①

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

Numerical Methods for Computational Science and Engineering

Prof. R. Hiptmair, SAM, ETH Zurich

(with contributions from Prof. P. Arbenz and Dr. V. Gradinaru)

Autumn Term 2016

URL: http://www.sam.math.ethz.ch/~hiptmair/tmp/NumCSE/NumCSE16.pdf

II. Direct Methods for LSE

Solve $A\underline{x} = \underline{b}$ $\quad A \in \mathbb{K}^{n,n}, \underline{b} \in \mathbb{K}^n$ given

coefficient matrix $\qquad$ r.h.s vector

Existence & uniqueness of $\underline{x}$:

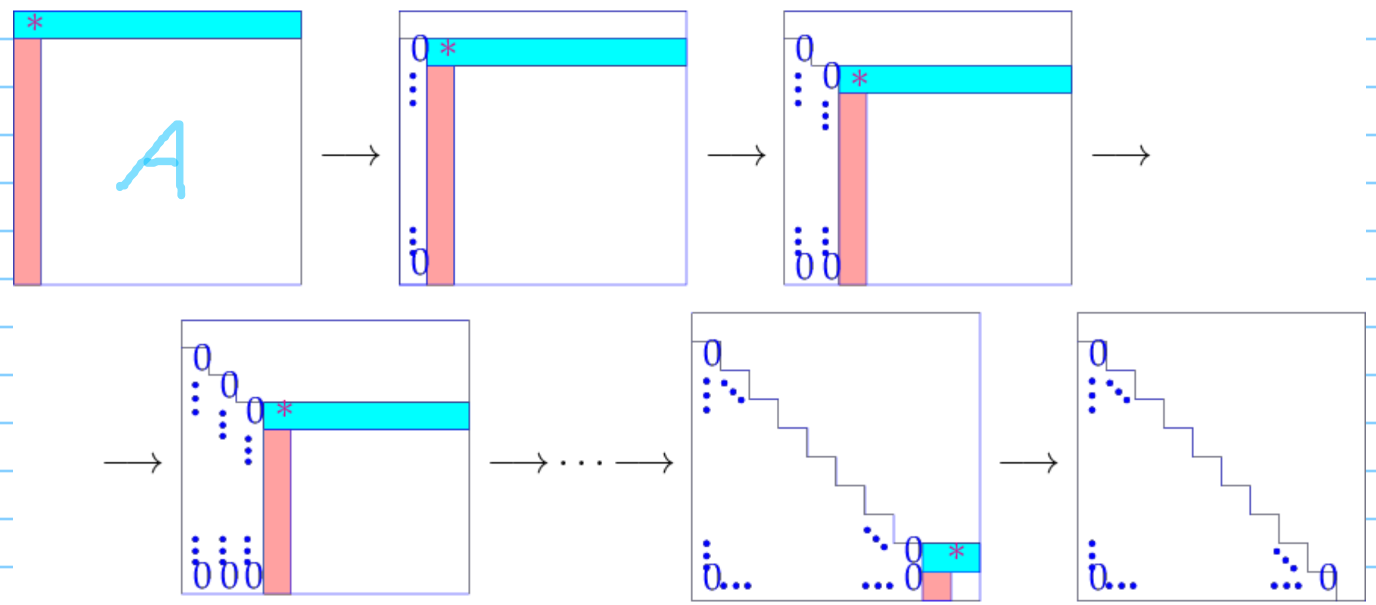
- A regular / invertible : $\exists B \in \mathbb{K}^{n,n} : A \cdot B = I_n$

$\Leftrightarrow$ $\mathrm{Rank}(A) = n \Leftrightarrow \det(A) \neq 0 \Leftrightarrow$ l.i. columns

