{y}) = \mathbf{I}_\mathcal{T}(\mathbf{y}) + \mathbf{I}_\mathcal{T}(\delta\mathbf{y})$ which gives rise to that pertubation: $\delta\mathbf{y} \longrightarrow \mathbf{I}_\mathcal{T}(\delta\mathbf{y})$

$$\|\mathsf{L}\|_{X\to Y} := \sup_{\delta\mathbf{x}\in X\setminus\{\mathbf{0}\}} \frac{\|\mathsf{L}(\delta\mathbf{x})\|_Y}{\|\delta\mathbf{x}\|_X}\,. \tag{5.2.4.9}$$

We refer it as operator norm of $\mathbf{I}_\mathcal{T}$ (Note!! This norm is not used for data space or result space, but for the mapping. It can be viewed as the general form of matrix norm, because matrix can be viewed as a linear mapping. Another thing that we need to be careful is that sensitivity is depended on the definition of norm.)

Due to fact that the operator norm relys on norms, we use ∞ to derive the sensitivity:

$$\|\mathsf{I}_\mathcal{T}(\mathbf{y})\|_{L^\infty(I)} = \left\|\sum_{j=0}^n y_j L_j\right\|_{L^\infty(I)} \le \sup_{t\in I}\sum_{j=0}^n |y_j||L_j(t)| \le \|\mathbf{y}\|_\infty \left\|\sum_{i=0}^n |L_i|\right\|_{L^\infty(I)}$$

Lemma 5.2.4.10. Absolute conditioning of polynomial interpolation

Given a mesh $\mathcal{T} \subset \mathbb{R}$ with generalized Lagrange polynomials L_i, $i = 0,\dots,n$, and fixed $I \subset \mathbb{R}$, the norm of the interpolation operator satisfies

$$\|\mathsf{I}_\mathcal{T}\|_{\infty\to\infty} := \sup_{\mathbf{y}\in\mathbb{R}^{n+1}\setminus\{0\}} \frac{\|\mathsf{I}_\mathcal{T}(\mathbf{y})\|_{L^\infty(I)}}{\|\mathbf{y}\|_\infty} = \left\|\sum_{i=0}^n |L_i|\right\|_{L^\infty(I)}\,, \tag{5.2.4.11}$$

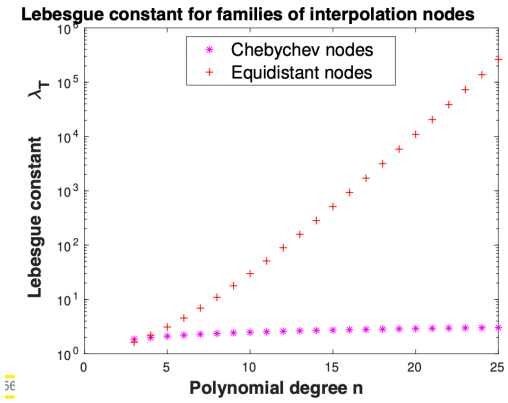
$$\|\mathsf{I}_\mathcal{T}\|_{2\to 2} := \sup_{\mathbf{y}\in\mathbb{R}^{n+1}\setminus\{0\}} \frac{\|\mathsf{I}_\mathcal{T}(\mathbf{y})\|_{L^2(I)}}{\|\mathbf{y}\|_2} \le \left(\sum_{i=0}^n \|L_i\|^2_{L^2(I)}\right)^{\frac{1}{2}}\,. \tag{5.2.4.12}$$

- $\|\sum_{i=0}^n |L_i|\|_{L^\infty(I)}$ depends on $\mathcal{T}$ (interpolated nodes)

- we define Lebesgue constant of $\mathcal{T}$: $\lambda_{\mathcal{T}} := \left\| \sum_{i=0}^{n} |L_i| \right\|_{L^\infty(I)} = \|I_{\mathcal{T}}\|_{\infty \to \infty}$

Seems like if we apply differnt interpolated nodes will give rise to different **Lebesgue constant**, resulting in different sensitivity. See following results:

Node set $\mathcal{T}_n = \{-1 + 2\frac{k}{n}\}_{k=0}^{n} \subset I \equiv [-1, 1]$



- Sophisticated theory gives a lower bound for the Lebesgue constant for uniformly spaced nodes:$\lambda_{\mathcal{T}} \geq Ce^{n/2}$ with $C > 0$ independent of n. (expontential growth)

Due to potentially “high sensitivity” interpolation with global polynomials of high degree is not suitable for data interpolation.

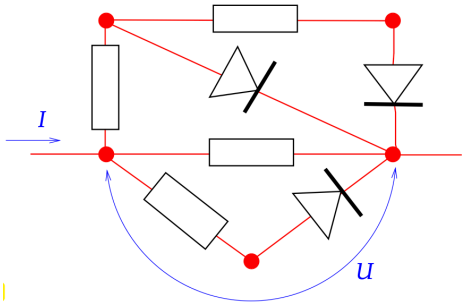
---

5 Approximation of Functions in 1D

5.1 Introduction

Problem background and motivation

Examples: Model reduction in circuit simulation



- Problem description: 2-port circuite element described by $I = I(U)$. $I(U)$ is computable $\forall U$, but this is expensive
- Idea: replace $U \to I(U)$ with $U \to \tilde{I}(U)$, that (1) allows fast evaluation; (2) is "close to" $I(U)$.
- $\tilde{I} \equiv$ surrogate function [model reduction]

