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 2015

URL: http://www.sam.math.ethz.ch/~hiptmair/tmp/NumCSE/NumCSE15.pdf

I. Computing with Matrices and Vectors [Part II]

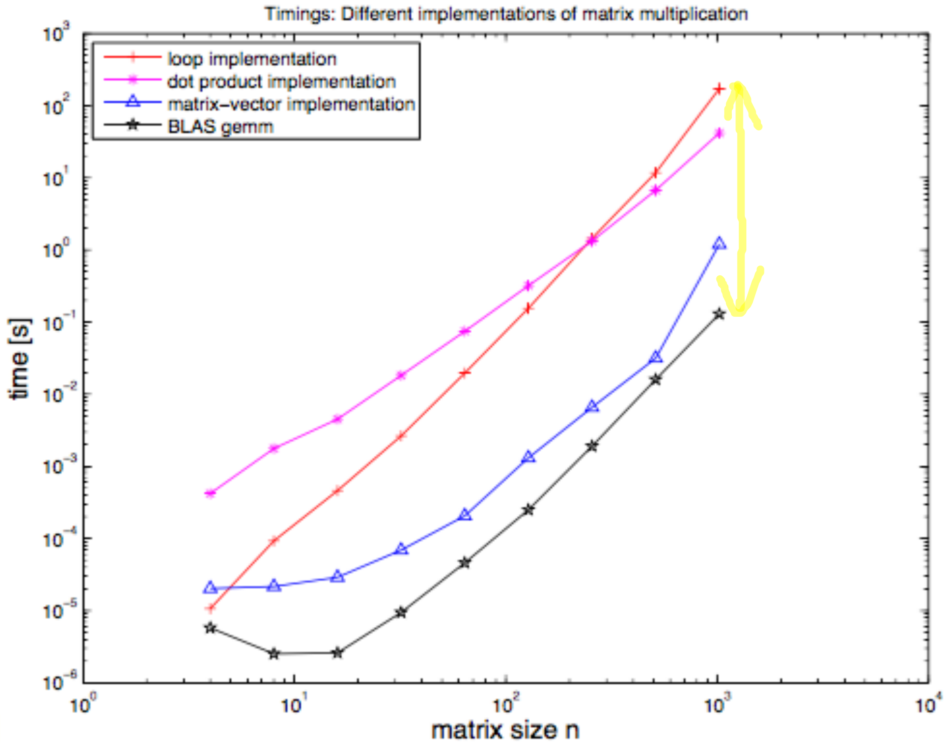
1.3. Basic linear algebra operations

'+', '-', '*', '/',

↳ matrix product

1.3.2. BLAS

Exp.: 1.3.15 : Matrix multiplication in Matlab



— ≙ Nested for loops

— ≙ Matlab *

factor 10^3 !

BLAS = highly optimized low level LA functions

- matrix×vector multiplication $\mathbf{y} = \alpha\mathbf{A}\mathbf{x} + \beta\mathbf{y}$

`xGEMV(TRANS,M,N,ALPHA,A,LDA,X,`

`INCX,BETA,Y,INCY)`

- $\mathtt{x} \in \{S,D,C,Z\}$, scalar type: S $\hat{=}$ type `float`, D $\hat{=}$ type `double`, C $\hat{=}$ type `complex`
- M, N $\hat{=}$ size of matrix $\mathbf{A}$
- `ALPHA` $\hat{=}$ scalar parameter α
- `A` $\hat{=}$ matrix $\mathbf{A}$ stored in *linear array* of length $M \cdot N$ (column major arrangement)

$$(\mathbf{A})_{i,j} = \mathtt{A}[N * (j-1) + i] \,.$$

- `LDA` $\hat{=}$ "leading dimension" of $\mathbf{A} \in \mathbb{K}^{n,m}$, that is, the number n of rows.
- X $\hat{=}$ vector $\mathbf{x}$: array of type `x`
- `INCX` $\hat{=}$ stride for traversing vector X
- `BETA` $\hat{=}$ scalar paramter β
- Y $\hat{=}$ vector $\mathbf{y}$: array of type `x`
- `INCY` $\hat{=}$ stride for traversing vector Y

1.4. Computational effort

Definition 1.4.1. Computational effort

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

Exp 1.8.15 : $\Rightarrow$

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

1.4.1. Asymptotic complexity

concrete function

Definition 1.4.3. (Asymptotic) complexity

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

vector length, matrix size

Notation: Landau - O :

Complexity $O(n^p)$: comp. eff. $\leq C n^p$

Tacit assumption : comp. eff $\not\leq C n^q$ for any $q < p$

Complexity $\not\Rightarrow$ runtime

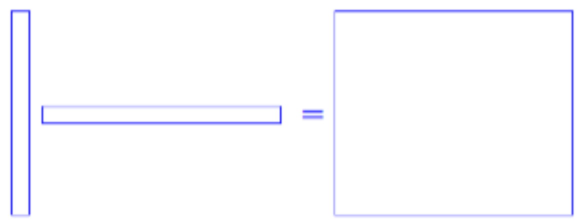
$\Rightarrow$ crude prediction of the dependence of the runtime of a code on n for large n

e.g. quadratic complexity $O(n^2)$: $n \to 2n$ will take ~ 4x time

1.4.2. Cost of basic operation

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	$n-1$	$O(n)$	opt
tensor product	$(\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n) \mapsto \mathbf{x}\mathbf{y}^H$	nm	0	$O(mn)$	opt
matrix product(*)	$(\mathbf{A} \in \mathbb{R}^{m,n}, \mathbf{B} \in \mathbb{R}^{n,k}) \mapsto \mathbf{AB}$	mnk	$mk(n-1)$	$O(mnk)$	not opt.

↳ nested loop implementation



tensor product

1.4.3 Tricks

Ex. 1.4.10: Exploit associativity : multiplication with tensor product

$b, a \in \mathbb{R}^n$, $x \in \mathbb{R}^n$

$$\mathbf{y} = \left(\mathbf{a}\mathbf{b}^\top\right)\mathbf{x}\,. \qquad (1.4.11)$$

```
T = a*b'; y = T*x;
```

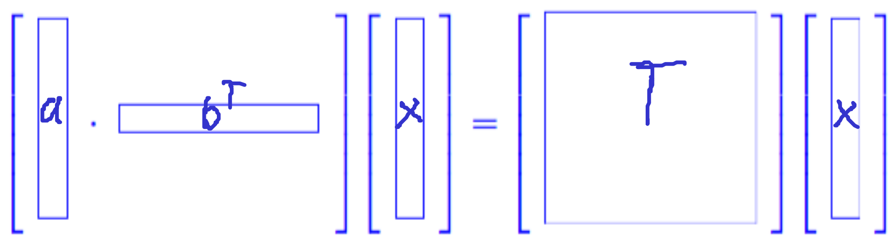
$$\mathbf{y} = \mathbf{a}\left(\mathbf{b}^\top\mathbf{x}\right)\,. \qquad (1.4.12)$$

```
t = b'*x; y = a*t;
```

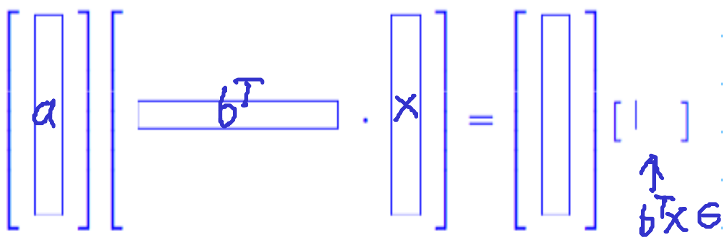
Complexity: $O(n^2)$ $O(n)$

y = a*b'*x ;

(1.4.11) :



(1.4.12) :



$b^T x \in \mathbb{R}$ (a number)

Ex. 1.4.14 (Hidden summation)

```
function y = lrtrimult(A,B,x)
y = triu(A*B')*x;
```

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

[Model case $p = 1$]

$p = 1$:

$$\mathbf{y} = \operatorname{triu}(\mathbf{a}\mathbf{b}^T)\mathbf{x} = \begin{array}{l} \cdot\, a_1 \rightarrow \\ \\ \\ \\ \cdot\, a_n \rightarrow \end{array} \begin{bmatrix} a_1b_1 & a_1b_2 & \dots & \dots & a_1b_n \\ 0 & a_2b_2 & a_2b_3 & \dots & a_2b_n \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \vdots \\ 0 & \dots & \dots & 0 & a_nb_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{bmatrix}$$

$\uparrow b_1$ $\uparrow \cdot b_n$

$$y = \begin{bmatrix} a_1 & & \\ & \ddots & \\ & & a_n \end{bmatrix} \underbrace{\begin{bmatrix} 1 & 1 & \cdots & & \cdots & 1 \\ 0 & 1 & 1 & \cdots & \cdots & 1 \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & \vdots \\ 0 & \cdots & & \cdots & 0 & 1 \end{bmatrix}}_{T} \begin{bmatrix} b_1 & & \\ & \ddots & \\ & & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{bmatrix}$$