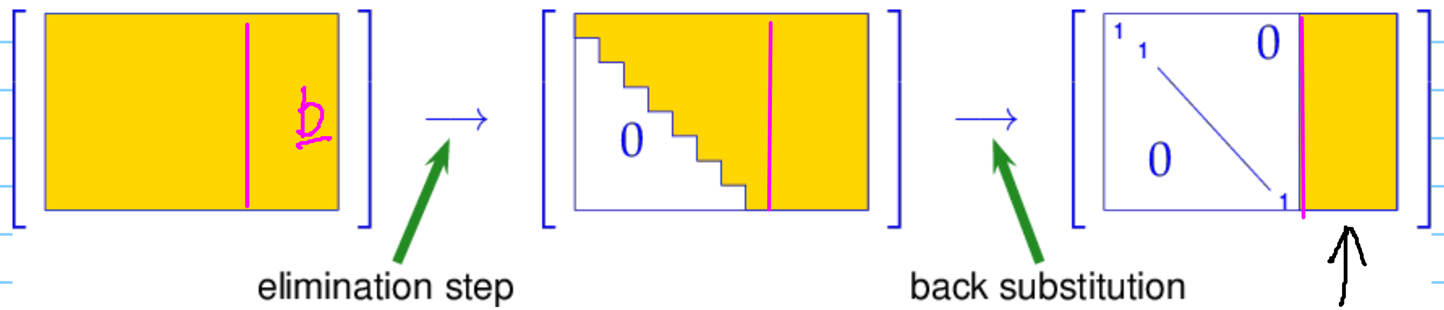
$\Leftrightarrow$ $\mathcal{N}(A) := \{\underline{x} : A\underline{x} = \underline{0}\} = \{\underline{0}\}$

2.3. Gaussian Elimination → L.A.

Basic algorithm:



→ row transformations & permutations



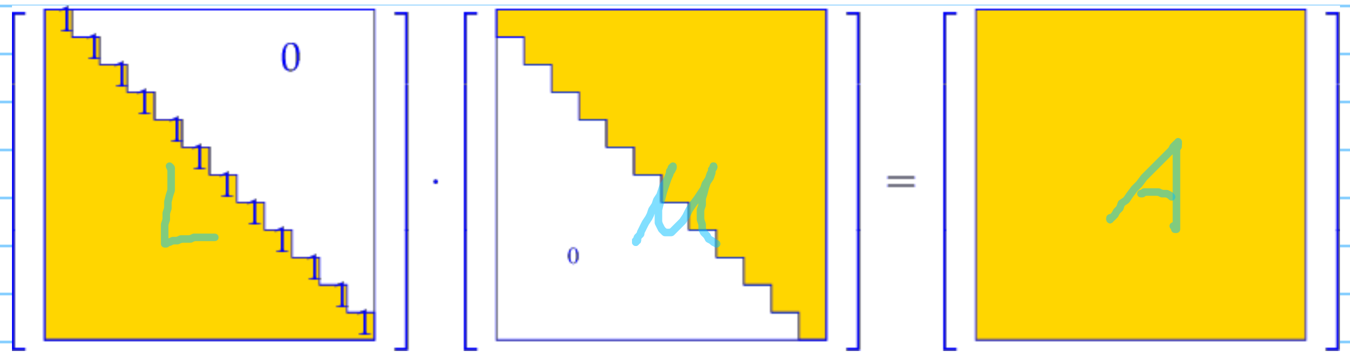
solution vector

Complexity: $\quad O(n^3) \qquad\qquad O(n^2)$

Alternative perspective : LU-decomposition

$$L \cdot U = PA$$

↳ row permutation



Solving LSE based on LU-decomposition :

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

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

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

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

$$\underline{A}\underline{x} = \underline{b} \Leftrightarrow L(U\underline{x}) = \underline{b} \Leftrightarrow \left.\begin{array}{l} L\underline{z} = \underline{b} \\ U\underline{x} = \underline{z} \end{array}\right\}$$

Eigen : $X = A.lu().solve(B)$

[$A \leftrightarrow n \times n$ matrix, $B \in \mathbb{R}^{n,\ell}$ $\Rightarrow$ $X = A^{-1}B$]

$$X = A^{-1}B = [A^{-1}(B)_{:,1}, \dots, A^{-1}(B)_{:,\ell}]$$

$\longrightarrow$ multiple r.h.s. : effort $O(n^3 + n^2\ell)$

Importance of LU-dec. :

C++11 code 2.5.11: Wasteful approach!

```
// 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);
}
```