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;
};
```

Different from data interpolation, now we are free to choose the number and location of the data points, because we can simply evaluate the function $U \to I(U)$ for any U and as often as needed.

General function approximation problem

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)` ($\to$ Rem. 5.1.0.9).

Goal: Find a *"simple"* function $\widetilde{\mathbf{f}} : D \mapsto \mathbb{R}^d$ such that the approximation error $\mathbf{f} - \widetilde{\mathbf{f}}$ is *"small"*.

Let's clearify the meaning of some terms:

- "Procedural form": only point evaluation possible. Don't have a explicit function formula. This is why we will discuss data interpolation for appromation after.
- "Approximation error $\mathbf{f} - \widetilde{\mathbf{f}}$": is a function
-

  Made more precise: *"simple"*

  The function $\widetilde{\mathbf{f}}$ can be encoded by a small amount of information and is easy to evaluate. For instance, this is the case for polynomial or piecewise polynomial $\widetilde{\mathbf{f}}$.

  Made more precise: *"small"* approximation error

  $\left\| \mathbf{f} - \widetilde{\mathbf{f}} \right\|$ is small for some norm $\|\cdot\|$ on the space $C^0(\overline{D})$ of (piecewise) continous functions.

- Needed: norms on function spaces:
  - The supremum norm: $\|\mathbf{g}\|_\infty := \|\mathbf{g}\|_{L^\infty(D)} := \max_{x \in D} |\mathbf{g}(x)|$

    If the approximation error is small with respect to the supremum, $\tilde{\mathbf{f}}$ is also called a good uniform approximant of $\mathbf{f}$.
  - The L^2-norm: $\|\mathbf{g}\|_2^2 := \|\mathbf{g}\|_{L^2(D)}^2 = \int_D |\mathbf{g}(x)|^2 dx$, see this for more norms about function space.

Below we consider only the case $n = d = 1$: approximation of scalar valued functions defined on an interval. The techniques can be applied componentwise in order to cope with the case of vector valued function ($d > 1$).

Difference between approximation and interpolation

Interpolation scheme + sampling $\to$ 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 $t_i \in I$

Compared with data interpolation, we have two new things:

1. Dataset is not given, we need to generate data by ourself → Sampled methods
2. Defined norm for the function error.

5.2 Approximation by Global Polynomials

5.2.1 Theory

Polynomials: these polynomials are discussed when we talk about data interpolation, and can be used as the basis for data interpolation.

- (Uni-variate)/(general) polynomials → moninomals
- Lagrange polynomials → lagrange basis
- Newton polynomials → newton basis

Confusing cocepts: these two concepts are sometimes confused, but they are not polynomials or not used for interpolation.

- ~~Piece-wise polynomials~~ ⇒piece-wise interpolation [Discussed when we talk about interpolation]
- Taylor polynomials ⇒ for approximating function [Not used for interpolation, but for approximation. If your function is analytic, we can use Taylor polynomials with ∞ degree. for accurate approximation. But it should be noted that the result can only be accurate into the region of convergence. If not in the domain, we need to use complex tools.] - 通常是taylor series

Some polynomials are introduced based on data interpolation, for example Lagrange/Newton. Basis of these polynomials is based on interpolated nodes. And the degree of these polynomials is the same as the number of interpolation points. Some polynomials are not related to the interpolation, including Taylor and general polynomials.

Background

We found that: Taylor polynomials are "natural" for approximation. For $f \in C^k(I)$, $k \in \mathbb{N}$, $I \subset R$, we approximate

$$f(t) \approx \underbrace{f(t_0) + f'(t_0)(t-t_0) + \frac{f^{(2)}(t_0)}{2}(t-t_0)^2 + \cdots + \frac{f^{(k)}(t_0)}{k!}(t-t_0)^k}_{=:T_k(t) \in \mathcal{P}_k}, \quad \text{for some } t_0 \in I\,.$$

- Local approximation of *sufficiently smooth* functions by polynomials is a key idea in calculus, which manifests itself in the importance of approximation by Taylor polynomials.
- In real cases, the function is the procedural form, we cannot use such methods.

Theory

Taylor polynomials corresponding to Taylor expansion of a function, which tells us polynomials are "natural" for approximation.

⇓ The question is,

1. Whether polynomials still offer uniform approximation on arbitrary bounded closed intervals and for functions that are merely continuous, but not any smoother.
2. How well polynomials can approxmate function beyond Taylor?

⇓ The answer for (1) is YES and this profound result is known as the Weierstrass Approximation Theorem. Here we give an extended version with a concrete formula due to Bernstein, see as follows:

Uniform approximation theorem:

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 $0 \le k \le m$.

✎ Notation: $g^{(k)} \triangleq k$-th derivative of a function $g : I \subset \mathbb{R} \to \mathbb{K}$.

- This theorem tells us any continuous function can be approximated by a polynomial (Bernstein approximant) with enough degrees. Taylor also needs the existence of derivates for the function.
- Good news is that there is little restriction for the funtion
- Bad news is that we have no further informaiton about $\|f - p_n\|_\infty \to 0$: how fast/well the approximated function compared with the original function. (也就是当degree增加时，逼近的程度逐渐怎么变化。Taylor can give us truncated term for analysis.)

