Numerical Methods for Computational Science and Engineering

Fall Semester 2017 (HS17)

Prof. Rima Alaifari, SAM, ETH Zurich

Example: "Hidden summation"

$A, B \in \mathbb{K}^{n,p}$ $\qquad p \ll n$

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

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

upper triangular of AB^T

computing AB^T: $\quad \mathcal{O}(n^2 p)$

Different approach?

Special case $p=1$:

$$\underbrace{\begin{bmatrix} \\ \\ \\ \end{bmatrix}}_{a \in \mathbb{K}^n} \cdot \underbrace{\begin{bmatrix} \qquad\qquad \end{bmatrix}}_{b^T} \quad (b \in \mathbb{K}^n)$$

Computing $y = \mathrm{triu}(ab^T)x$: Decompose & regroup first

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

$$= \underbrace{\begin{bmatrix} a_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & a_n \end{bmatrix}}_{\text{multipl by diag: } \mathcal{O}(n)} \underbrace{\begin{bmatrix} 1 & 1 & \dots & & \dots & 1 \\ 0 & 1 & 1 & \dots & \dots & 1 \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & \vdots \\ 0 & \dots & & \dots & 0 & 1 \end{bmatrix}}_{\mathbf{T}} \underbrace{\begin{bmatrix} b_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{bmatrix}}_{=: z \qquad \mathcal{O}(n)} .$$

$$\mathbf{z} := \begin{bmatrix} b_1 x_1 \\ \vdots \\ b_n x_n \end{bmatrix}$$

Efficient multiplication $T\mathbf{z}$? $\quad \Theta(n^2)$ worst case

$$T\mathbf{z} = \begin{bmatrix} \sum_{i=1}^{n} z_i \\ \sum_{i=2}^{n} z_i \\ \vdots \\ z_{n-1} + z_n \\ z_n \end{bmatrix}$$

Can be implemented in $\Theta(n)$ using partial_sum (backwards computation from last entry)

$p > 1$?

$$AB^T = \sum_{\ell=1}^{p} \underbrace{(A)_{:,\ell} \cdot \left((B)_{:,\ell}\right)^T}_{\text{outer product of 2 vectors ("}p=1\text{" case)}}$$

$\Rightarrow$ overall complexity: $\Theta(pn)$!

C++11 code 1.4.16: Efficient multiplication with the upper diagonal part of a rank-p-matrix in EIGEN ➺ GITLAB

```cpp
//!  Computation of y = triu(AB^T)x
//!  Efficient implementation with backward cumulative sum
//!  (partial_sum)
template<class Vec, class Mat>
void lrtrimulteff(const Mat& A, const Mat& B, const Vec& x, Vec& y){
  const int n = A.rows(), p = A.cols();
  assert( n == B.rows() && p == B.cols()); // size missmatch
  for(int l = 0; l < p; ++l){
    Vec tmp = (B.col(l).array() * x.array()).matrix().reverse();
    std::partial_sum(tmp.data(), tmp.data()+n, tmp.data());
    y += (A.col(l).array() * tmp.reverse().array()).matrix();
  }
}
```

Example: Kronecker product $A \otimes B$

$A \in \mathbb{K}^{m,n}$, $B \in \mathbb{K}^{l,k}$

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

$(A \otimes B)x \qquad x \in \mathbb{K}^{n \cdot k}$

$A \otimes B \qquad \Theta(mlnk)$

Reshape $\quad x = \begin{bmatrix} x^1 \\ x^2 \\ \vdots \\ x^n \end{bmatrix} \qquad x^j \in \mathbb{K}^k$

$$X = \begin{bmatrix} x^1 & x^2 & \dots & x^n \end{bmatrix} \in \mathbb{K}^{k,n}$$

$$\underset{\in \mathbb{K}^{m \cdot l}}{(\mathbf{A} \otimes \mathbf{B})\mathbf{x}} = \begin{bmatrix} (\mathbf{A})_{1,1}\mathbf{B}\mathbf{x}^1 + (\mathbf{A})_{1,2}\overbrace{\mathbf{B}\mathbf{x}^2}^{\in \mathbb{K}^l} + \dots + (\mathbf{A})_{1,n}\mathbf{B}\mathbf{x}^n \\ (\mathbf{A})_{2,1}\mathbf{B}\mathbf{x}^1 + (\mathbf{A})_{2,2}\mathbf{B}\mathbf{x}^2 + \dots + (\mathbf{A})_{2,n}\mathbf{B}\mathbf{x}^n \\ \vdots \\ \vdots \\ (\mathbf{A})_{m,1}\mathbf{B}\mathbf{x}^1 + (\mathbf{A})_{m,2}\mathbf{B}\mathbf{x}^2 + \dots + (\mathbf{A})_{m,n}\mathbf{B}\mathbf{x}^n \end{bmatrix} \begin{matrix} \leftarrow \in \mathbb{K}^l \\ \\ m \text{ such} \\ \text{entries} \\ \\ \end{matrix}$$

$$\underbrace{BXA^T}_{} = \left[\sum_{j=1}^{n} Bx^j \cdot (A)_{1,j}, \dots, \sum_{j=1}^{n} Bx^j \cdot (A)_{m,j} \right]$$

$$BX = \begin{bmatrix} Bx^1 & \dots & Bx^n \end{bmatrix}$$

$$A^T = \begin{bmatrix} ((A)_{1,:})^T & \dots & ((A)_{m,:})^T \end{bmatrix}$$

reshaping BXA^T to a column vector: $(A \otimes B)x$

$B \in \mathbb{K}^{l,k}$, $X \in \mathbb{K}^{k,n}$, $A^T \in \mathbb{K}^{n,m}$

$Y = BX \qquad \Theta(lkn) \qquad Y \in \mathbb{K}^{l,n} \qquad YA^T: \Theta(lnm)$