Multiplication with T ? $\rightarrow$ cumulative summation

$p > 1$: $$AB^T = \sum_{\ell=1}^{p} (A)_{:,\ell} (B)_{:,\ell}^T$$

$$\mathrm{triu}(AB^T)x = \sum_{\ell=1}^{p} \underbrace{(A)_{:,\ell} (B)_{:,\ell}^T x}_{p=1 \text{ case}}$$

```
function y = lrtrimulteff(A,B,x)
[n,p] = size(A);
if (size(B) ~= [n,p]), error('size mismatch'); end
y = zeros(n,1);
for l=1:p, y = y + A(:,l).*cumsum(B(:,l).*x,'reverse'); end
```

Complexity $O(np)$

Testing $(x == 0)$ for a result x of a floating point computing is numerical crime :

test, if $\|x\| \approx 0$ in relative sense

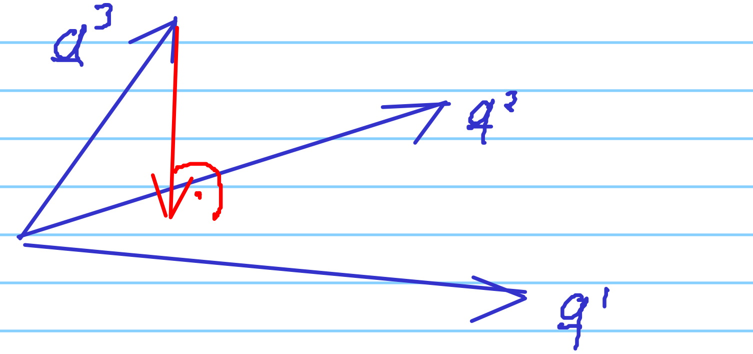
1.5. Machine arithmetic

1.5.1. Experiment: loss of orthogonality

Gram - Schmidt orthogonalization

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

1: $\mathbf{q}^1 := \frac{\mathbf{a}^1}{\|\mathbf{a}^1\|}$ % 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**
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** { $\mathbf{q}^j \leftarrow \frac{\mathbf{q}^j}{\|\mathbf{q}^j\|}$ }
}



Output : If not STOP, $\{q^1, \dots, q^k\}$ orthonormal

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

MATLAB-code 1.5.3: Gram-Schmidt orthogonalisation in MATLAB

```
function Q = gramschmidt(A)
% Gram-Schmidt orthogonalization of column vectors
% Arguments: Matrix A passes vectors in its columns
% Return values: Matrix Q contains the orthornormal basis in its
  columns
[n,k] = size(A); % Get number k of vectors and dimension n of space
Q = A(:,1)/norm(A(:,1)); % First basis vector
for j=2:k
  q = A(:,j) - Q*(Q'*A(:,j)); % Orthogonal projection; loop-free
    implementation
  nq = norm(q);                 % Check premature termination
  if (nq < (1E-9)*norm(A(:,j))), break; end % Safe check for == 0
  Q = [Q,q/nq]; % Add new basis vector as another column of Q
end
```

Exp: $(A)_{i,j} = \frac{1}{i+j-1}$; $1 \le i,j \le n$

```matlab
1  % MATLAB script demonstrating the effect of roundoff on the result of
     Gram-Schmidt orthogonalization
2  format short; % Print only a few digits in outputs
3  % Create special matrix the so-called Hilbert matrix:  (A)_{i,j} = (i+j-1)^{-1}
4  A = hilb(10);          % 10x10 Hilbert matrix
5  Q = gramschmidt(A); % Gram-Schmidt orthogonalization of columns of A
6  % Test orthonormality of column of Q, which should be an orthogonal
7  % matrix according to theory
8  I = Q'*Q,   % Should be the unit matrix, but isn't !
9
10 % MATLAB's internal Gram-Schmidt orthogonalization by QR-decomposition
11 [Q1,R1] = qr(A);
12 D = A - Q1*R1,   % Check whether we get the expected result
13 I1 = Q1'*Q1,     % Test orthonormality
```

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

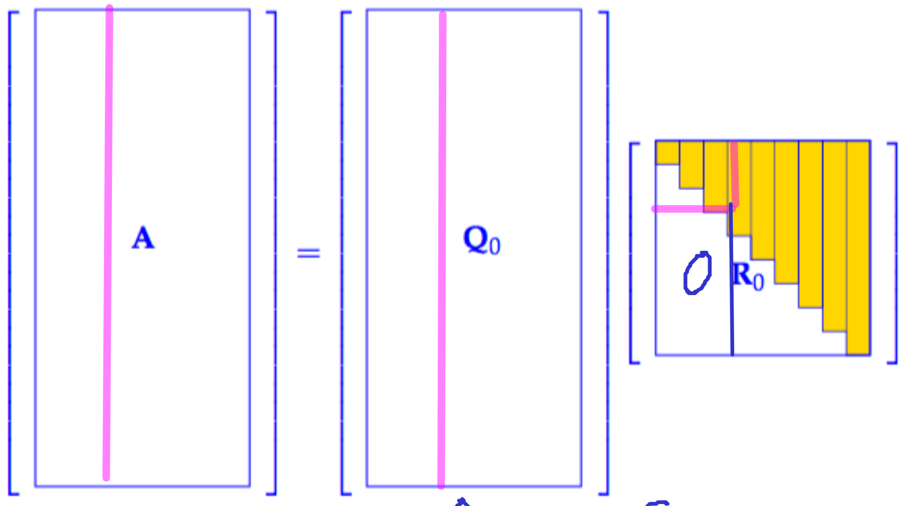
My computer cannot compute

Line 11: QR-decomposition $A \in \mathbb{R}^{n,k}$ there is $(k \le n)$

$Q \in \mathbb{R}^{n,k}$, $R \in \mathbb{R}^{k,k}$: $A = QR$

$Q^T Q = I_k$ orthonormal col.

R upper triangular



$, \mathbf{A} = \mathbf{Q}_0\mathbf{R}_0, \quad \mathbf{Q}_0 \in \mathbb{K}^{n,k}, \quad \mathbf{R}_0 \in \mathbb{K}^{k,k},$

$Q_0^T Q_0 = I_k$

$\Rightarrow$ QR-dec. $\stackrel{\wedge}{=}$ G.-S.

```
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   -0.0000   -0.0000   -0.0000    0.0000         0   -0.0000         0
    0.0000         0    1.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000         0
    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
    0.0000   -0.0000    0.0000    0.0000    0.0000    1.0000   -0.0000   -0.0000    0.0000         0
    0.0000    0.0000    0.0000   -0.0000    0.0000   -0.0000    1.0000    0.0000    0.0000   -0.0000
   -0.0000         0    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         0   -0.0000         0         0   -0.0000   -0.0000    0.0000    1.0000
```

LA	Matlab
QR $\stackrel{\wedge}{=}$ G.-S.	QR $\not\stackrel{\wedge}{=}$ G.-S.
	"good" "bad"

```
D = 1.0e-15 *
    0.2220    0.4441    0.3331    0.3053    0.2220    0.1388    0.1665    0.1249    0.1110    0.1388
         0    0.0555    0.0555         0    0.0278         0   -0.0278   -0.0139         0    0.0139
   -0.0555    0.0555         0         0         0         0         0   -0.0139    0.0278         0
         0    0.0278    0.0278         0         0         0   -0.0278         0         0         0
         0    0.0278    0.0278    0.0139         0    0.0278         0         0    0.0139    0.0278
   -0.0278    0.0278    0.0278         0    0.0139    0.0139    0.0139   -0.0139    0.0139    0.0139
         0    0.0139         0         0         0         0   -0.0139    0.0139         0         0
         0    0.0278    0.0139         0   -0.0139   -0.0139   -0.0139    0.0278    0.0278    0.0069
    0.0139    0.0278    0.0278    0.0139    0.0139         0         0         0         0    0.0139
         0    0.0278         0   -0.0139   -0.0139    0.0139         0    0.0069   -0.0069         0
```

1.5.2. Machine numbers (= floating point numbers)

Of course: computers can handle only finitely many numbers

machine numbers $\mathbb{M} \subset \mathbb{R}$
(finite, discrete subset)

$$op : \mathbb{M} \times \mathbb{M} \longrightarrow \mathbb{R} \qquad \not\longrightarrow \mathbb{M} \qquad op \in \{+,-,*,/\}$$

replace with $\downarrow$

$$\widetilde{op} : \mathbb{M} \times \mathbb{M} \longrightarrow \mathbb{M} ,$$

Implementation : $\widetilde{op} = \mathrm{rd} \circ op$

Definition 1.5.23. Correct rounding

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

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

! $\tilde{*}, \tilde{+}$ not associative

▷ Roundoff errors

1.5.3. Roundoff errors

Maximal relative roundoff error :

$$\mathrm{EPS} := \max_{x \in \mathbb{R} \setminus \{0\}} \frac{|x - \mathrm{rd}(x)|}{|x|} \quad : \text{machine precision}$$

```
>> format hex; eps, format long; eps
ans = 3cb0000000000000
ans = 2.220446049250313e-16
```