⇓ AND Jackson therorem tells us more about the bad news if it has more smoothness of the original function. We can derive the upper bound for the best approximation polynomials with degree n. [Review the definition of inf & sup]

Definition 6.2.1.10. (Size of) best approximaton error

Let $\|\cdot\|$ be a (semi-)norm on a space X of functions $I \mapsto \mathbb{K}$, $I \subset \mathbb{R}$ an interval. The (size of the) best approximation error of $f \in X$ in the space $\mathcal{P}_k$ of polynomials of degree $\le k$ with respect to $\|\cdot\|$ is

$$\operatorname{dist}_{\|\cdot\|}(f, \mathcal{P}_k) := \inf_{p \in \mathcal{P}_k} \|f - p\| .$$

- The best approximation polynomial p of degree k with respect to norm $\|\cdot\|$.

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 $$\left\|f^{(r)}\right\|_{L^\infty([-1,1])} := \max_{x \in [-1,1]} |f^{(r)}(x)|.$$

- And ↑ Jackson theorem gives us the upper bound based on norm L_∞ for the best approximation error, which means that if we can figure out the best approximation polynomial, the error of it will be less than that. Upper bound for max-norm of best-approximation error
- NOTE that the Jackson theorem is not relevant to the Bernstein approximant.

Bernstein tells us that the universal approximation theorem of polynomials (Bernstein approximant). Jackson theorm tells us the quantitatively change of the best-approximation error with the change of the degree of polynomial. But they are both theoretical analysis, just like Neural Network, it didn't tell us the concrete form of best approximation polynomials

WE can see that this bound formula is a little difficult, for a simpler bound: reveals asymptotics for $n \to \infty$

$$\inf_{p \in \mathcal{P}_n} \|f - p\|_{L^\infty([-1,1])} \leq C(r) n^{-r} \left\| f^{(r)} \right\|_{L^\infty([-1,1])}$$

with $C(r)$ dependent on r, but *independent of* f and, in particular, the polynomial *degree* n. Using the Landau symbol from chapter 1 we can rewrite the statement of (6.2.1.13) in asymptotic form: $\inf_{p \in \mathcal{P}_n} \|f - p\|_{L^\infty([-1,1])} = O(n^{-r})$ for $n \to \infty$

- In chapter 1, the Landau symbol is used for asymptoic complexity.
- This asymptoic form is very important, which tells us the convergence speed of the upper bound for best approximation. And it is referred as algebraic convergence with rate r for $n \to \infty$. But this upper bound is very loose, we can further decrease it to achieve expontential convergence. (see the following chapters)
- And we still need to understand that these formula tells us the expressivity of the polynomials for function approximation (just theory analysis). It didn't tell us how to derive such best approximation polynomial for function. We can image that if we want to derive the expressivity of neural networks for function approximation, we can use the similiar trick. And the result is that we can derive the upper bound of best approximation error for function.

Transformation of approximation error estimation

Jackson theory only tell us the upper bound for function approximation on the interval [-1,1], can it tell us the upper bound on [a,b]? Let's first consider that whether we can transform a polynomial approximation scheme from [-1,1] to [a,b].

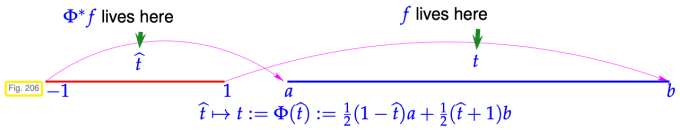
⇓ Tool: Bijective affine linar transformation

Assume that an interval $[a, b] \subset \mathbb{R}$, $a < b$, and a polynomial approximation scheme $\hat{A} : C^0([-1,1]) \to \mathcal{P}_n$. Based on the affine linear mapping

$$\Phi : [-1,1] \to [a,b] \quad , \quad \Phi(\hat{t}) := a + \tfrac{1}{2}(\hat{t}+1)(b-a) \, , \quad -1 \leq \hat{t} \leq 1$$

we can introduce the affine pullback of functions:

$$\Phi^* : C^0([a,b]) \to C^0([-1,1]) \quad , \quad \Phi^*(f)(\hat{t}) := (f \circ \Phi)(\hat{t}) := f(\Phi(\hat{t})) \, , \quad -1 \leq \hat{t} \leq 1$$



- ! Remember $\Phi^*(f)(\hat{t})$
- Transformation of norms under pullback:

Lemma 6.2.1.20. Transformation of norms under affine pullbacks

For every $f \in C^0([a,b])$ we have

$$\|f\|_{L^\infty([a,b])} = \|\Phi^* f\|_{L^\infty([-1,1])} \quad , \quad \|f\|_{L^2([a,b])} = \sqrt{|b-a|}\,\|\Phi^* f\|_{L^2([-1,1])} \,. \tag{6.2.1.21}$$

- $p \in \mathcal{P}_n \Rightarrow \Phi^* p \in \mathcal{P}_n$
- Transformation of norms under affine pullbacks don't change.

Pullback & derivation ($\Phi \leftrightarrow$ dilation)

By the 1D chain rule: $\frac{d}{d\hat{t}}(\Phi^* f)(\hat{t}) = \frac{df}{dt}(\Phi(\hat{t}))\frac{d\Phi}{d\hat{t}} = \frac{df}{dt}(\Phi(\hat{t}))\cdot\frac{1}{2}(b-a)$, which implies a simple scaling rule for derivatives of arbitrary order $r \in \mathbb{N}_0$:

$$(\Phi^* f)^{(r)} = \left(\frac{b-a}{2}\right)^r \Phi^*(f^{(r)}) \,. \tag{6.2.1.24}$$

Lemma 6.2.1.20

$$\left\|(\Phi^* f)^{(r)}\right\|_{L^\infty([-1,1])} = \left(\frac{b-a}{2}\right)^r \left\|f^{(r)}\right\|_{L^\infty([a,b])} \,, \quad f \in C^r([a,b]), r \in \mathbb{N}_0 \,. \tag{6.2.1.25}$$

Polynomial best-approximation error estimation on general intervals

$$\begin{aligned} \inf_{p\in\mathcal{P}_n} \|f-p\|_{L^\infty([a,b])} &\overset{(*)}{=} \inf_{p\in\mathcal{P}_n} \|\Phi^* f - p\|_{L^\infty([-1,1])} \\ &\overset{\text{Thm. 6.2.1.11}}{\leq} (1+\pi^2/2)^r \frac{(n-r)!}{n!} \left\|(\Phi^* f)^{(r)}\right\|_{L^\infty([-1,1])} \\ &\overset{(6.2.1.24)}{=} (1+\pi^2/2)^r \frac{(n-r)!}{n!} \left(\frac{b-a}{2}\right)^r \left\|f^{(r)}\right\|_{L^\infty([a,b])} \end{aligned}$$