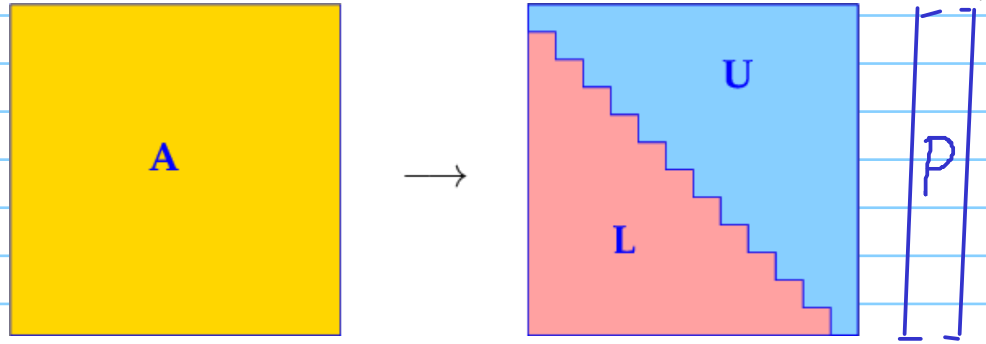
computational effort $O(Nn^3)$

C++11 code 2.5.12: Smart approach!

```
// Setting: N ≫ 1,
// large matrix A ∈ K^{n,n}
auto A_lu_dec = A.lu();          O(n^3)
for(int j = 0; j < N; ++j){
    x = A_lu_dec.solve(b);       O(n^2)
    b = some_function(x);
}
```

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

* Eigen internal in-situ LU by lu()



↑ VectorXi

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

If possible, use algorithms from numerical libraries! (→ Exp. 2.3.7)

③

1.6.5. Exploiting structure when solving LSE

Abstract: Block elimination

Recall: Block matrix multiply

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{11}\mathbf{B}_{11} + \mathbf{A}_{12}\mathbf{B}_{21} & \mathbf{A}_{11}\mathbf{B}_{12} + \mathbf{A}_{12}\mathbf{B}_{22} \\ \mathbf{A}_{21}\mathbf{B}_{11} + \mathbf{A}_{22}\mathbf{B}_{21} & \mathbf{A}_{21}\mathbf{B}_{12} + \mathbf{A}_{22}\mathbf{B}_{22} \end{bmatrix}. \quad (1.3.16)$$