Assumption 1.5.28. "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 $|\delta| < \mathrm{EPS}$.

relative

1.5.4. Cancellation

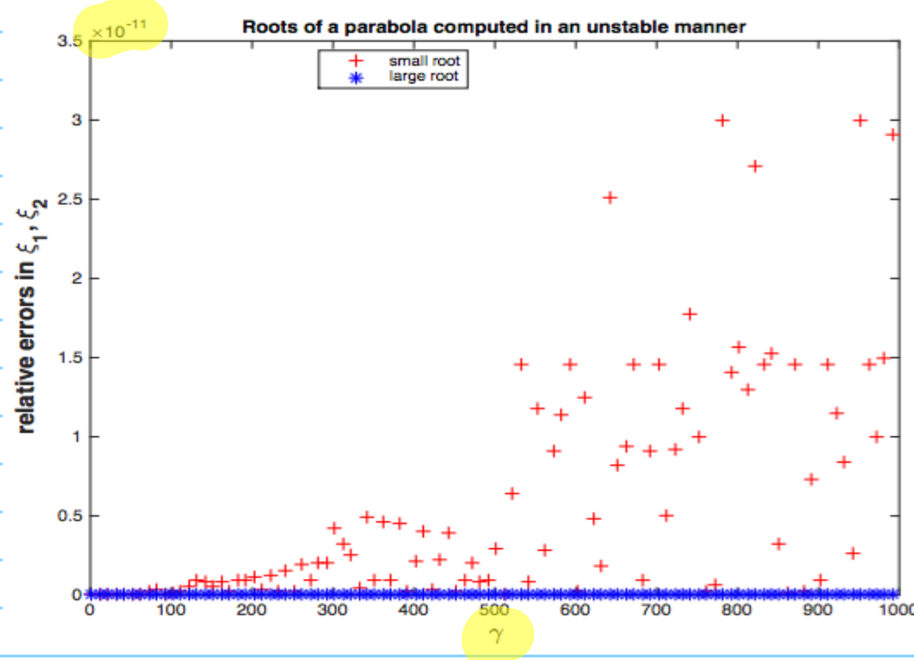
Ex 1.5.34: (Roots of a quadratic polynomial)

MATLAB-code 1.5.36: Discriminant formula for the real roots of $p(\xi) = \xi^2 + \alpha\xi + \beta$

```
function z = zerosquadpol(alpha,beta)
% MATLAB function computing the zeros of a quadratic polynomial
% ξ → ξ^2 + αξ + β by means of the familiar discriminant
% formula ξ_{1,2} = 1/2(−α ± sqrt(α^2 − 4β)). However
% this implementation is vulnerable to round-off! The zeros are
% returned in a column vector
D = alpha^2-4*beta; % discriminant
if (D < 0), z = []; % No real zeros
else
    % The famous discriminant formula
    wD = sqrt(D);
    z = 0.5*[-alpha-wD;-alpha+wD];
end
```

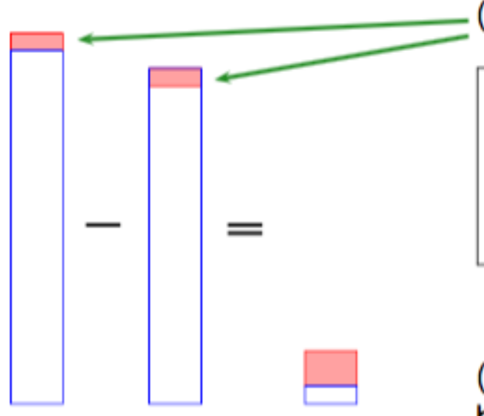
$$p(\xi) = (\xi - \gamma)(\xi - 1/\gamma) = \xi^2 - (\gamma + 1/\gamma)\xi + 1$$

computed before calling zerosquadpol()



Rather big rel. error in the small root $1/\gamma$ for $\gamma \gg 1$

Cause:



Cancellation

$\hat{=}$ Subtraction of almost equal numbers

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

($\lhd$ Roundoff error introduced by subtraction itself is negligible.)

```
D = alpha^2-4*beta; % discriminant
if (D < 0), z = []; % No real zeros
else
    % The famous discriminant formula
    wD = sqrt(D);
    z = 0.5*[-alpha-wD;-alpha+wD];
end
```

cancellation here

γ big $\Rightarrow$ $-\alpha$ big, D big

Ex 1.5.40: Cancellation & difference quotient

Approximation of derivative of a smooth function by difference quotient

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \quad \text{for } h \ll 1$$

cancellation expected here

Approximation error $O(h)$ for $h \to 0$

$$\left[\; f(x+h) = f(x) + hf'(x) + \tfrac{1}{2}h^2 f''(\xi),\quad x \le \xi \le x+h \;\right]$$

Ex: $f(x) = e^x$, $x = 0$, $f'(0) = 1$

```
h = 0.1; x = 0.0;
for i = 1:16
   df = (exp(x+h)-exp(x))/h;
   fprintf('%d %16.14f\n',i,df-1);

   h = h*0.1;
end
```

red $\to$ $\hat{=}$ correct digits

$\log_{10}(h)$	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

Roundoff error analysis

$$df = \frac{e^{x+h}(1+\delta_1) - e^x(1+\delta_2)}{h} \qquad |\delta_1|, |\delta_2| \le eps$$

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

Taylor formula $\Rightarrow$ $1 + O(h)$

Approximation error

Cancellation error $O(h^{-1})$

$\delta^*(h)$ becomes minimal for $h \approx \sqrt{eps}$

Avoiding cancellation:

Ex 1.5.43: Stable discriminant formula

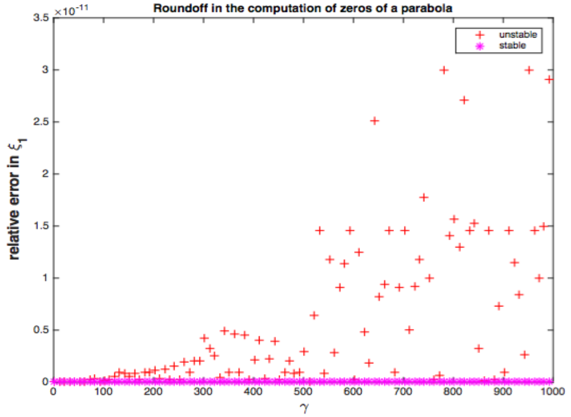
$$p(\xi) = \xi^2 + \alpha\xi + \beta, \quad \text{zeros } \xi_1, \xi_2$$

$$\Rightarrow \xi_1 \cdot \xi_2 = \beta$$

Idea: (i) Compute "large" root (in modulus) ξ_2

(ii) $\xi_1 = \beta / \xi_2$

```
D = alpha^2-4*beta; % discriminant
if (D < 0), z = [];
else
   wD = 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)
      t = 0.5*(-alpha-wD);     ← no cancellation
      z = [t;beta/t];
   else
      t = 0.5*(-alpha+wD);     ← no cancellation
      z = [beta/t;t];
   end
end
```

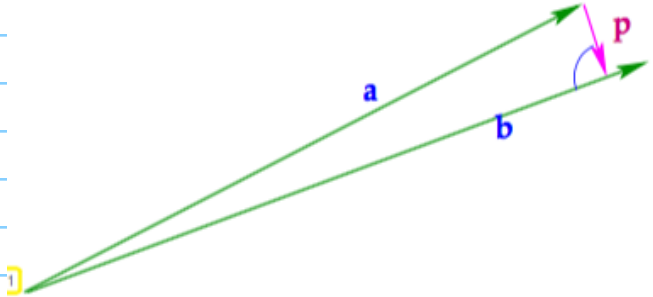


Ex 1.5.42: Cancellation & orthogonalization

$$p = a - \frac{a \cdot b}{b \cdot b} b$$

If $a \approx b$ $\Rightarrow$ $\|p\| \ll \|a\|, \|b\|$

$\hookrightarrow$ cancellation here



Ex 1.5.46 : Avoiding cancellation by trigonometric identities

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

cancellation here no cancellation

Exp : e^x by truncated exponential series $e^x \approx \sum_{k=0}^{N} \frac{1}{k!} x^k$

MATLAB-code 1.5.52: Summation of exponential series

```
function y = expeval(x,tol)
% Initialization
y=1; term=1; k=1;
% Termination
while
  (abs(term)>tol*~~min~~(y,~~1~~))*
% Next summand
  term = term*x/k;
% Summation
  y = y + term; %
  k = k+1;
end
```