And we end up with the simpler bound

$$f \in C^r([a,b]) \;\Rightarrow\; \inf_{p\in\mathcal{P}_n} \|f-p\|_{L^\infty([a,b])} \leq C(r)\left(\frac{b-a}{n}\right)^r \left\|f^{(r)}\right\|_{L^\infty([a,b])}$$

Transformation of polynomial approximation schemes

We use affline pullback function for us to analyze the upper bound of best approximaiton error with interval [a,b] in above discussion. Here, we can also use it for transformation of polynomial approximation schemes in different intervals.

We can define a polynomial approximation scheme A on $C^0([a,b])$ by

$$A : C^0([a,b]) \to \mathcal{P}_n \quad , \quad \boxed{A := (\Phi^*)^{-1} \circ \hat{A} \circ \Phi^*} \,,$$

when $\hat{A}$ is a polynomial approximation scheme on [-1,1].

5.2.2 Error Estimates for Polynomial Interpolation

Introduction

The choice of node set is very important, which will help us build the mapping (interpolation scheme/operator and approximation scheme/operator). $\mathbf{I}_\mathcal{T}$ interpolation scheme/operator is based on nodes.

Interpolation scheme + sampling $\rightarrow$ 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 $t_i \in I$

NOW, we introduce Lagrangian approximation scheme.

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

$$\mathsf{L}_{\mathcal{T}}(f) := \mathsf{I}_{\mathcal{T}}(\mathbf{y}) \in \mathcal{P}_n \quad \text{with} \quad \mathbf{y} := [f(t_0), \ldots, f(t_n)]^T \in \mathbb{R}^{n+1},$$
$$\mathsf{I}_{\mathcal{T}}(\mathbf{y})(t_j) = (\mathbf{y})_j, \quad j = 0, \ldots, n.$$

- $\mathbf{L}_{\mathcal{T}}(f)$: mapping function space $\to$ function space; $\mathbf{I}_{\mathcal{T}}$ mapping vector space $\to$ function space.

Convergence of interpolation errors

Consider sequencies of node sets $(\mathcal{T}_n)_{n\in\mathbb{N}}$, the number of $\mathcal{T} = n + 1 \leftrightarrow$ degree of polynomials

Example: I = [a,b]: equidistant nodes

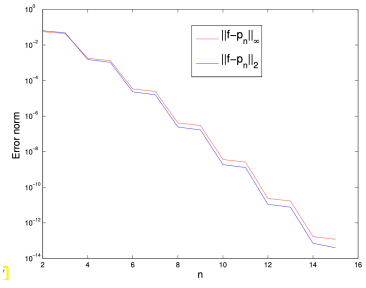
$$\mathcal{T}_n := \{t_j^{(n)} := a + (b-a)\frac{j}{n} : j = 0, \ldots, n\} \subset I$$

For families of Lagrange interpolation schemes $\{\mathbf{L}_{\mathcal{T}_n}\}_{n\in\mathbb{N}_0}$ we can shift the focus onto estimating the asymptotic behavior of the norm of the interpolation error for $n \to \infty$.

$$\exists\, C \neq C(n) > 0: \quad \|f - \mathsf{L}_{\mathcal{T}_n} f\| \leq C\, T(n) \quad \text{for } n \to \infty. \qquad (6.2.2.6)$$

- $C \neq C(n)$ mean: independent of n!
- $C \to$ "generic constant", generally unknown
- Here, we don't expect to give the accurate upper bound for the approximation error, but want to give the asymptotic convergence of the approximation error.
- NOTICE: Here, we didn't discuss best approximation error. the result is based on the approximation scheme scheme/nodes we choose.

Asymptoic behavior of lagrange interpolation error



In the numerical experiment the norms of the interpolation errors can be computed only approximately as follows.

- L^∞-norm: approximated by sampling on a grid of meshsize $\pi/1000$.
- L^2-norm: numerical quadrature ($\to$ Chapter 7) with trapezoidal rule (7.5.0.4) on a grid of meshsize $\pi/1000$.

◁ approximate norms $\|f - \mathsf{L}_{\mathcal{T}_n} f\|_*$, $* = 2, \infty$.

Classification of asymptoic 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*:

$$\begin{array}{lll} \exists\, p > 0: & T(n) \leq n^{-p} & : \text{ algebraic convergence, with rate } p > 0, \\ \exists\, 0 < q < 1: & T(n) \leq q^n & : \text{ exponential convergence}, \end{array} \quad \forall n \in \mathbb{N}.$$

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.

Determining type of convergence in numerical experiments

"Measured": $(n_i, \varepsilon_i) \in \mathbb{N} \times \mathbb{R}^+$ for $i = 1, 2, \ldots$, where ε means error norm

Conjectured: algebraic convergence: $\epsilon_i \approx C n_i^{-p}$

$\log(\epsilon_i) \approx \log(C) - p \log n_i$ (affine linear in log-log scale) .

$\log(\varepsilon_i/\varepsilon_{i-1}) \approx -p \log(n_i/n_{i-1}) \to p \quad n \to ?$ for $\varepsilon \to \varepsilon/2$ 这种直接列出两个误差的比较值即可

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) .

By numerical experiments, we can know the asymptoic convergence of an approximation operator.

5.2.2.1 Convergence of Interpolation Errors

5.2.2.2 Interpolation of Finite Smoothness

Though polynomials possess great power to approximate functions, see Bernstein approximation scheme, here polynomial interpolants fail completely. Approximation theorists even discovered the following "negative result":

Just like artificial neural network, although it possesses great power for approximating functions, we cannot express its power at most cases.

Theorem 6.2.2.13. Divergent polynomial interpolants

Given a sequence of meshes of increasing size $\{\mathcal{T}_n\}_{n=1}^{\infty}$, $\mathcal{T}_j = \{t_0^{(n)}, \ldots, t_n^{(n)}\} \subset [a,b]$, $a \le t_0^{(n)} < t_2^{(j)} < \cdots < t_n^{(n)} \le b$, there exists a continuous function f such that the sequence of interpolating polynomials $(\mathsf{L}_{\mathcal{T}_n} f)_{n=1}^{\infty}$ does not converge to f uniformly as $n \to \infty$.