$\Rightarrow$ overall complexity: $\Theta(lkn + lnm)$

[compared to $\Theta(lknm)$

Other examples: divide-and-conquer algorithms

Strassen's algorithm: for multiplication of 2 $n \times n$ matrices: $\Theta(n^{\log_2 7})$ ≈ 2.81

(compared to $\Theta(n^3)$)

1.5.4. Cancellation

Example: Roots of quadratic polynomial

$$p(z) = z^2 + \alpha z + \beta$$

$$p(z_{1/2}) = 0$$

$$z_{1/2} = -\frac{\alpha}{2} \pm \frac{1}{2}\sqrt{\alpha^2 - 4\beta} \qquad (*)$$

Scenario: z_1 large, z_2 small

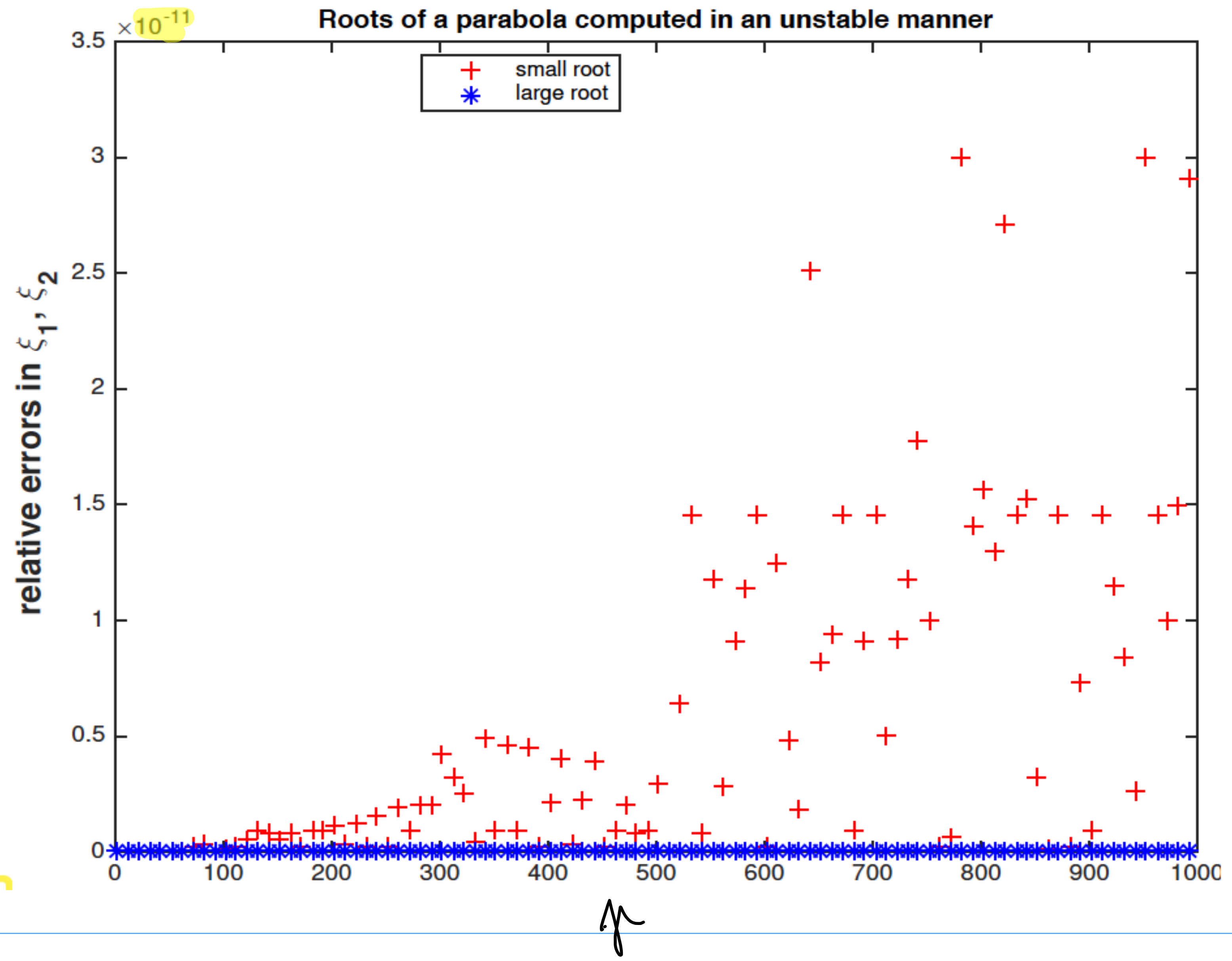
for example: $p(z) = (z - \mu)(z - 1/\mu)$

$$p(z) = z^2 - (\mu + 1/\mu)z + 1 \qquad (\alpha = \mu + 1/\mu, \; \beta = 1)$$

Use $(*)$ to compute the roots & compare the errors made in z_1, z_2

C++11-code 1.5.41: Discriminant formula for the real roots of $p(\xi) = \xi^2 + \alpha\xi + \beta$ ⇢ GITLAB

```
//!  C++ function computing the zeros of a quadratic polynomial
//!  ξ → ξ² + αξ + β by means of the familiar discriminant
//!  formula ξ_{1,2} = ½(−α ± √(α² − 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;
}
```



Large errors in ξ_2 compared to EPS!