* termination criterion

x	Approximation $\widetilde{\exp}(x)$	$\exp(x)$	$\frac{\lvert\exp(x)-\widetilde{\exp}(x)\rvert}{\exp(x)}$
-20	5.6218844674e-09	2.0611536224e-09	1.727542676201181
-18	1.5385415977e-08	1.5229979745e-08	0.010205938187564
-16	1.1254180496e-07	1.1253517472e-07	0.000058917020257
-14	8.3152907681e-07	8.3152871910e-07	0.000000430176956
-12	6.1442133148e-06	6.1442123533e-06	0.000000156480737
-10	4.5399929556e-05	4.5399929762e-05	0.000000004544414
-8	3.3546262817e-04	3.3546262790e-04	0.000000000788902
-6	2.4787521758e-03	2.4787521767e-03	0.000000000333306
-4	1.8315638879e-02	1.8315638889e-02	0.000000000530694
-2	1.3533528320e-01	1.3533528324e-01	0.000000000273603
0	1.0000000000e+00	1.0000000000e+00	0.000000000000000
2	7.3890560954e+00	7.3890560989e+00	0.000000000479969
4	5.4598149928e+01	5.4598150033e+01	0.000000001923058
6	4.0342879295e+02	4.0342879349e+02	0.000000001344248
8	2.9809579808e+03	2.9809579870e+03	0.000000002102584
10	2.2026465748e+04	2.2026465795e+04	0.000000002143800
12	1.6275479114e+05	1.6275479142e+05	0.000000001723845
14	1.2026042798e+06	1.2026042842e+06	0.000000003634135
16	8.8861105010e+06	8.8861105205e+06	0.000000002197990
18	6.5659968911e+07	6.5659969137e+07	0.000000003450972
20	4.8516519307e+08	4.8516519541e+08	0.000000004828738

$tol = 10^{-8}$

? cancellation

! $e^x \approx 0$ by summing relatively large terms with alternating signs

Remedy : $e^{-x} = 1/e^x$

Ex 1.5.53 : "Rather approximate than suffer cancellation"

$$I(a) := \int_0^1 e^{at} \, dt = 1/a \, (e^a - 1)$$

cancellation for $a \approx 0$

Idea : Taylor approximation $(a \geq 0)$

$$1/a \, (e^a - 1) = \underbrace{\sum_{k=0}^{m} \frac{1}{(k+1)!} a^k}_{=: S_m(a)} + \underbrace{\frac{1}{(m+1)!} e^{\xi} a^m}_{\text{Taylor remainder term}}$$

[no cancellation] $0 \leq \xi \leq a$

Issue : Choice of m to achieve rel. approximation error below prescribed tolerance

$$\text{rel. err.} = \left| \frac{S_m(a) - I(a)}{I(a)} \right| \overset{!}{\leq} tol$$

Use $I(a) \geq 1$

$$\leq \frac{1}{(m+1)!} e^a a^m \overset{!}{\leq} tol$$

$\rightarrow$ m can be found a priori for all $a \leq a_0$

```
if (abs(a) < 1E-4)
  v = 1.0 + (1.0/2 + 1.0/6*a)*a;
else
  v = (exp(a)-1.0)/a;
end
```

$\leftarrow$ Cancellation mild for big a

1.5.5. Numerical stability

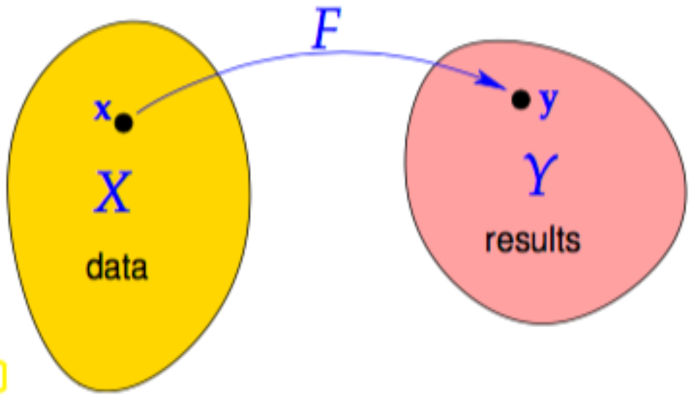
G.-S. orthogonormalization: "Good" and "bad" algorithms for same problem

"Good" → stable, "bad" → unstable

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.



$X, Y \triangleq$ finite-dimensional vector spaces $\triangleq \mathbb{K}^n$

[equipped with norm, e.g. Euclidean norm, max. norm]

Definition 1.5.72. Stable algorithm

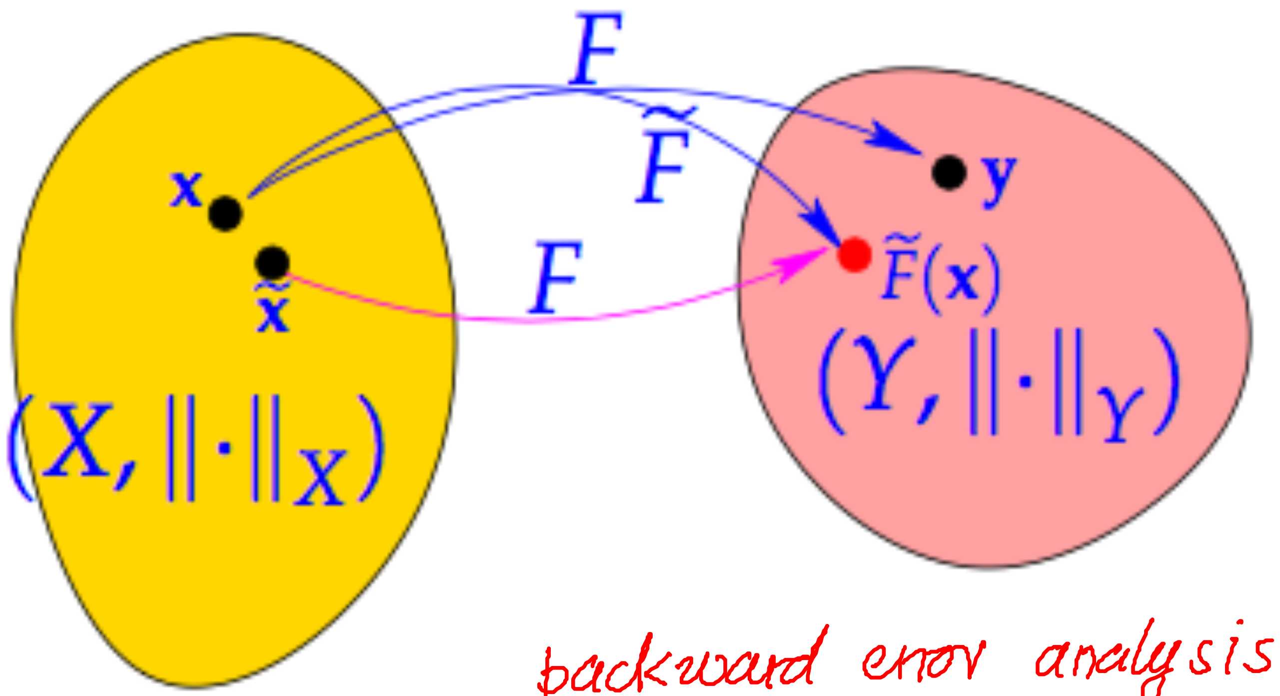
An algorithm $\widetilde{F}$ for solving a problem $F : X \mapsto Y$ is numerically stable, if for all $\mathbf{x} \in X$ its result $\widetilde{F}(\mathbf{x})$ (possibly affected by roundoff) is the exact result for "slightly perturbed" data:

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

Alg. stable: impact of roundoff errors* during execution is not (much) worse than the effect of rounding the input data.

$w(\mathbf{x})$: no. of elementary ops. in algorithm

* and approximation errors



backward error analysis

Ex.: Stability of matrix × vector

function y = multmv(A,x) stable ?*

Problem: $F : \begin{cases} X := \mathbb{K}^{m,n} \times \mathbb{K}^n \longrightarrow Y := \mathbb{K}^m \\ (A, \underline{x}) \longrightarrow A\underline{x} \end{cases}$

* Given $\tilde{y}$, when is there an $\hat{A} \in \mathbb{K}^{m,n}$ such that $\hat{A}x = \tilde{y}$, $\|A - \hat{A}\|_2 \leq C\, n\, \|A\|_2$

Possible choice: $\hat{A} = A + \underline{z}\,\underline{x}^T$, $\underline{z} = \dfrac{\tilde{y} - Ax}{\|x\|_2^2}$

$$\tilde{A}x = Ax + z\|x\|^2 = Ax + \tilde{y} - Ax = \tilde{y}$$

$$\|A - \tilde{A}\|_2 = \|zx^T\|_2 = \sup_{w \neq 0} \frac{\|zx^Tw\|_2}{\|w\|_2} = \|z\|_2 \|x\|_2$$

Recall: matrix norm : $\|M\|_2 = \sup_{x \neq 0} \frac{\|Mx\|_2}{\|x\|_2}$

$$= \|\tilde{y} - Ax\|_2 / \|x\|_2$$

$$\frac{\|A - \tilde{A}\|_2}{\|A\|_2} \leq Cn\,\text{eps} \,,\text{ if } \|\tilde{y} - Ax\|_2 \frac{1}{\|A\|_2\|x\|_2} \leq Cn \cdot \text{eps}$$

Note: If problem is sensitive ($\|F(x) - F(\tilde{x})\| \gg 1$ though $x \approx \tilde{x}$)

$\Rightarrow$ easy to find a stable algorithm !