Now we aim to establish bounds for the supremum norm of the interpolation error of Lagrangian interpolation similar to the result of Jackson's best approximation theorem.

It states a result for at least continuously differentiable functions and its bound for the polynomial best approximation error involves norms of certain derivatives of the function f to be approximated. Thus some smoothness of f is required, but only a few of derivatives need exist. Thus, we say, that this chapter deals with functions of finite smoothnes, that is, $f \in C^k$ for some $k \in \mathbb{N}$. In this section we aim to bound polynomial interpolation errors for such functions.

5.2.2.3 Analytic Interpolands

Motivation

We have seen that for some Lagrangian approximation schemes applied to certain functions we can observe exponential convergence of the approximation error for increasing polynomial degree. This section presents a class of interpolands, which often enable this convergence.

Example: Hump function

whose radius of convergence $= 1$. More generally, deeper theory tells us that the Taylor series at t_0 has radius of convergence $= \sqrt{t_0^2 + 1}$. Thus, in $[-5, 5]$ cannot be represented by a single power series, though it is a perfectly smooth function. This is very interesting! This function can only be accurately calculated by Taylor series on small sub-intervals.

In last chapter, we discussed about the blow-up of the interpolation error with degree of the polynomial for hump function.

Remark: Analytic function everywhere

Approximation by truncated power series

Residue calculus

Why go $\mathbb{C}$? The reason is that this permits us to harness powerful tool from complex analysis, a field of mathematics, which studies analytic functions. One of theses tools is the residue theorem.

Theorem 6.2.2.50. Residue theorem [Rem84, Ch. 13]

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 $D \setminus \Pi$ 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 $p \in \mathbb{C}$.

- p is a complex point. Π often stands for the set of poles of f, that is, points where "f Attains the value ∞".

5.2.3 Chebychev Interpolation

5.2.3.1 Motivation and Definition

Now, based on the insight into the structure of the interpolation error gained before, we seek to choose "optimal" sampling points. They will give rise to the so-called Chebychev polynomial approximation schemes, also known as Chebychev interpolation.

$$\|f - \mathrm{L}_\mathcal{T} f\|_{L^\infty(I)} \leq \frac{\|f^{(n+1)}\|_{L^\infty(I)}}{(n+1)!} \max_{t \in I} |(t - t_0) \cdot \cdots \cdot (t - t_n)|$$

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!)

Of course, an *a posteriori* choice based on information gleaned from evaluations of the interpoland f may yield much better interpolants (in the sense of smaller norm of the interpolation error). Many modern algorithms employ this a posteriori adaptive approximation policy, but this chapter will not cover them.[See Problem 6-3]

5.2.3.2 Chebychev Interpolation Error Estimates

6 Numerical Quature

6.1 Introduction

6.2 Quadrature Formulas/Rules

6.3 Polynomial Quadrature Formulas

6.4 Gauss Quadrature

6.4.1 Order of a Quadrature Rule

6.4.2 Maximal-Order Quadrature Rules

6.4.3 Gauss-Legendre Quadrature Error Estimates

6.5 Composite Quadrature

6.6 Adaptive Quadrature

7 Iterative Methods for Non-Linear Systems of Equations

Iterative Methods for $F(\mathbf{x}) = 0$

1. Fixed point iteration - analytical form of F。后面的是procedural form即可，point evalutaion。
2. Bisection - Stable and derivative free method
3. Newton Methods - 一阶和二阶连续要求
4. Multi-point - 和Newton Methods同属于model function points,replace F with interpolating polynomial. Didn't use derivatives information of $F(\mathbf{x})$
5. secant method - based on inverse interpolation,这个方法真的好妙啊。发挥了interpolation的优势和procedural form的优势。

下面定义了iterative methods方法的重要metrics，asymptotic efficiency efficiencey = # digits gained / total work required

8 Numerical Integration - Single Step Methods

8.1 Initial-Value Problems (IVPs) for ODEs

8.1.1 Ordinary Differential Equations (ODEs)

Solution of ODE:

Definition 11.1.1.2. Solution of an ordinary differential equation

A solution of the ODE $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$ with continuous right hand side function $\mathbf{f}$ is a continuously differentiable *function* "of time t" $\mathbf{y} : J \subset I \to D$, defined on an open interval J, for which $\dot{\mathbf{y}}(t) = \mathbf{f}(t, \mathbf{y}(t))$ holds for all $t \in J$ ($\hat{=}$ "*pointwise*").

- J is an open interval for time.
- A solution describes a continuous trajectory in state space, a one-parameter family of states, parameterized by time.
- right hand side relys on time and state. For and atonomous ODE (right hand side function does not depend on time, but only on state), the right hand side function defines a vector field.

Smoothness of solutions inherited from the right hand side function.

Lemma 11.1.1.3. Smoothness of solutions of ODEs

Let $\mathbf{y} : I \subset \mathbb{R} \to D$ *be a solution of the ODE* $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ *on the time interval* I.

If $\mathbf{f} : D \to \mathbb{R}^N$ *is* r*-times continuously differentiable with respect to both arguments,* $r \in \mathbb{N}_0$*, then the trajectory* $t \mapsto \mathbf{y}(t)$ *is* $r+1$*-times continuously differentiable in the interior of* I.