$$\xi_2 = -\frac{\alpha}{2} + \frac{1}{2}\sqrt{\alpha^2-4}$$

α large

$\alpha \gg 1$

$\sqrt{\alpha^2-4} \approx \alpha$

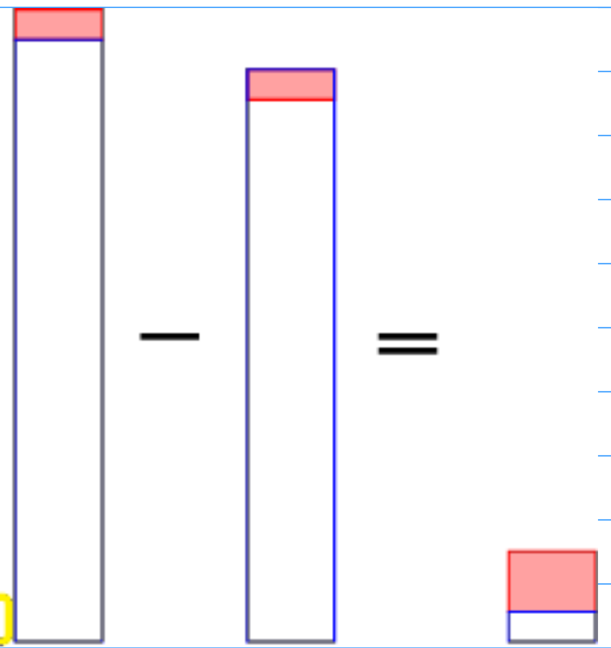


Fig. 41

errors get amplified as we subtract 2 numbers that are about the same size : cancellation

Stable computation of roots?

Step 1 : $\xi_1 = -\frac{\alpha}{2} - \frac{1}{2}\sqrt{\alpha^2-4\beta}$ (stable)

Idea: $\xi_2 = -\frac{\alpha}{2} + \frac{1}{2}\sqrt{\alpha^2-4\beta}$

$\xi_1 + \xi_2 = -\alpha$

$\xi_2 = -\alpha - \xi_1$

BUT $\alpha = \mu + \frac{1}{\mu}$ $\quad \mu = \xi_1$

$\xi_1 \approx \alpha$

Vieta's formula: $\beta = \xi_1 \cdot \xi_2$

Step 2: $\xi_2 = \beta / \xi_1$ (stable)

C++11-code 1.5.57: Stable computation of real root of a quadratic polynomial ⇢ GITLAB

```
//! C++ function computing the zeros of a quadratic polynomial
//! ξ → ξ² + αξ + β by means of the familiar discriminant
//! formula ξ_{1,2} = ½(−α ± √(α² − 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); //
      z << t, beta/t;
    }
    else{
      double t = 0.5*(-alpha+wD); //
      z << beta/t, t;
    }
  }
  return z;
}
```

Example: Difference quotients

$$f'(x) \approx \frac{f(x+h)-f(x)}{h}$$

approximation

Analysis: if f is suff. smooth e.g. $f \in C^2$

$$f'(x) = \frac{f(x+h)-f(x)}{h} + \mathcal{O}(h)$$

$$f'(x) = \lim_{h \to 0} \frac{f(x+h)-f(x)}{h}$$

(approximation error tends to 0 as $h \to 0$)

BUT: $f(x+h) - f(x)$: subtraction of 2 close-by numbers

$\rightarrow$ cancellation happens + amplified by $\frac{1}{h}$ (large)