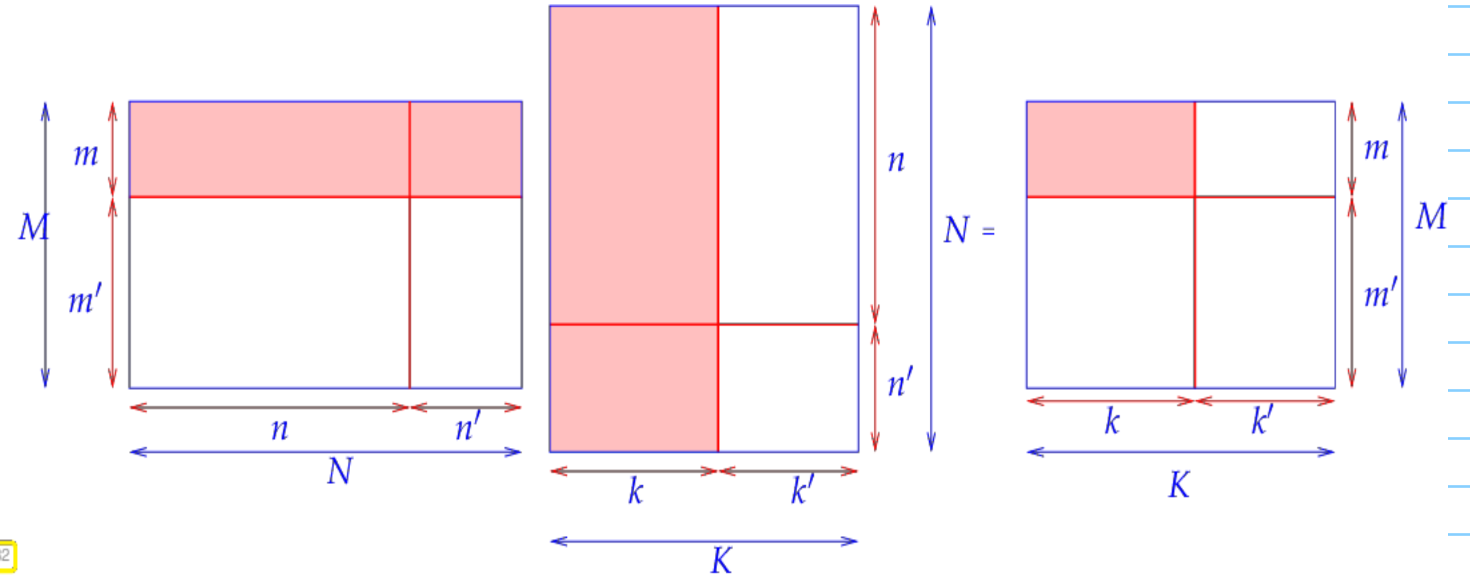


Fig. 32

BE: $\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix}$, $\mathbf{A}_{11} \in \mathbb{K}^{k,k}$, $\mathbf{A}_{12} \in \mathbb{K}^{k,\ell}$, $\mathbf{A}_{21} \in \mathbb{K}^{\ell,k}$, $\mathbf{A}_{22} \in \mathbb{K}^{\ell,\ell}$, $\mathbf{x}_1 \in \mathbb{K}^k$, $\mathbf{x}_2 \in \mathbb{K}^\ell$, $\mathbf{b}_1 \in \mathbb{K}^k$, $\mathbf{b}_2 \in \mathbb{K}^\ell$. (2.6.3)

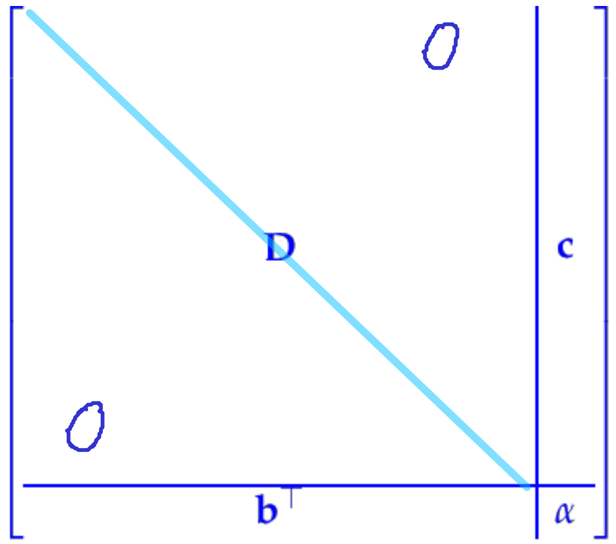
$$\underline{x}_1 = A_{11}^{-1}(b_1 - A_{12}\underline{x}_2)$$

$$\Rightarrow \underbrace{(A_{22} - A_{21}A_{11}^{-1}A_{12})}_{\text{Schur complement}}\underline{x}_2 = \underline{b}_2 - A_{21}A_{11}^{-1}\underline{b}_1$$

Useful, if A_{11} is easy to invert (e.g. diagonal)

Ex: Arrow matrix $(D \in \mathbb{R}^{n,n})$

$\mathbf{A} =$



(2.6.6)

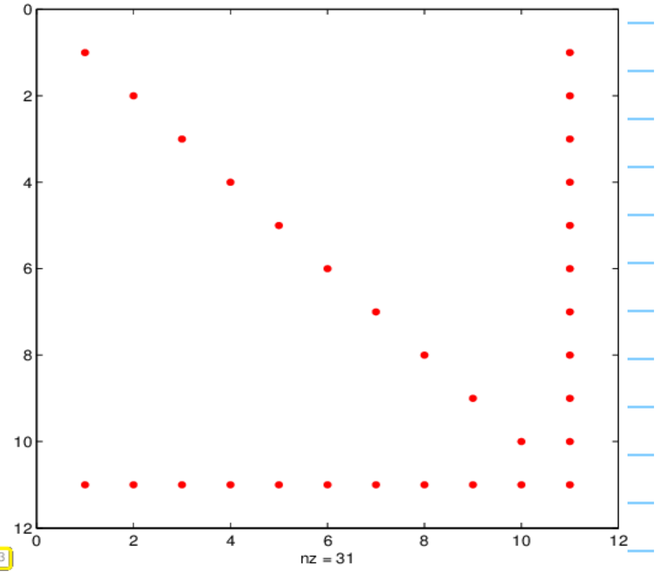


Fig. 63

$$A\underline{x} = \underline{b} \iff \begin{bmatrix} D & \underline{c} \\ \underline{b}^T & \alpha \end{bmatrix} \begin{bmatrix} \underline{x}_1 \\ \xi \end{bmatrix} = \begin{bmatrix} \underline{b}_1 \\ \beta \end{bmatrix}, \quad \underline{b} \in \mathbb{R}^n$$

C++11 code 2.6.9: Dense Gaussian elimination applied to arrow system

```
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); A.setZero();
  A.diagonal().head(n) = d;
  A.col(n).head(n) = c;
  A.row(n).head(n) = b;
  A(n, n) = alpha;
  return A.lu().solve(y);
}
```

Initialize arrow matrix

→ $O(n^3)$ full LU-factors

[$O(n^2)$ memory!]