A simplest class of ODEs, linear ordinary differential equations

Definition 11.1.1.9. Linear first-order ODE

A first-order ordinary differential equation $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$, as introduced in § 11.1.1.1 is **linear**, if

$$\mathbf{f}(t, \mathbf{y}) = \mathbf{A}(t)\mathbf{y} \quad , \quad \mathbf{A} : I \to \mathbb{R}^{N,N} \quad \text{a continuous function .} \tag{11.1.1.10}$$

Lemma 11.1.1.11. Space of solutions of linear ODEs

The set of solutions $\mathbf{y} : I \to \mathbb{R}^N$ *of* (11.1.1.10) *is a vector space.*

8.1.2 Mathematical Modeling with Ordinary Differention Equations

8.1.3 Theory of IVPs

A generic **Initial value problem** (IVP) for a first-order ordinary differential equation (ODE) (→ [Str09, Sect. 5.6], [DR08, Sect. 11.1]) can be stated as:

Find a function $\mathbf{y} : I \to D$ that satisfies, *cf.* Def. 11.1.1.2,

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \quad , \quad \mathbf{y}(t_0) = \mathbf{y}_0 \, . \tag{11.1.3.2}$$

- $\mathbf{f} : I \times D \mapsto \mathbb{R}^N \triangleq$ right hand side (r.h.s.) $(N \in \mathbb{N})$,
- $I \subset \mathbb{R} \triangleq$ (time)interval $\leftrightarrow$ "time variable" t,
- $D \subset \mathbb{R}^N \triangleq$ **state space**/phase space $\leftrightarrow$ "state variable" $\mathbf{y}$,
- $\Omega := I \times D \triangleq$ extended state space (of tupels $(t, \mathbf{y})$),
- $t_0 \in I \triangleq$ initial time, $\mathbf{y}_0 \in D \triangleq$ initial state $\succ$ **initial conditions**.

⇓ IVPs for autonomous ODEs

Hence, for autonomous ODEs we have $I = \mathrm{R}$ and the right hand side function $\mathbf{y} \to \mathbf{f}(\mathbf{y})$ can be regarded as a stationary vector field (velocity field), see Fig. 394 or Fig. 397. An important observation: If $t \to \mathbf{y}(t)$ is a solution of an autonomous ODE, then, for any $\tau \in \mathrm{R}$, also the shifted function $t \to \mathbf{y}(t - \tau)$ is a solution.

Initial time for autonomous ODEs

For initial value problems for autonomous ODEs the initial time is irrelevant and therefore we can always make the "canonical" choice $t_0 = 0$.

⇓ From high-order ODEs to first order systems

For ODEs of order $n \in \mathbb{N}$ well-posed initial value problems need to specify initial values for the first $n-1$ derivatives.

⇓ Smoothness classes for right-hand side functions

Now we review results about existence and uniqueness of solutions of initial value problems for first-order ODEs. These are surprisingly general and do not impose severe constraints on right hand side functions. Some kind of smoothness of the right-hand side function $\mathbf{f}$ is required, nevertheless and the following definitions describe it in detail.

Definition 11.1.3.11. Lipschitz continuous function (→ [Str09, Def. 4.1.4])

Let $\Theta := I \times D$, $I \subset \mathbb{R}$ an interval, $D \subset \mathbb{R}^N$, $N \in \mathbb{N}$, an open domain. A function $\mathbf{f} : \Theta \mapsto \mathbb{R}^N$ is Lipschitz continuous (in the second argument) on Θ, if

$$\exists L > 0: \quad \|\mathbf{f}(t,\mathbf{w}) - \mathbf{f}(t,\mathbf{z})\| \leq L\|\mathbf{w}-\mathbf{z}\| \quad \forall (t,\mathbf{w}),(t,\mathbf{z}) \in \Theta\,. \tag{11.1.3.12}$$

The following is the the most important mathematical result in the theory of initial-value problems for ODEs:

Theorem 11.1.3.17. Theorem of Peano & Picard-Lindelöf [Ama83, Satz II(7.6)], [Str09, Satz 6.5.1], [DR08, Thm. 11.10], [Han02, Thm. 73.1]

If the right hand side function $\mathbf{f} : \Omega \mapsto \mathbb{R}^N$ is locally Lipschitz continuous (→ Def. 11.1.3.13) then for all initial conditions $(t_0, \mathbf{y}_0) \in \Omega$ the IVP

$$\dot{\mathbf{y}} = \mathbf{f}(t,\mathbf{y}) \quad , \quad \mathbf{y}(t_0) = \mathbf{y}_0\,. \tag{11.1.3.2}$$

has a solution $\mathbf{y} \in C^1(J(t_0,\mathbf{y}_0),\mathbb{R}^N)$ with maximal (temporal) domain of definition $J(t_0,\mathbf{y}_0) \subset \mathbb{R}$.

- Explain the domain of definition of solutions of IVPs
  - Solutions of an IVP have an *intrinsic* maximal domain of definition
  - domains of definition $J(t_0,\mathbf{y}_0)$: that is the domain of the solution for ode with the initial conditions $(t_0,\mathbf{y}_0)$. The reason why we define this is that we found the solution may be blow-up under specific initial conditions. Intrinsic life-span of solutions, for maximal (temporal) domain of definition

8.1.4 Evolution Operators

⇓ Evolution Operators

Now we examine a difficult but fundamental concept for time-dependent models stated by means of ODEs. For the sake of simplicity we restrict the discussion to autonomous initial-value problems (IVPs) (NOTE that the discussion about the evolution operator is under the restriction of IVPs)

with locally Lipschitz continuous (→ Def. 11.1.3.13) right hand side $\mathbf{f} : D \subset \mathrm{R}N \to \mathrm{R}N$, $N \in$ N, and make the following assumption. A more general treatment is given in

Assumption 11.1.4.1. Global solutions

Solutions of (11.1.3.18) are global: $J(\mathbf{y}_0) = \mathbb{R}$ for *all* $\mathbf{y}_0 \in D$.

Now we return to the study of a generic ODE (ODE) instead of an IVP (11.1.3.2). We do this by temporarily changing the perspective: we fix a "time of interest" $t \in \mathrm{R} \setminus \{0\}$ and follow all trajectories for the duration t. This induces a mapping of points in state space:

$$\succ \text{ mapping } \quad \boldsymbol{\Phi}^t : \begin{cases} D & \mapsto & D \\ \mathbf{y}_0 & \mapsto & \mathbf{y}(t) \end{cases} , \quad t \mapsto \mathbf{y}(t) \text{ solution of IVP (11.1.3.18)}$$

Now, we may also let t vary, which spawns a *family* of mappings $\{\boldsymbol{\Phi}^t\}_{t\in\mathbb{R}}$ of the state space D into itself. However, it can also be viewed as a mapping with two arguments, a duration t and an initial state value $\mathbf{y}_0$!

Definition 11.1.4.3. Evolution operator/mapping

Under Ass. 11.1.4.1 the mapping

$$\boldsymbol{\Phi} : \begin{cases} \mathbb{R} \times D & \mapsto & D \\ (t, \mathbf{y}_0) & \mapsto & \boldsymbol{\Phi}^t \mathbf{y}_0 := \mathbf{y}(t) \end{cases} ,$$

where $t \mapsto \mathbf{y}(t) \in C^1(\mathbb{R}, \mathbb{R}^N)$ is the unique (global) solution of the IVP $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{y}(0) = \mathbf{y}_0$, is the **evolution operator**/mapping for the autonomous ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$.

Note that $t \mapsto \boldsymbol{\Phi}^t \mathbf{y}_0$ describes the solution of $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ for $\mathbf{y}(0) = \mathbf{y}_0$ (a trajectory). Therefore, by virtue of definition, we have

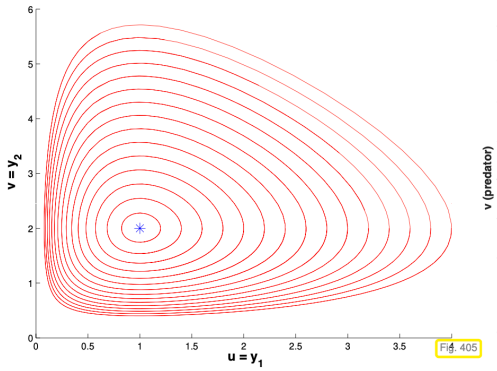
$$\frac{\partial \boldsymbol{\Phi}}{\partial t}(t, \mathbf{y}) = \mathbf{f}(\boldsymbol{\Phi}^t \mathbf{y}) \,. \tag{11.1.4.4}$$

- It can be viewed as a mapping with two arguments, a duration t and an initial state value $\mathbf{y}_0$

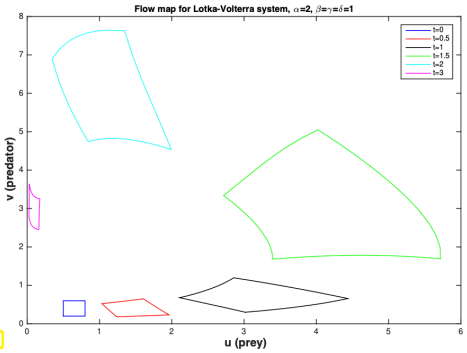
Let us repeat the different kinds of information contained in an evolution operator when viewed from different angles:

$t \mapsto \boldsymbol{\Phi}^t \mathbf{y}_0$, $\mathbf{y}_0 \in D$ fixed $\quad \widehat{=}$ a trajectory = solution of an IVP ,

$\mathbf{y} \mapsto \boldsymbol{\Phi}^t \mathbf{y}$, $t \in \mathbb{R}$ fixed $\quad \widehat{=}$ a mapping of the state space onto itself .



trajectories $t \mapsto \boldsymbol{\Phi}^t \mathbf{y}_0$



state mapping $\mathbf{y} \mapsto \boldsymbol{\Phi}^t \mathbf{y}$

There is a one-to-one relationship between ODEs and their evolution operators, and those are the key objects behind an ODE.

An ODE "encodes" an evolution operator.

Understanding the concept of evolution operators is indispensable for numerical integration, that the is the construction of numerical methods for the solution of IVPs for ODEs:

Numerical integration is concerned with the approximation of evolution operators.

8.2 Introduction: Polygonal Approximation Methods

8.3 General Single-Step Methods

8.3.1 Definition

8.3.2 (Asymptotic) Convergence of Single-Step Methods

Approximation errors in numerical integration are also called discretization errors.

Depending on the objective of numerical integration as stated in § 11.2.0.1 different (norms of) discretization errors are of interest: We need to define our errors for analyzing.

(I) If only the solution at final time T is sought, the relevant norm of the discretization error is

$$\epsilon_M := \|\mathbf{y}(T) - \mathbf{y}_M\| ,$$

where $\|\cdot\|$ is some vector norm on $\mathbb{R}^N$.

(II) If we want to approximate the solution trajectory for (11.1.3.18) the discretization error is the function

$$t \mapsto \mathbf{e}(t) \quad , \quad \mathbf{e}(t) := \mathbf{y}(t) - \mathbf{y}_h(t) ,$$

where $t \mapsto \mathbf{y}_h(t)$ is the approximate trajectory obtained by post-processing, see § 11.3.1.16. In this case accuracy of the method is gauged by looking at norms of the function $\mathbf{e}$, see § 5.2.4.4 for examples.

(III) Between (I) and (II) is the pointwise discretization error, which is the sequence (a so-called grid function)

$$\mathbf{e} : \mathcal{M} \to D \quad , \quad \mathbf{e}_k := \mathbf{y}(t_k) - \mathbf{y}_k , \quad k = 0, \dots, M . \tag{11.3.2.2}$$

In this case one usually examines the maximum error in the mesh points

$$\|(\mathbf{e})\|_\infty := \max_{k \in \{1,\dots,N\}} \|\mathbf{e}_k\| ,$$

where $\|\cdot\|$ is a suitable vector norm on $\mathbb{R}^N$, customarily the Euclidean vector norm.

Specified discrete evolution Ψ, the single step method is fixed:

$$\mathbf{y}_{k+1} := \Psi(t_{k+1} - t_k, \mathbf{y}_k) , \quad k = 0, \dots, M-1 , \tag{11.3.1.6}$$

The only way to control the accuracy of the solution $\mathbf{y}_N$ or $t \mapsto \mathbf{y}_h(t)$ is through the selection of the mesh $\mathcal{M} = \{0 = t_0 < t_1 < \cdots < t_N = T\}$.

- Only way to control the accuracy of the solution is through the selection of the mesh.

  Hence we study convergence of single step methods for families of meshes $\{\mathcal{M}_\ell\}$ and track the decay of (a norm) of the discretization error ($\to$ § 11.3.2.1) as a function of the number $M := \sharp\mathcal{M}$ of mesh points. In other words, we examine **h-convergence**. Convergence through mesh refinement is discussed for piecewise polynomial interpolation in Section 6.6.1 and for composite numerical quadrature in Section 7.5.

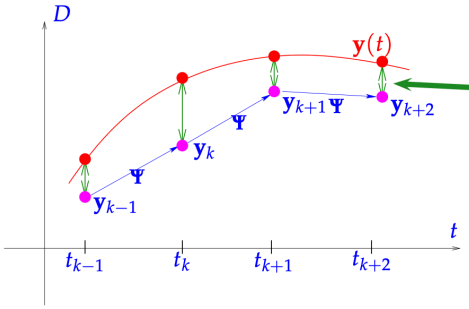
- When investigating asymptotic convergence of single step methods we often resort to families of equidistant meshes of $[0, T]$:

  $$\mathcal{M}_M := \{t_k := \frac{k}{M} T : k = 0 \dots, M\} . \tag{11.3.2.4}$$