Ex: $f(x) = e^x$

C++11-code 1.5.48: 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;
  }
}
```

$\log_{10}(h)$	relative error
-1	0.05170918075648
-2	0.00501670841679
-3	0.00050016670838
-4	0.00005000166714
-5	0.00000500000696
-6	0.00000049996218
-7	0.00000004943368
-8	-0.00000000607747
-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 ▷

We observe an initial decrease of the relative approximation error followed by a steep increase when h drops below 10^{-8}.

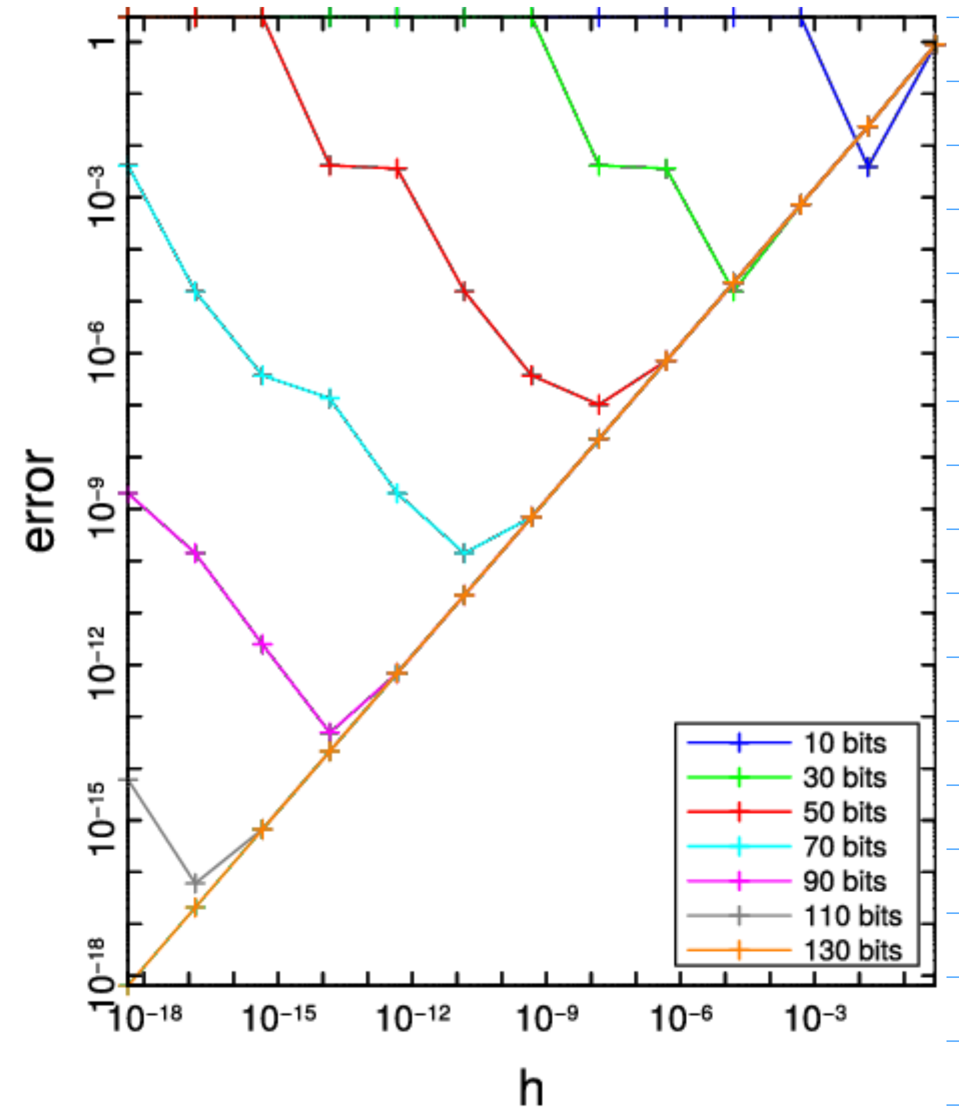


Fig. 42

Use the axiom of roundoff analysis (to compute choice of h):

difference quotient $dq = \dfrac{e^{x+h}(1+\delta_1) - e^x(1+\delta_2)}{h}$ $\qquad |\delta_1|, |\delta_2| \le EPS$

Choose h s.t. relative error $\left|\dfrac{e^x - dq}{e^x}\right|$ is minimized

$$dq = e^x\left(\frac{e^h-1}{h} + \frac{\delta_1 e^h - \delta_2}{h}\right)$$

$$\frac{dq - e^x}{e^x} = \overbrace{\frac{e^h-1}{h} - 1}^{\approx h/2} + \overbrace{\frac{\delta_1 e^h - \delta_2}{h}}^{|\;| \le \frac{2EPS}{h}}$$

Use $\dfrac{e^{x+h} - e^x}{h} - e^x = \dfrac{1}{2} h e^{\xi}$ for some $\xi \in [x, x+h]$

with $x=0$: $\dfrac{e^h - 1}{h} - 1 = \dfrac{1}{2} h e^{\xi}$ $\qquad \xi \in [0, h]$

$$\left|\frac{dq - e^x}{e^x}\right| \approx \frac{h}{2} + \frac{2\,EPS}{h} \quad \leftarrow \text{minimize}$$

$$\frac{1}{2} - \frac{2\,EPS}{h^2} \approx 0 \quad \Rightarrow \quad h \approx 2\sqrt{EPS}$$

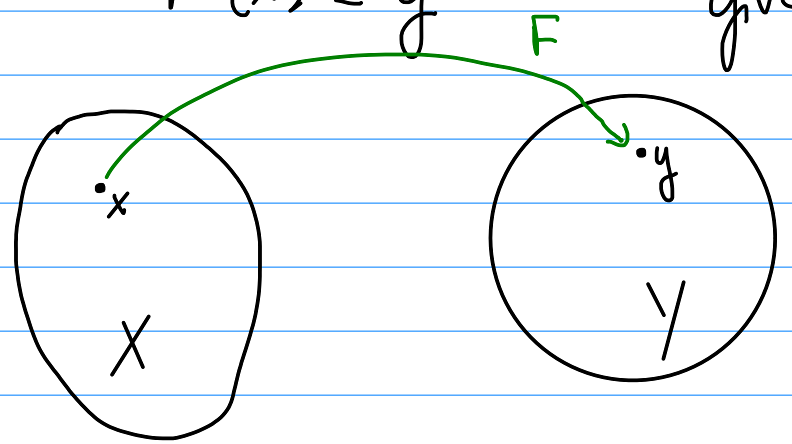
1.5.5. Numerical stability

Mathematical problem: data space X with $\|\cdot\|_X$

solution space Y with $\|\cdot\|_Y$

mapping $F: X \to Y$

$F(x) = y$ Given $x \in X$, find $y \in Y$



F is well-defined: for any $x \in X$ there exists a solution $y \in Y$

Problem	Algorithm