(4)

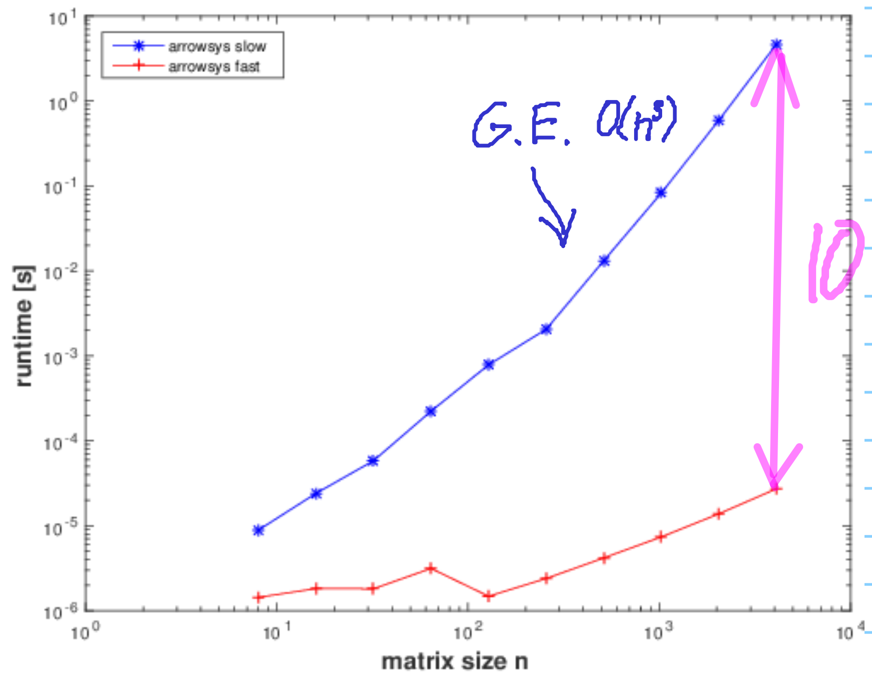
BE: $(\alpha - \underline{b}^T D^{-1} \underline{c})\,\xi = \beta - \underline{b}^T D^{-1} \underline{b}_1$

$$\underline{x}_1 = D^{-1}(\underline{b}_1 - \xi\, \underline{c})$$

C++11 code 2.6.10: Solving an arrow system according to (2.6.8)

```
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        O(n)
    VectorXd w = y.head(n).array() / d.array();       // w = D^{-1}y_1      O(n)
    double xi = (y(n) - b.dot(w)) / (alpha - b.dot(z));                     O(n)
    VectorXd x(n+1);
    x << w - xi*z, xi;        O(n)
    return x;
}
```

⇒ $O(n)$



▽ Possible instability of block elimination

Safe for

- s.p.d. LSE
  $A = A^T$, $\underline{x}^T A \underline{x} > 0\ \forall \underline{x} \neq 0$
- diagonally dominant LSE:
  $$|(A)_{ii}| \geq \sum_{j \neq i} |(A)_{ij}| \quad \forall i$$

Low-rank modification of an LSE

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

– Then solve $\tilde{A}x = \tilde{b}$: $A, \tilde{A}$ differ by one entry

$$\mathbf{A}, \tilde{\mathbf{A}} \in \mathbb{K}^{n,n}: \quad \tilde{a}_{ij} = \begin{cases} a_{ij} & \text{, if } (i,j) \neq (i^*, j^*)\,, \\ z + a_{ij} & \text{, if } (i,j) = (i^*, j^*)\,, \end{cases} \quad , \quad i^*, j^* \in \{1, \dots, n\}\,. \tag{2.6.14}$$

$$\blacktriangleright \quad \boxed{\tilde{\mathbf{A}} = \mathbf{A} + z \cdot \mathbf{e}_{i^*} \mathbf{e}_{j^*}^T}\,. \tag{2.6.15}$$

$$\begin{array}{cc} & \\ i^* \Rightarrow & \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & z & \vdots \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \\ & \uparrow j^* \end{array}$$

Example of a rank-1-modification

General: $\boxed{\tilde{A} = A + \underline{u}\underline{v}^T}$,

$\underline{u}, \underline{v} \in \mathbb{R}^n \setminus \{0\}$

Trick: Block elimination $\begin{bmatrix} A & \underline{u} \\ \underline{v}^T & -1 \end{bmatrix} \begin{bmatrix} \tilde{\underline{x}} \\ \xi \end{bmatrix} = \begin{bmatrix} \underline{b} \\ 0 \end{bmatrix}$