1.6. Direct Methods for Linear Systems of Equations

Given $A \in \mathbb{K}^{n,n}$ (regular), $\underline{b} \in \mathbb{K}^n$, find $x \in \mathbb{R}^n$: $A\underline{x} = \underline{b}$

"Our problem" : $F : (A, \underline{b}) \rightarrow A^{-1}\underline{b}$

How tell that a solver for LSE is stable :
$\hookrightarrow$ yields $\tilde{\underline{x}} \in \mathbb{K}^n$

$$A\underline{x} = \underline{b} \quad \leftrightarrow \quad A\tilde{\underline{x}} = \hat{\underline{b}}$$

$$\frac{\|\underline{b} - \hat{\underline{b}}\|}{\|\underline{b}\|} = \frac{\|\underline{b} - A\tilde{\underline{x}}\|}{\|\underline{b}\|} = \frac{\|\underline{r}\|}{\|\underline{b}\|} \,,\quad \underline{r} := \underline{b} - A\hat{\underline{x}} \text{ residual}$$

stable, if $\boxed{\|\underline{r}\| \leq C\,w \cdot \text{EPS} \cdot \|\underline{b}\|}$

1.6.4. Elimination solvers for LSE

> Never contemplate implementing a general solver for linear systems of equations!
>
> If possible, use algorithms from numerical libraries! (→ Exp. 1.6.25)

cost (Solving a general LSE) = $O(n^3)$ [with small constant]

Cheaper for special matrices :

diagonal $\Downarrow$ $O(n)$, unitary $[x = A^H b]$ $O(n^2)$, triangular [forward, backward elim.] $O(n^2)$

$$\begin{bmatrix} \diagdown & \\ 0 & \end{bmatrix} \begin{bmatrix} \underline{x} \end{bmatrix} = \begin{bmatrix} \underline{b} \end{bmatrix}$$ "backward"

Matlab : $A\underline{x} = \underline{b}$: $x = A \backslash b$;

[$X = A\backslash B$; $B \in \mathbb{K}^{n,\ell}$ $\hat{=}$ multiple r.h.s. $(B)_{:,j}$, $j = 1,\dots,\ell$]

Eigen : $X = A.$<dec>$.\text{solve}(B)$

```cpp
#include <Eigen/Dense>
using namespace Eigen;
using namespace std;
.....
// Initialize a special invertible matrices
MatrixXd mat = MatrixXd::Identity(n,n) +
  VectorXd::Constant(n,1.0)*RowVectorXd::Constant(n,1.0);
cout << "Matrix mat = " << endl << mat << endl;
// Multiple right hand side vectors stored in matrix, cf. MATLAB
MatrixXd B = MatrixXd::Random(n,2);
// Solve linear system using various decompositions
MatrixXd X = mat.lu().solve(B);                 ← default
MatrixXd X2 = mat.fullPivLu().solve(B);
MatrixXd X3 = mat.householderQr().solve(B);
MatrixXd X4 = mat.llt().solve(B);
MatrixXd X5 = mat.ldlt().solve(B);
cout << "|X2-X| = " << (X2-X).norm() << endl;
cout << "|X3-X| = " << (X3-X).norm() << endl;
cout << "|X4-X| = " << (X4-X).norm() << endl;
cout << "|X5-X| = " << (X5-X).norm() << endl;
```

Elimination solve = setup phase + elimination phase

Complexity: setup phase $\downarrow$ $O(n^3)$; elimination phase $\downarrow$ $O(n^2)$

Matlab : $[L,U] = lu(A)$ $x = U \backslash (L \backslash b)$

$(A = LU)$

```matlab
% Setting: N ≫ 1, large matrix A
for j=1:N
    x = A\b;
    b = some_function(x);
end
```

Cost $O(Nn^3)$

```matlab
% Setting: N ≫ 1, large matrix A
[L,U] = lu(A);
for j=1:N
    x = U\(L\b);
    b = some_function(x);
end
```

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

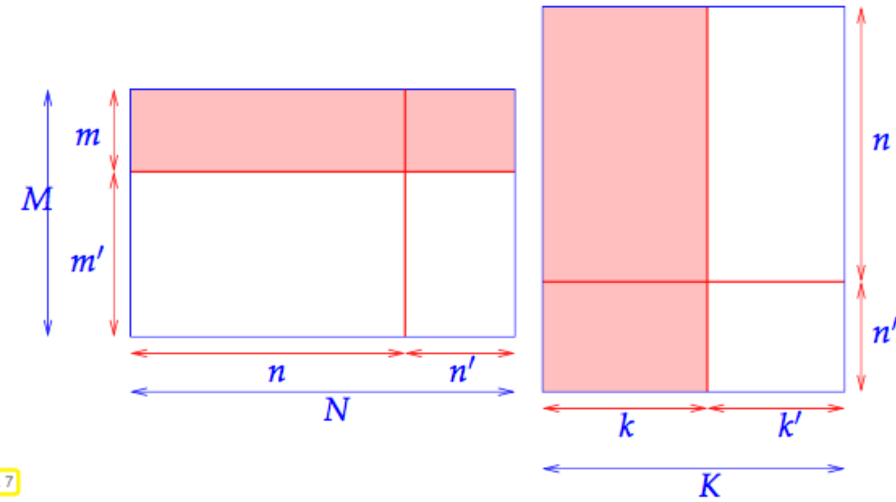
Eigen example :

```cpp
template<class VecType, class MatType>
VecType invpowit(const Eigen::MatrixBase<MatType> &A, double tol)
{
  using index_t = typename MatType::Index;
  using scalar_t = typename VecType::Scalar;
  // Make sure that the function is called with a square matrix
  const index_t n = A.cols();
  const index_t m = A.rows();
  eigen_assert(n == m);
  // Request LU-decomposition
  auto A_lu_dec = A.lu();                  ← O(n^3) cost
  // Initial guess for inverse power iteration
  VecType xo = VecType::Zero(n);
  VecType xn = VecType::Random(n);
  // Normalize vector
  xn /= xn.norm();
  // Terminate if relative (normwise) change below threshold
  while ((xo-xn).norm() > xn.norm()*tol) {
    xo = xn;
    xn = A_lu_dec.solve(xo);    ← cost O(n^2)
    xn /= xn.norm();
  }
  return(xn);
}
```

1.6.5. Exploiting structure of an LSE

Tool: Block elimination

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{11}\mathbf{B}_{11} + \mathbf{A}_{12}\mathbf{B}_{21} & \mathbf{A}_{11}\mathbf{B}_{12} + \mathbf{A}_{12}\mathbf{B}_{22} \\ \mathbf{A}_{21}\mathbf{B}_{11} + \mathbf{A}_{22}\mathbf{B}_{21} & \mathbf{A}_{21}\mathbf{B}_{12} + \mathbf{A}_{22}\mathbf{B}_{22} \end{bmatrix}. \tag{1.3.14}$$



Block partitioned linear system:

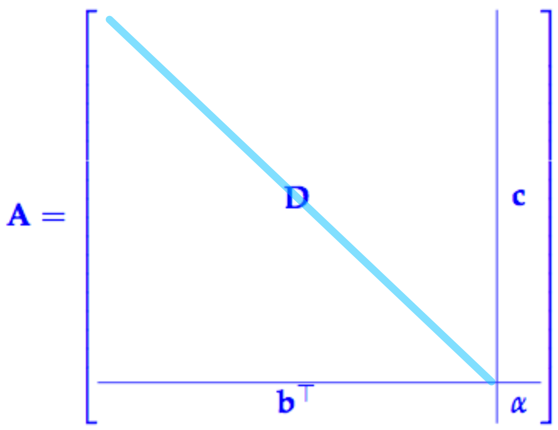
$$\begin{bmatrix} \mathbf{A}_{11} \mathbf{A}_{12} \\ \mathbf{A}_{21} \mathbf{A}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix}, \quad \begin{array}{l} \mathbf{A}_{11} \in \mathbb{K}^{k,k}, \mathbf{A}_{12} \in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

$$A_{11}x_1 + A_{12}x_2 = b_1 \quad \overset{A_{11}\text{ reg.}}{\Longrightarrow} \quad x_1 = A_{11}^{-1}(b_1 - A_{12}x_2)$$