$F: X \to Y$	$\tilde{F}: X \to \tilde{Y} \subset M^m$

Example: Solving a linear system (LSE)

$$F: \begin{cases} X = \mathbb{R}^{n,n}_{reg} \times \mathbb{R}^n \longrightarrow Y = \mathbb{R}^n \\ (A, b) \longmapsto x \end{cases}$$

well-defined? Need to restrict to set of regular matrices $\mathbb{R}^{n,n}_{reg}$

Given problem F and an algorithm $\widetilde{F}$

$$\widetilde{F} : X \longrightarrow \widetilde{Y} \subseteq \mathbb{M}^m$$

When is $\widetilde{F}$ numerically stable?

Definition 1.5.85. Stable algorithm [backward stable]

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:

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

$$\frac{\|x - \widetilde{x}\|_X}{\|x\|_X} \leq C\, w(x)\, \mathrm{EPS}$$

$$\widetilde{F}(x) = F(\widetilde{x})$$

$\widetilde{F}$ is outputting the exact solution to nearly the right question

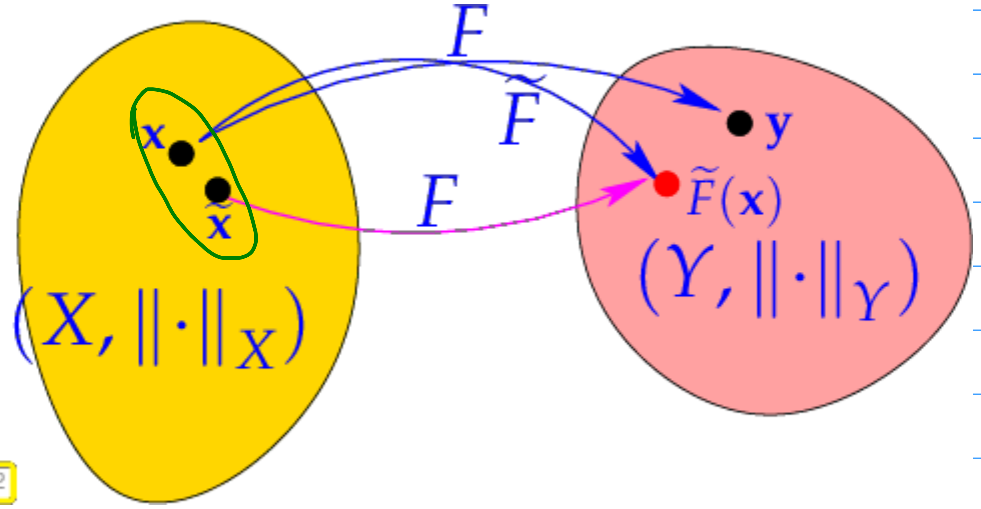


Fig. 52

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

More general: (mixed) stability

there is $\widetilde{x}$ with $\dfrac{\|x - \widetilde{x}\|}{\|x\|} \leq O(\mathrm{EPS})$ s.t.

$$\frac{\|\widetilde{F}(x) - F(\widetilde{x})\|}{\|F(\widetilde{x})\|} \leq O(\mathrm{EPS})$$

backward stability $\Rightarrow$ mixed stability

Q: How close is $\tilde{F}(x)$ to $F(x)$?

Recall toy example 3x3:

$$\frac{\|b-b^\delta\|_2}{\|b\|_2} \text{ small } \sim 10^{-4}$$

$$\frac{\|x-x^\delta\|_2}{\|x\|_2} \text{ large } \sim 10^{2}$$

Condition number of F:

$$F: X \to Y$$

$$c_F(x) = \sup_{\Delta x} \left(\frac{\|F(x+\Delta x)-F(x)\|}{\|F(x)\|} \Big/ \frac{\|\Delta x\|}{\|x\|} \right)$$

$\rightarrow$ sensitivity of the output to changes in the input

Condition number of a matrix