I: $[\ \xi = \underline{v}^T \tilde{\underline{x}} \Rightarrow A\tilde{\underline{x}} + \underline{u}\underline{v}^T \tilde{\underline{x}} = b$

$\Rightarrow (A + \underline{u}\underline{v}^T)\tilde{\underline{x}} = b$

II: $[\ \underline{v}^T A^{-1}(b - \underline{u}\xi) - \xi = 0 \Rightarrow \xi = \underline{v}^T A^{-1} \underline{b} / (1 + \underline{v}^T A^{-1} \underline{u})\ ]$

$$\tilde{\underline{x}} = A^{-1}(\underline{b} + \underline{u} \cdot \xi) = A^{-1}\left(b + \frac{\underline{u}\underline{v}^T A^{-1} \underline{b}}{(1 + \underline{v}^T A^{-1} \underline{u})}\right)$$

$$\tilde{\mathbf{x}} = \mathbf{A}^{-1}\mathbf{b} - \frac{\mathbf{A}^{-1}\mathbf{u}(\mathbf{v}^H(\mathbf{A}^{-1}\mathbf{b}))}{1+\mathbf{v}^H(\mathbf{A}^{-1}\mathbf{u})} \,. \qquad (2.6.23)$$

[Sherman-Morrison Woodbury]

costs $O(n^2)$, if LU-decomposition of A available [triangular solves]

C++11 code 2.6.24: Solving a rank-1 modified LSE

```cpp
// Solving rank-1 updated LSE based on (2.6.23)
template <class LUDec>
VectorXd smw(const LUDec &lu, const MatrixXd &u, const VectorXd &v,
    const VectorXd &b) {
  VectorXd z = lu.solve(b); //
  VectorXd w = lu.solve(u); //
  double alpha = 1.0 + v.dot(w);
  if (std::abs(alpha) < std::numeric_limits<double>::epsilon())
    throw std::runtime_error("A nearly singular");
  else return (z - w * v.dot(z) / alpha);
}
```


2.7. Sparse Linear Systems

↳ "most of the entries of $A = 0$"

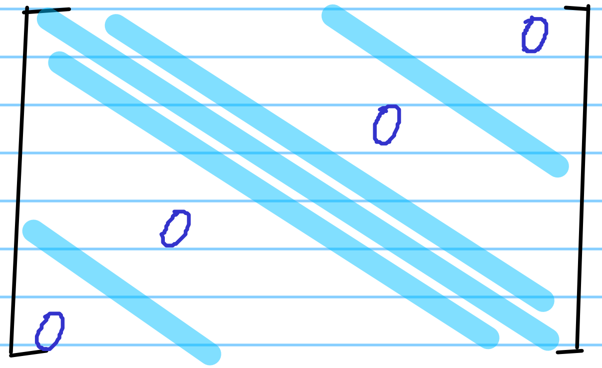
Notion 2.7.1. Sparse matrix

$\mathbf{A} \in \mathbb{K}^{m,n}$, $m, n \in \mathbb{N}$, is sparse, if

$$\mathrm{nnz}(\mathbf{A}) := \#\{(i,j) \in \{1,\dots,m\} \times \{1,\dots,n\}: a_{ij} \neq 0\} \text{ " } \ll \text{ " } mn \,.$$

Example: "Arrow matrix", diagonal matrix

banded matrix



2.7.1. Sparse matrix Storage Formats

Goal: req. memory $\sim$ nnz(A)

cost (matrix $\times$ vector) $\sim$ nnz(A)

Example: COO / triplet format

→ List of triplets $(i, j, (A)_{ij})$

```
struct TripletMatrix {
  size_t m,n;              // Number of rows and columns
  vector<size_t> I;        // row indices
  vector<size_t> J;        // column indices
  vector<scalar_t> a;      // values associated with index pairs
};
```

! Repetition of index pair is possible — values added up!