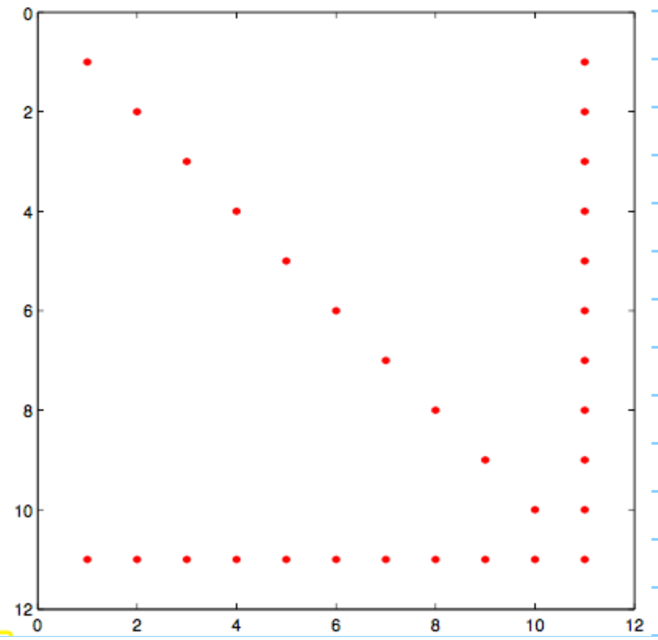
2nd equ. $\Rightarrow$
$$\underbrace{(A_{22} - A_{21}A_{11}^{-1}A_{12})}_{\text{Schur complement}} x_2 = b_2 - A_{21}A_{11}^{-1}b_1$$

Ex: LSE with arrow system matrix

$b, c \in \mathbb{R}^n$, $D \in \mathbb{R}^{n,n}$ diagonal, $\alpha \in \mathbb{R}$



(1.6.94)



$\longrightarrow$
$$\begin{bmatrix} D & \underline{c} \\ \underline{b}^T & \alpha \end{bmatrix} \begin{bmatrix} \underline{x}_1 \\ \xi \end{bmatrix} = \begin{bmatrix} \underline{y}_1 \\ \eta \end{bmatrix}$$

```
function x = arrowsys_slow(d,c,b,alpha,y)
A = [diag(d),c;transpose(b),alpha];
x = A\y;
```

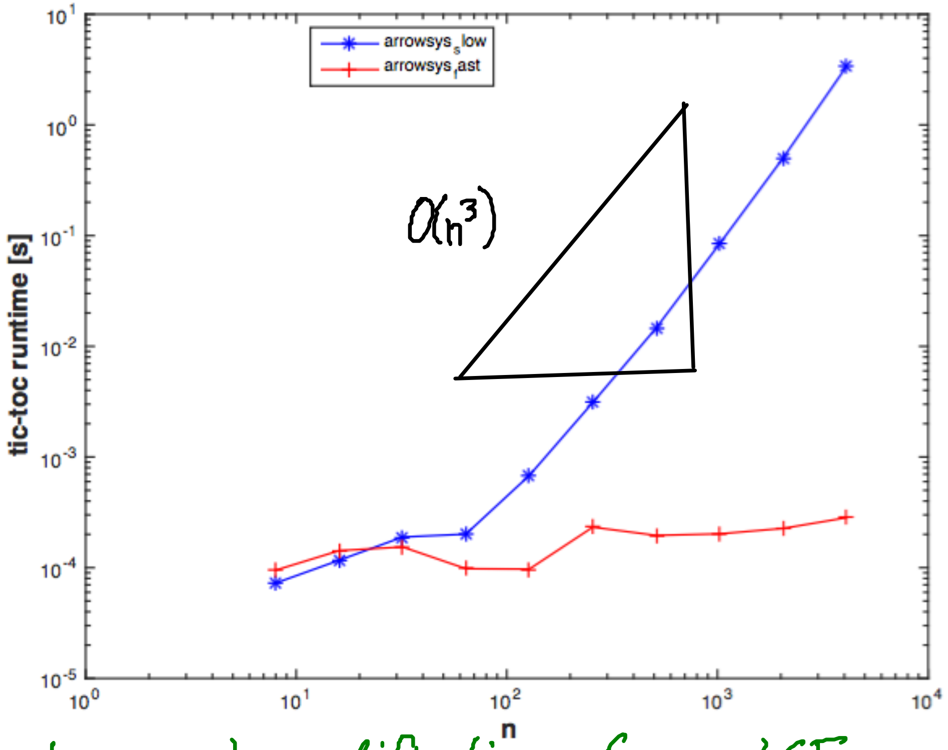
Cost $O(n^3)$

Smarter: Block elimination $(\alpha - \underline{b}^T D^{-1}\underline{c})\,\xi = \eta - \underline{b}^T D^{-1}\underline{y}_1$

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

```
function x = arrowsys_fast(d,c,b,alpha,y)
z = c./d;                  % z = D^-1 c
w = y(1:end-1)./d;  % w = D^-1 y_1
xi = (y(end)-dot(b,w))/(alpha - dot(b,z));
x = [ w-xi*c./d;xi];
```

Cost $O(n)$



How to tell polynomial complexity from runtime data (n_i, t_i)

If $t_i \approx C n_i^p$

$\Rightarrow$ $\log t_i \approx \log C + p \log n_i$

$\downarrow$

Straight line with slope p in doubly logarithmic plot

Low rank modification of an LSE

$Ax = b$ has already been solve (setup phase done)

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

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

▶ $\boxed{\tilde{\mathbf{A}} = \mathbf{A} + z \cdot \mathbf{e}_{i^*}\mathbf{e}_{j^*}^T}$.

General : rank-1-modification $\tilde{A} = A + u v^T$

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

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

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

B.E. of $\hat{x}$: $\quad (-1 - v^T A^{-1} u)\xi = v^T A^{-1} b$

$$\Rightarrow \quad A\tilde{x} = b - u \frac{v^T A^{-1} b}{1 + v^T A^{-1} u}$$

```
function x = smw(L,U,u,v,b)
z = U\(L\b); w = U\(L\u);
alpha = 1+dot(v,w);
if (abs(alpha) <
   eps*norm(U,1)),
   error('Nearly singular
   matrix'); end;
x = z - w*dot(v,z)/alpha;
```

$\rightarrow O(n^2)$

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} - \frac{\mathbf{A}^{-1}\mathbf{u}(\mathbf{v}^H(\mathbf{A}^{-1}\mathbf{b}))}{\underbrace{(1 + \mathbf{v}^H(\mathbf{A}^{-1}\mathbf{u}))}_{=\alpha}}$$

Cost $O(n^2)$

1.7. Sparse linear system

↳ "most entries of system matrix = 0"

Notion 1.7.1. Sparse matrix

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

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

1.7.1. Sparse matrix storage formats

Memory ~ $O(\mathrm{nnz}(A))$ cost (Matrix × Vector) ~ $O(\mathrm{nnz}(A))$

Example: triplet format (COO)

$\hat{=}$ List $\{(i_k, j_k, a_k)\}_{k=1}^{N}$

row index col. index "entry" : repetitions of index pairs possible

```
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
};
```

C++-code 1.7.7: Matrix×vector product $y = Ax$ in triplet format

```
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]];
  }
```

$y = y + Ax$

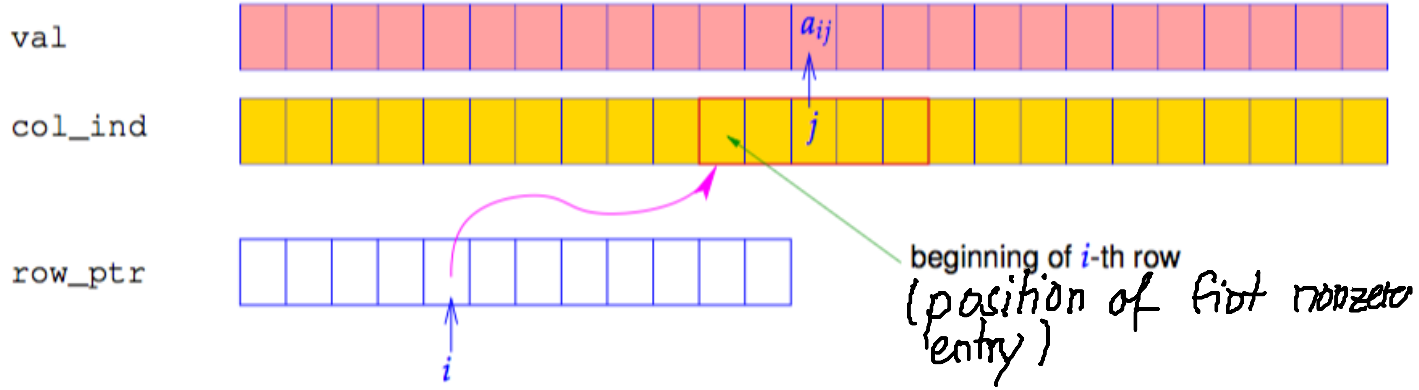
Cost $O(N)$

CRS (compressed row format) format:

$A \in \mathbb{K}^{n,n}$

`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

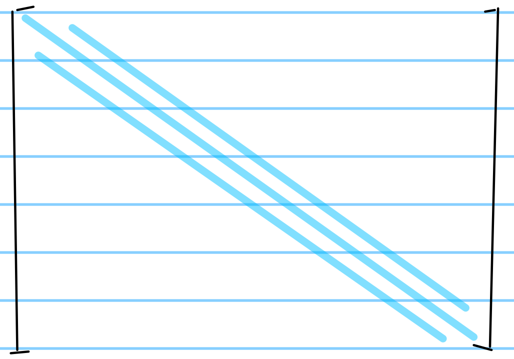
[Matlab indexing!]