$$c_A(x) = \sup_{\Delta x} \left(\frac{\|A\Delta x\|}{\|Ax\|} \Big/ \frac{\|\Delta x\|}{\|x\|} \right)$$

$$c_A(x) = \sup_{\Delta x} \left(\frac{\|A\Delta x\|}{\|\Delta x\|} \cdot \frac{\|x\|}{\|Ax\|} \right)$$

$$= \|A\| \cdot \frac{\|x\|}{\|Ax\|} \qquad \|A\| := \sup_x \frac{\|Ax\|}{\|x\|} = \sup_{\|x\|=1} \|Ax\|$$

$$c_A := \|A\| \cdot \|A^{-1}\|$$

$$= \frac{\sigma_{max}}{\sigma_{min}}$$

if small: well-conditioned

if large: ill-cond.

$$\frac{\|\tilde{F}(x) - F(x)\|_y}{\|F(x)\|_y} \leq \mathcal{O}(c_F(x) \cdot EPS)$$

If the problem is well-conditioned:

backward stability guarantees accurate results

No guarantee when c_F is large

We can't expect $\frac{\|\tilde{F}(x) - F(x)\|}{\|F(x)\|} \leq \mathcal{O}(EPS)$

II. Direct Methods for Solving LSE

Solve an LSE $Ax = b$ $\quad \underbrace{A \in \mathbb{K}^{n,n}, b \in \mathbb{K}^n}_{\text{given}}$

solve for x

If A is regular / invertible:

existence & uniqueness of a solution x

A invertible: $\exists B \in \mathbb{K}^n$ s.t. $AB = I_n = BA$

$\Leftrightarrow$ $\operatorname{rank}(A) = n \Leftrightarrow \det(A) \neq 0$

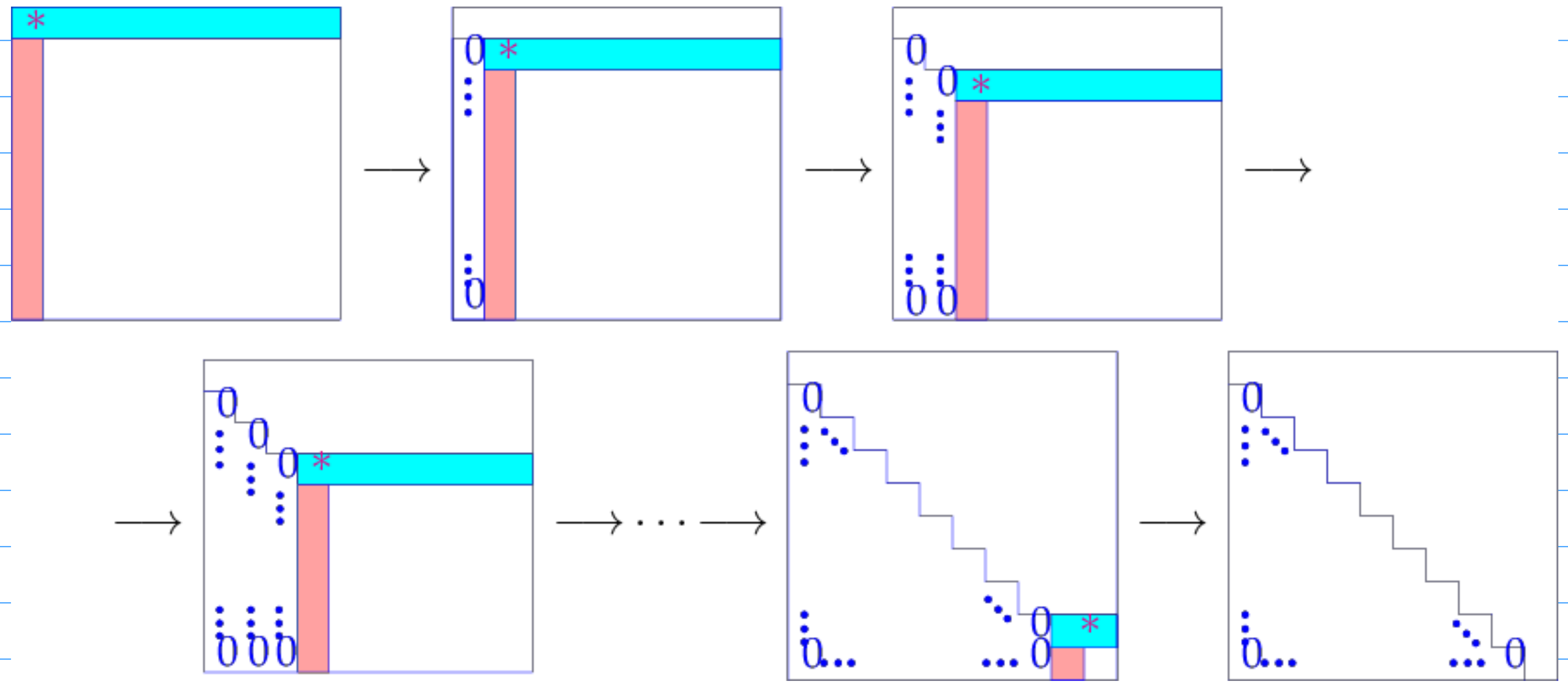
$\Leftrightarrow$ $\mathcal{N}(A) = \{x : Ax = 0\} = \{0\}$

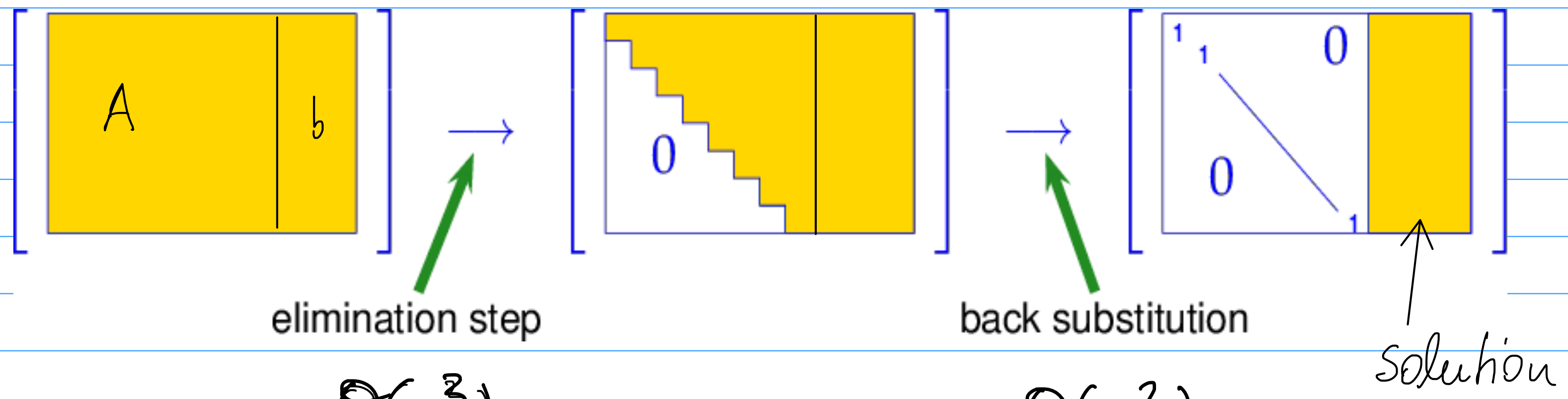
2.3 Gauss elimination

idea: $Ax = b$, $T \in \mathbb{K}^{n,n}$ reg.

$\Leftrightarrow \quad TAx = Tb$

$\hat{=}$ row transformations & permutations





$O(n^3)$ $\qquad\qquad$ $O(n^2)$

$\longrightarrow$ overall complexity: $O(n^3)$

[solving LSE triangular matrix: $O(n^2)$]

$\longrightarrow$ solving without computing A^{-1}

avoid computing A^{-1}

Alternative way: LU-decomposition

$$L \cdot U = P \cdot A$$

L: lower triangular

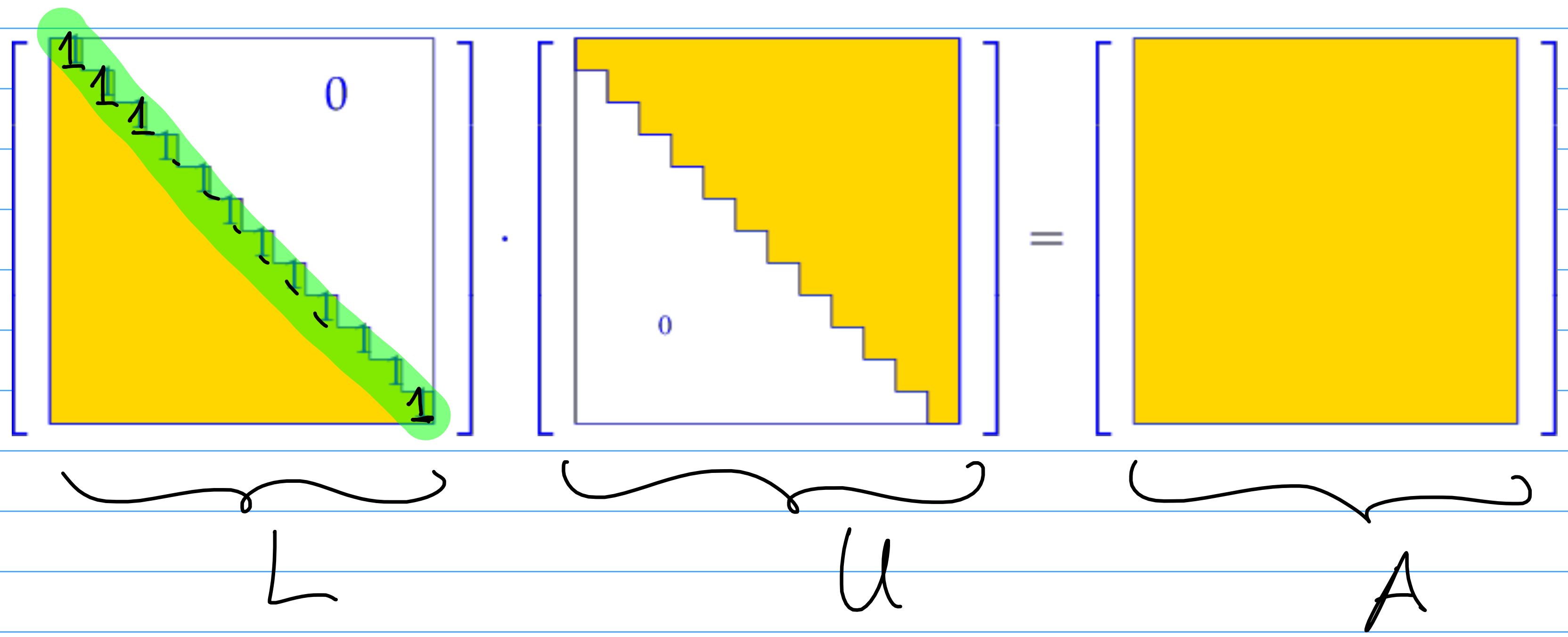
U: upper triangular