  We also call this the use of uniform timesteps of size $h := \frac{T}{M}$. ⌟

Recall the recursion defining the explicit Euler method

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \mathbf{f}(\mathbf{y}_k) , \quad h_k := t_{k+1} - t_k , \quad k = 1, \ldots, M-1 . \tag{11.2.1.5}$$



In numerical analysis one studies the

error sequence: $\mathbf{e}_k := \mathbf{y}_k - \mathbf{y}(t_k)$.

◁ — ≙ trajectory $t \mapsto \mathbf{y}(t)$
— ≙ Euler polygon,
• ≙ $\mathbf{y}(t_k)$,
• ≙ $\mathbf{y}_k$,
⟶ ≙ discrete evolution $\Psi^{t_{k+1}-t_k}$

① Abstract **splitting of error**:

Fundamental error splitting:

$$\begin{aligned} \mathbf{e}_{k+1} &= \Psi^{h_k}\mathbf{y}_k - \Phi^{h_k}\mathbf{y}(t_k) \\ &= \underbrace{\Psi^{h_k}\mathbf{y}_k - \Psi^{h_k}\mathbf{y}(t_k)}_{\text{propagated error}} \\ &\quad + \underbrace{\Psi^{h_k}\mathbf{y}(t_k) - \Phi^{h_k}\mathbf{y}(t_k)}_{\text{one-step error}} . \end{aligned} \tag{11.3.2.12}$$

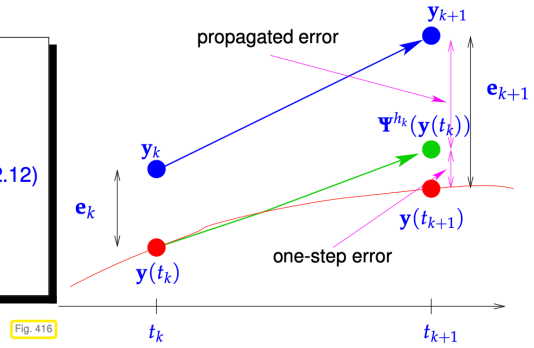


Fig. 416

Total error arises from accumulation of propagated one-step errors!

§11.3.2.20 (One-step error and order of a single step method) In the analysis of the global discretization error of the explicit Euler method in § 11.3.2.9 a one-step error of size $O(h_k^2)$ led to a total error of $O(h)$ through the effect of error accumulation over $M \approx h^{-1}$ steps. This relationship remains valid for almost all single step methods [DB02, Theorem 4.10]:

Order of algebraic convergence of single-step methods

Consider an IVP (11.1.3.2) with solution $t \mapsto \mathbf{y}(t)$ and a single step method defined by the discrete evolution Ψ (→ Def. 11.3.1.5). If the *one-step error along the solution trajectory* satisfies (Φ is the evolution map associated with the ODE, see Def. 11.1.4.3)

$$\left\| \Psi^h \mathbf{y}(t) - \Phi^h \mathbf{y}(t) \right\| \leq C h^{p+1} \quad \forall h \text{ sufficiently small}, \; t \in [0, T] , \tag{11.3.2.22}$$

for some $p \in \mathbb{N}$ and $C > 0$, then, usually,

$$\max_k \|\mathbf{y}_k - \mathbf{y}(t_k)\| \leq \overline{C} h_{\mathcal{M}}^p ,$$

with $\overline{C} > 0$ independent of the temporal mesh $\mathcal{M}$: The (pointwise) discretization error *converges algebraically* with order/rate p.

8.4 Explicit Runge-Kutta Single-Step Methods (RKSSMs)

8.5 Adaptive Stepsize Control

Algorithm building temporal mesh $\mathcal{M} = \{t_k\}_{k=0}^{\mathcal{M}}$ during execution of SSM

9 Single-Step Methods for Stiff Initial-Value Problems

9.1 Model Problem Analysis

9.2 Stiff Initial-Value Problems

这一章主要讲了怎么通过线性化来判断一个ODE是否为Stiff IVP

Notion 12.2.0.7. Stiff IVP

An initial value problem is called stiff, if stability imposes much tighter timestep constraints on *explicit single step methods* than the accuracy requirements.

The short-time evolution of $\mathbf{y}$ with $\mathbf{y}(0) = \mathbf{y}^*$ is approximately governed by the affine-linear ODE

$$\dot{\mathbf{y}} = \mathbf{M}(\mathbf{y} - \mathbf{y}^*) + \mathbf{b}\,, \quad \mathbf{M} := \mathrm{D}\mathbf{f}(\mathbf{y}^*) \in \mathbb{R}^{N,N}\,, \quad \mathbf{b} := \mathbf{f}(\mathbf{y}^*) \in \mathbb{R}^N\,. \tag{12.2.0.10}$$

In the scalar case we have come across this linearization already in Ex. 12.1.0.3. ⌟

⇓ derivation

We *fix* a state $\mathbf{y}^* \in D$, D the state space, write $t \mapsto \mathbf{y}(t)$ for the solution with $\mathbf{y}(0) = \mathbf{y}^*$. We set $\mathbf{z}(t) = \mathbf{y}(t) - \mathbf{y}^*$, which satisfies

$$\mathbf{z}(0) = 0 \quad , \quad \dot{\mathbf{z}} = \mathbf{f}(\mathbf{y}^* + \mathbf{z}) = \mathbf{f}(\mathbf{y}^*) + \mathrm{D}\mathbf{f}(\mathbf{y}^*)\mathbf{z} + R(\mathbf{y}^*, \mathbf{z})\,, \quad \text{with} \quad \|R(\mathbf{y}^*, \mathbf{z})\| = O(\|\mathbf{z}\|^2)\,.$$

This is obtained by Taylor expansion of $\mathbf{f}$ at $\mathbf{y}^*$, see [Str09, Satz 7.5.2]. Hence, in a neighborhood of a state $\mathbf{y}^*$ on a solution trajectory $t \mapsto \mathbf{y}(t)$, the deviation $\mathbf{z}(t) = \mathbf{y}(t) - \mathbf{y}^*$ satisfies

$$\dot{\mathbf{z}} \approx \mathbf{f}(\mathbf{y}^*) + \mathrm{D}\mathbf{f}(\mathbf{y}^*)\mathbf{z}\,. \tag{12.2.0.9}$$

Learning the theory about how to find stiff problem

Remark 12.2.0.16 (Characteristics of stiff IVPs) Often one can already tell from the expected behavior of the solution of an IVP, which is often clear from the modeling context, that one has to brace for stiffness.

Typical features of stiff IVPs:
- Presence of fast transients in the solution, see Ex. 12.1.0.3, Ex. 12.1.0.35,
- Occurrence of strongly attractive fixed points/limit cycles, see Ex. 12.2.0.4

9.3 Implicit Runge-Kutta Single-Step Methods

Recall:

Notion 12.2.0.7. Stiff IVP

An initial value problem is called stiff, if stability imposes much tighter timestep constraints on *explicit single step methods* than the accuracy requirements.

In the previous problem, we only discussed the explicit problem for the Stiff IVP. In this chapter, we started to view some implicit problems.

The goal in this chapter is to introduce a new approach for such still IVP.

Our discussion is located at the complex plane.

Unconditional stability: no timestep constraint for stability analysis. [The potential of implicit method]

$\Downarrow$ We want to give a more general/high-order SSM

Goal: unconditionally stable SSMs of high order