CCS = CRS for A^T

1.7.2. Sparse matrices in Matlab

Dedicated functions : Initialization

`A = sparse(m,n);`	create empty $m \times n$ "sparse matrix"
`A = spalloc(m,n,nnz);`	create $m \times n$ sparse matrix & reserve memory
`A = sparse(I,J,a,m,n);`	initialize $m \times n$ sparse matrix from triplets → § 1.7.6
`A = spdiags(B,d,m,n);`	create sparse banded matrix → Section 1.7.6
`A = speye(n);`	sparse identity matrix



$\in \mathbb{R}^{n,n}$: $B \in \mathbb{R}^{n,3}$

$d = [-1, 0, 1]$

lower main upper diagonal

→ 'doc spdiags'

(1.7.13) Efficient initialization of sparse matrices

MATLAB-code 1.7.14: Initialization of sparse matrices: entry-wise (I)

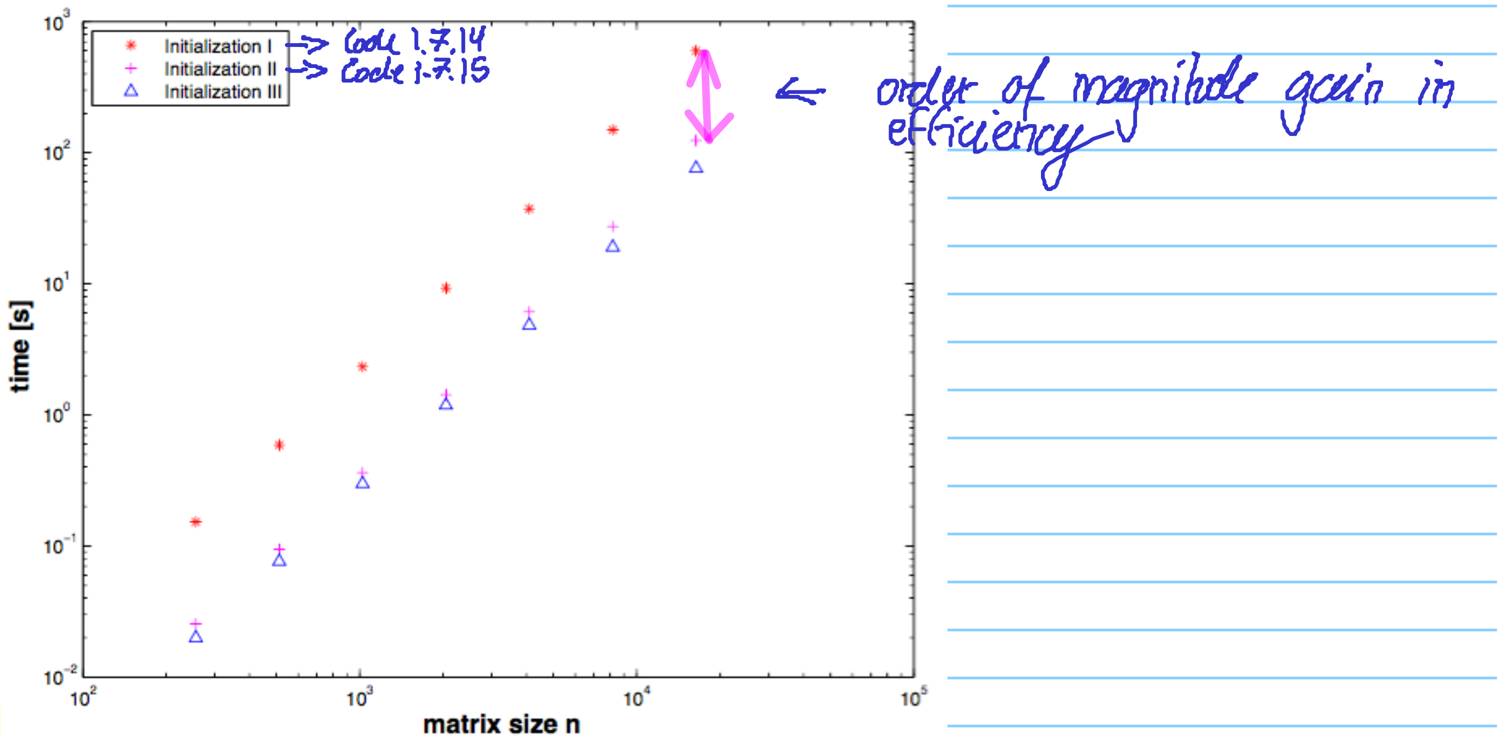
```
A1 = sparse(n,n);
for i=1:n
    for j=1:n
        if (abs(i-j) == 1), A1(i,j) = A1(i,j) + 1; end;
        if (abs(i-j) == round(n/3)), A1(i,j) = A1(i,j) -1; end;
end; end
```

→ Enormous amount of allocation & copying !

MATLAB-code 1.7.15: Initialization of sparse matrices: triplet based (II)

```
dat = [];
 for i=1:n
    for j=1:n
      if (abs(i-j) == 1), dat = [dat; i,j,1.0]; end;
      if (abs(i-j) == round(n/3)), dat = [dat; i,j,-1.0];
 end; end; end;
 A2 = sparse(dat(:,1),dat(:,2),dat(:,3),n,n);
```

→ Build CRS format



1.7.3 Sparse matrices in Eigen

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

scalar type

Initialization: from triplet format (as in Matlab)

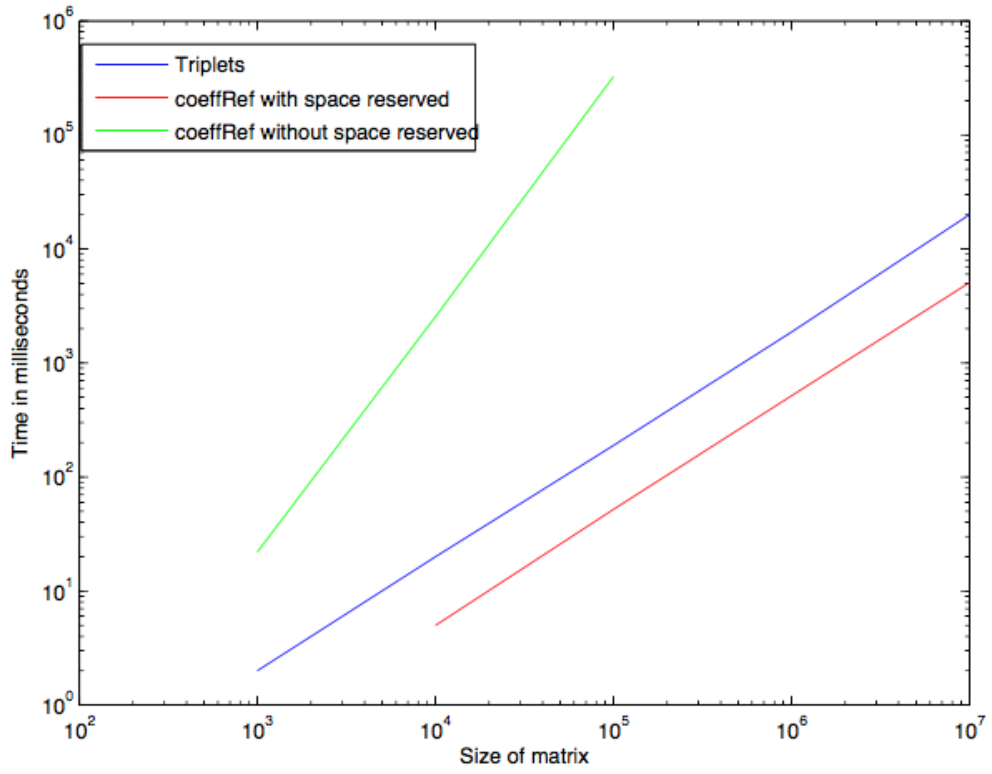
```
std::vector <Eigen::Triplet <double >  > triplets;
// .. fill the std::vector triplets ..
Eigen::SparseMatrix<double, Eigen::RowMajor> spMat(rows, cols);
spMat.setFromTriplets(triplets.begin(), triplets.end());
spMat.makeCompressed(); → build CRS/CCS
```

```
unsigned int row_idx = 2;
unsigned int col_idx = 4;
double value = 2.5;
Eigen::Triplet<double> triplet(row_idx,col_idx,value);
std::cout << '(' << triplet.row() << ',' << triplet.col()
          << ',' << triplet.value() << ')' << std::endl;
```

Remark: reserve() - method for sparse matrice
→ preallocation of space

```
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();
```