P: permutation of rows

$a_{11} = \ell_{11} u_{11}$

if $a_{11} = 0$

$\rightsquigarrow$ permutation needed



Solve for $Ax = B$ using LU decomposition:

$Ax = b$:

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

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

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

$\mathcal{O}(n^2)$

$$Ax = b$$

(1) $L\underbrace{Ux}_{z} = b$

(2) $Lz = b$ solve for z

(3) $Ux = z$ solve for x

Remark: Overall complexity is $\mathcal{O}(n^3)$ (same as Gauss elimination)

Q: What's the benefit?

Benefit if we solve for multiple RHS!

Suppose A $n \times n$, N different RHS

complexity $\Theta(n^3 + Nn^2) \Leftarrow$ LU decomp

Gauss elimination: $\Theta(Nn^3)$

C++11 code 2.5.11: Wasteful approach!
➜ GITLAB

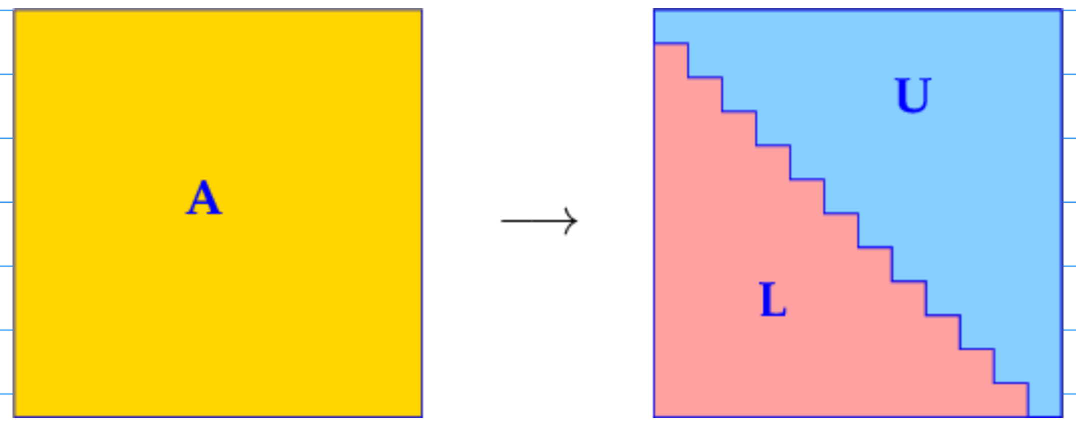
```
// Setting: N ≫ 1,
// large matrix A ∈ K^{n,n}
for(int j = 0; j < N; ++j){
    x = A.lu().solve(b);
    b = some_function(x);
}
```

computational effort $O(Nn^3)$

C++11 code 2.5.12: 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 $O(n^3 + Nn^2)$



(EIGEN)
↑
one matrix containing L & U [without storing diag(L)]

Use existing general solvers (for LSE) that are built-in

2.6 Exploiting structure when solving LSE

some entries vanish

matrix entries fulfill some formula

simple example: LSE with triangular matrix $\mathcal{O}(n^3) \longrightarrow \mathcal{O}(n^2)$.

2.6.2. Block elimination

Block matrix multiplication

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

Block elimination

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

$$A_{11} x_1 + A_{12} x_2 = b_1$$

$$x_1 = A_{11}^{-1} (b_1 - A_{12} x_2)$$

$$A_{21} x_1 + A_{22} x_2 = b_2$$