C++-code 2.7.7: Matrix×vector product ~~$y =$~~ $\mathbf{A}\mathbf{x}$ in triplet format $y = Ax + y$

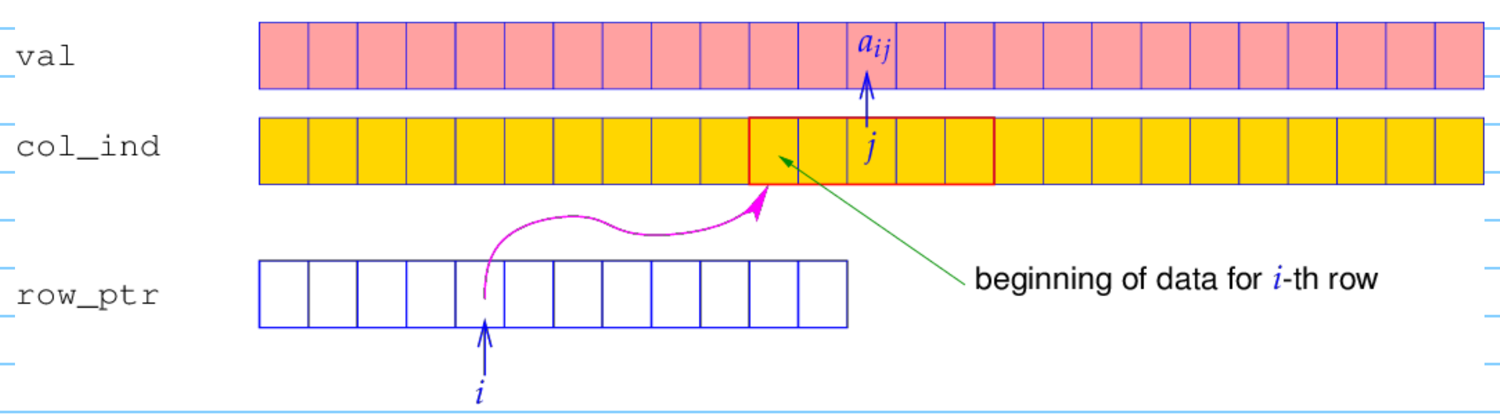
```
void multTriplMatvec(const TripletMatrix &A,
                     const vector<scalar_t> &x,
                     vector<scalar_t> &y)
 for (size_t l=0; l<A.a.size(); l++) {
   y[A.I[l]] += A.a[l]*x[A.J[l]];
 }
```

Example: Compressed row/column storage: CRS/CCS

$A \in \mathbb{R}^{n,n}$: CRS

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

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



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

`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

[index from 1]

Start of row #1

CRS: non-zero entries of rows in contiguous memory

2.7.3. Sparse Matrices in Eigen

```
#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

↳ Just setting entries may involve massive data movement

↳ Two-pass initialization from COO format

COO in Eigen:

```
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());
spMat.makeCompressed();
```

Experiment:

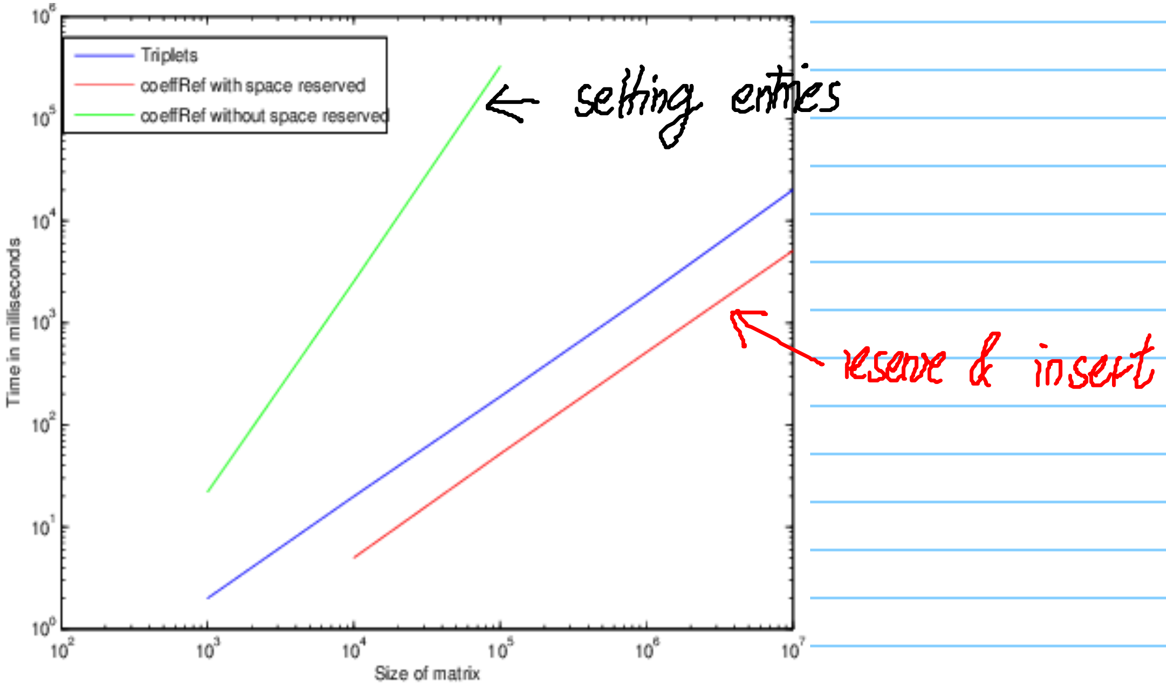


Fig. 75

Alternative: reserve() & insert()

C++11-code 2.7.21: 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));
// do many (incremental) initializations
for ( ) {
  mat.insert(i,j) = value_ij;
  mat.coeffRef(i,j) += increment_ij;
}
mat.makeCompressed();
```