$$A_{21} A_{11}^{-1} (b_1 - A_{12} x_2) + A_{22} x_2 = b_2$$

$$\underbrace{(A_{22} - A_{21} A_{11}^{-1} A_{12})}_{\text{Schur complement}} x_2 = b_2 - A_{21} A_{11}^{-1} b_1$$

easy to solve if A_{11}^{-1} can be easily computed

Example: A_{11} is diagonal

$$\mathbf{A} = \left[\begin{array}{c|c} \mathbf{D} & \mathbf{c} \\ \hline \mathbf{b}^\top & \alpha \end{array}\right]$$

(2.

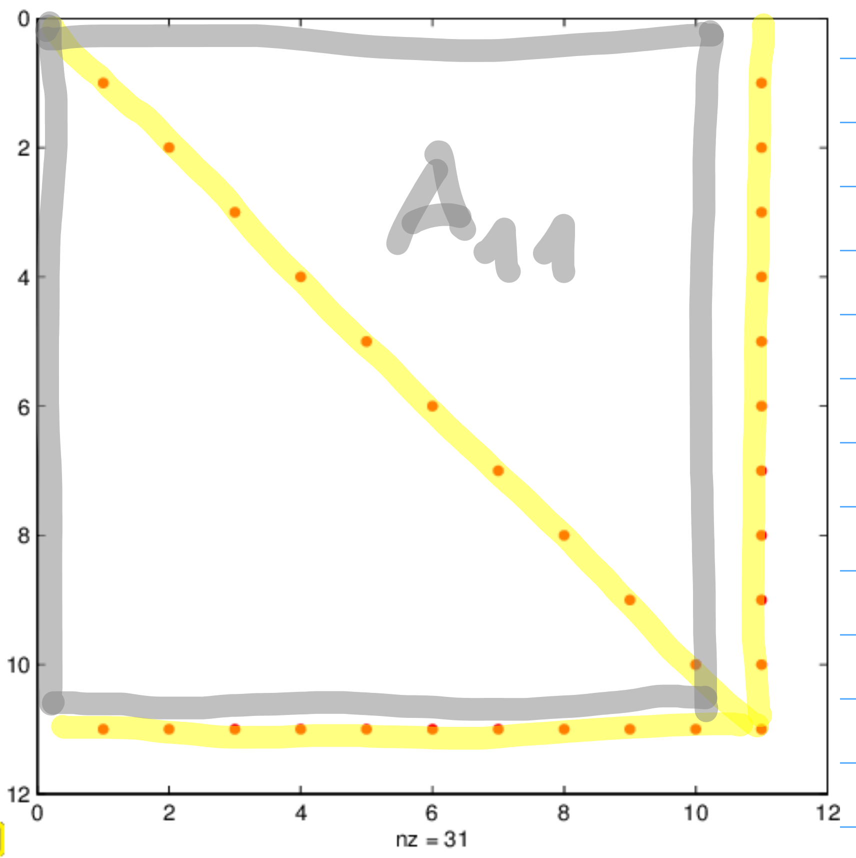


Fig. 65

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

$$\begin{bmatrix} D & c \\ b^T & \alpha \end{bmatrix}\begin{bmatrix} x_1 \\ \xi \end{bmatrix} = \begin{bmatrix} b_1 \\ \beta \end{bmatrix}$$

scalar   scalar

Solving with LU: $\underline{\Theta(n^3)}$

$$D x_1 + c\xi = b_1$$

(I) $\quad x_1 = D^{-1}(b_1 - c\xi)$

$$b^T x_1 + \alpha\xi = \beta$$

$$b^T D^{-1}(b_1 - c\xi) + \alpha\xi = \beta$$

(II) $\quad (\alpha - b^T D^{-1} c)\,\xi = \beta - b^T D^{-1} b_1$

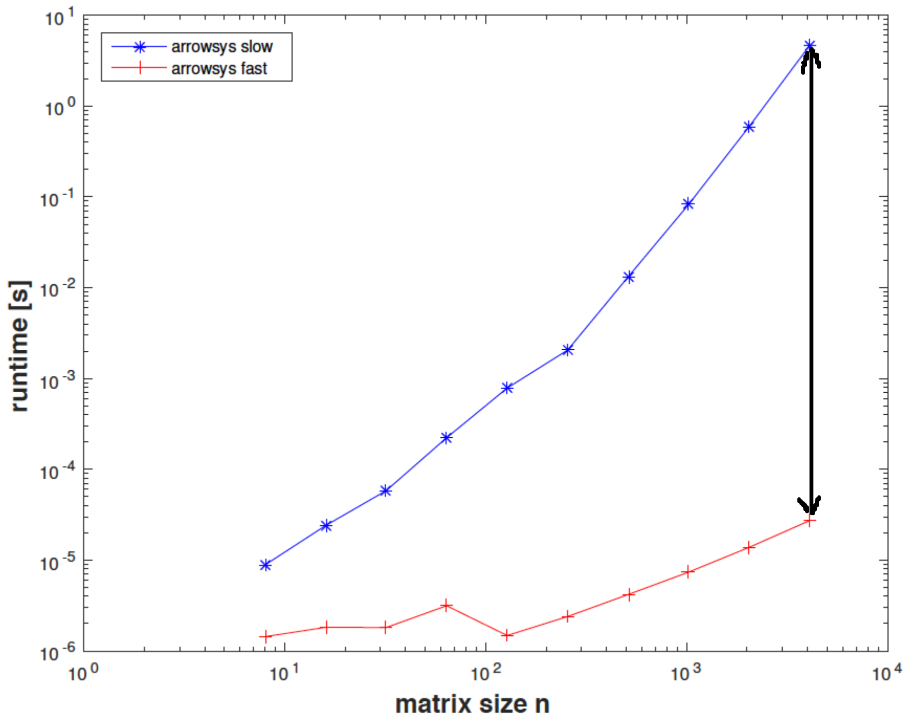
$z := D^{-1}c$: $\Theta(n)$

$w := D^{-1}b_1$: $\Theta(n)$

$$\left.\begin{array}{ll} \text{computing } \xi: & \Theta(n) \\ \text{computing } x: & \Theta(n) \end{array}\right\} \text{ overall: } \underline{\Theta(n)}$$

C++11 code 2.6.10: Solving an arrow system according to (2.6.8) ➜ **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}b_1
  double xi = (y(n) - b.dot(w)) / (alpha - b.dot(z));
  VectorXd x(n+1);
  x << w - xi*z, xi;
  return x;
}
```



$\sim 10^6$

warning: Block elimination can suffer from numerical instability [equivalent to Gauss elimination without pivoting]

roughly:
- stable for s.p.d. matrices
- stable for diagonally dominant matrices

for all $i \in \{1,\dots,n\}$: $|(A)_{i,i}| \geq \sum_{j \neq i} |(A)_{i,j}|$

Example: Low-rank modifications of LSE