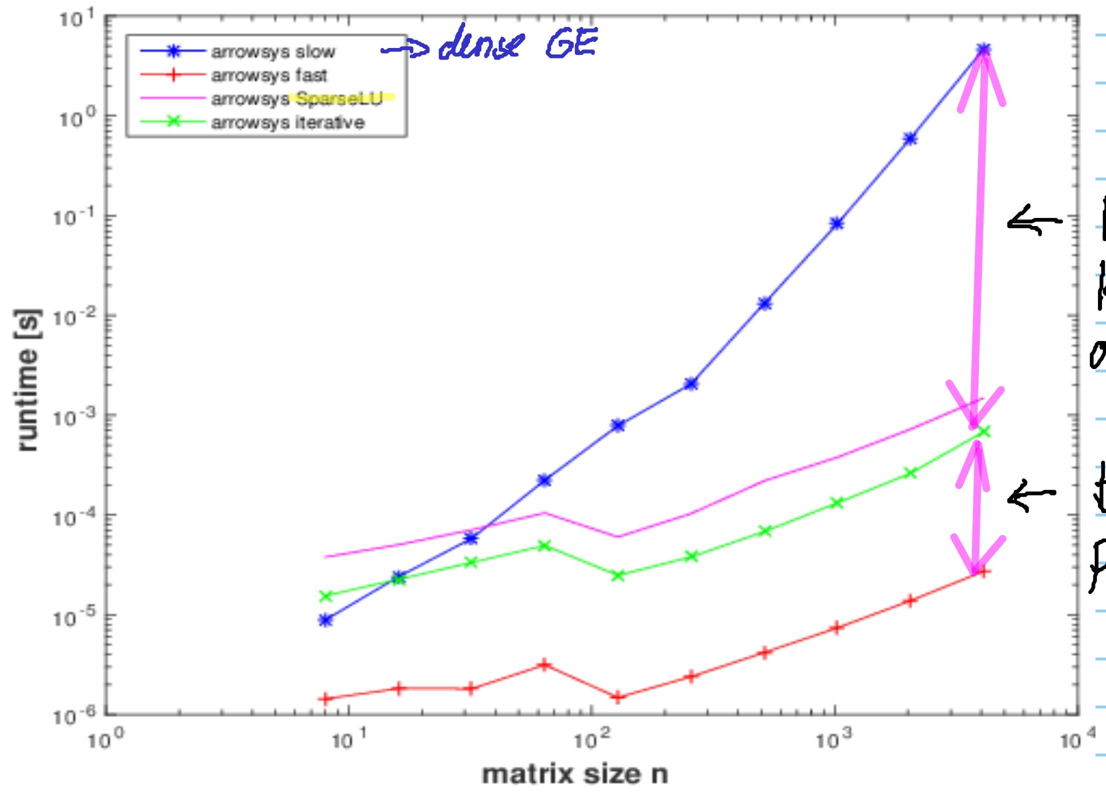
→ Create final CRS

1.7.4. Direct solution of sparse linear systems of Equation

Sparse matrix format ⇒ tells about location of zeros in matrix!

→ important for sparse elimination method

Example: Arrow matrix



← Exploit knowledge of pos. of zeros

← bookkeeping / pivoting overhead

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 (→ 2.7.1) to convey information about zero entries.

→ In practice: cost of sparse solves $\sim O(\text{nnz}(A)^{\alpha})$, $\alpha \approx 1.5 - 2.5$