Runtimes for initialization of banded matrix in Eigen



1.7.4 Direct solution of sparse linear systems

↓

system matrix in sparse format

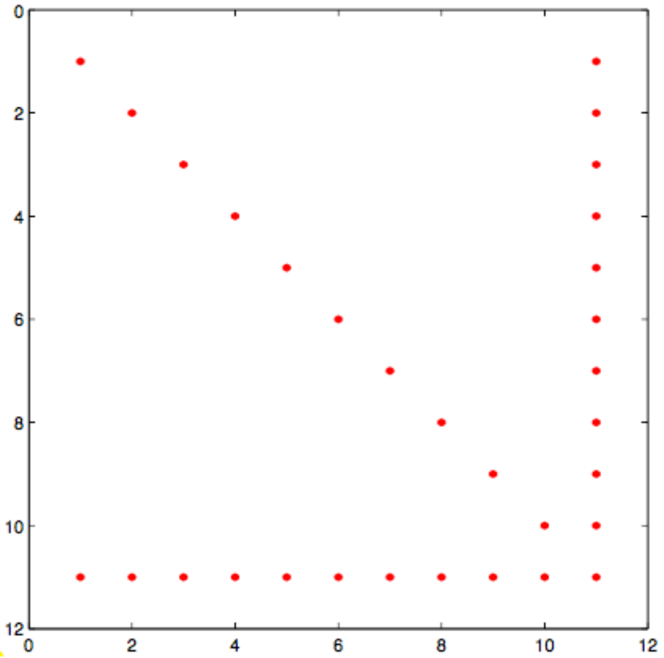
↓

use special sparse elimination algorithms

Matlab: still '\'

Example: Arrow matrix [Recall: $O(n)$ solution alg.]

$$A = \left[\begin{array}{c|c} D & c \\ \hline b^\top & \alpha \end{array}\right] \tag{1.6.94}$$



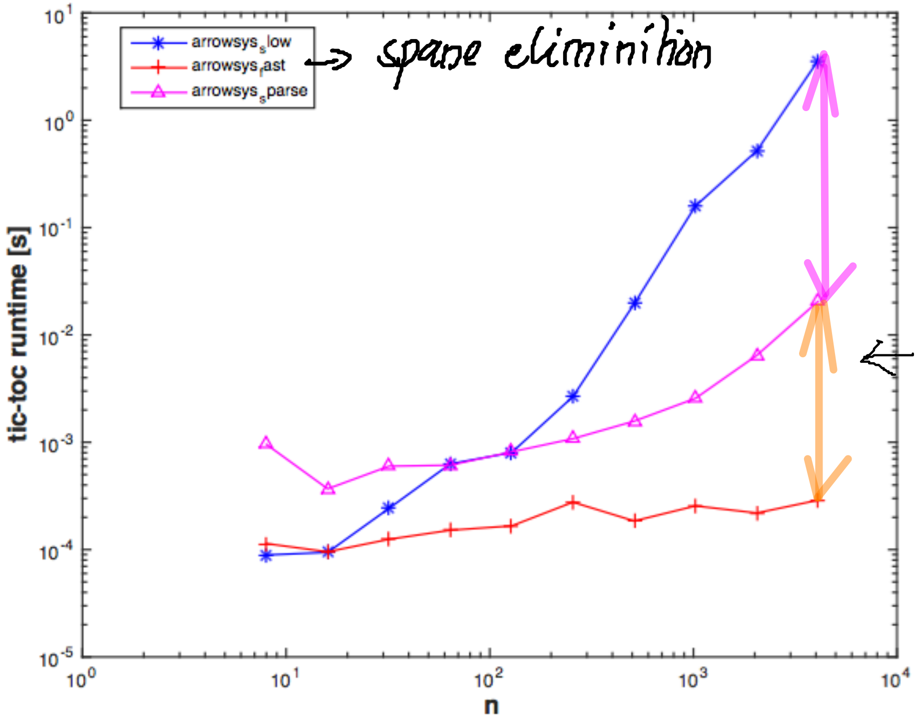
```
A = [diag(d),c;transpose(b),alpha];
x = A\y;
```

↳ dense

MATLAB-code 1.7.38: Invoking sparse elimination solver for arrow matrix

```
function x = arrowsys_sparse(d,c,b,alpha,y)
n = numel(d);
A = [spdiags(d,[0],n,n),c;transpose(b),alpha];
x = A\y;
```

→ sparse format



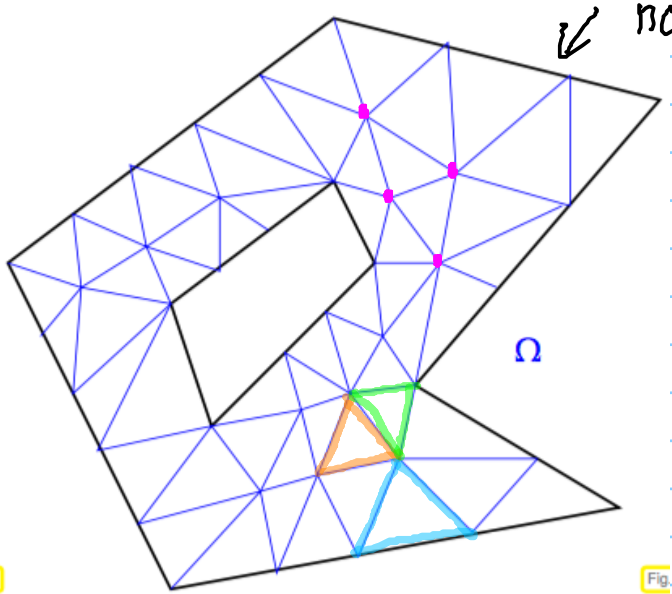
← Overhead for matrix scanning & pivoting

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

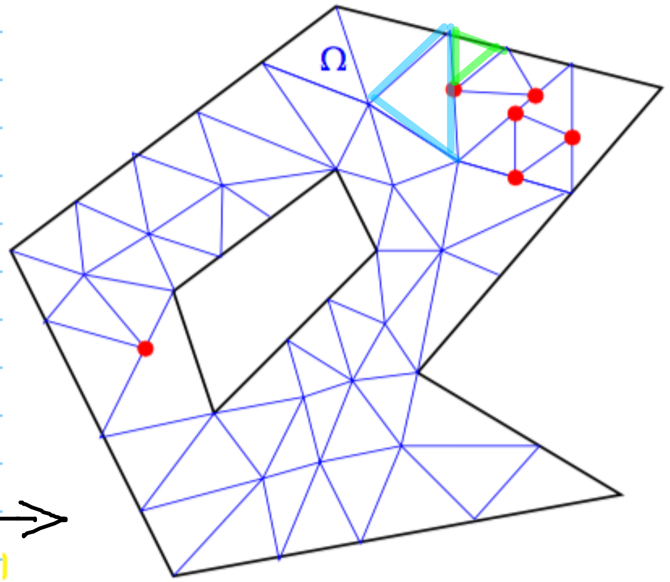
System matrices are passed to these algorithms in sparse storage formats (→ 1.7.1) to convey information about zero entries.

Case study : Smoothing of (planar) triangulation

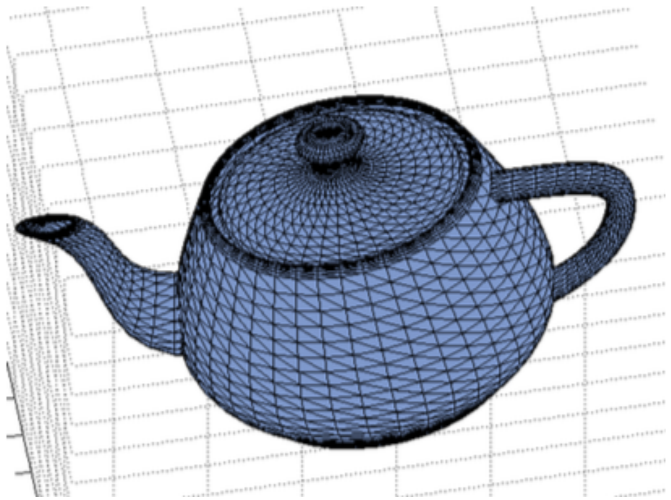
no gaps, no overlap



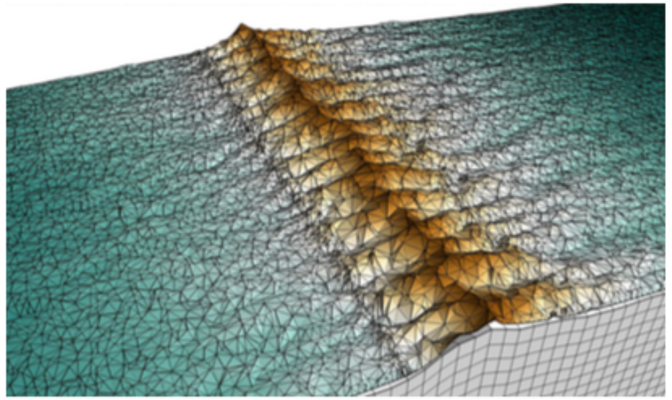
not a valid triangulation →



Mesh with “illegal” hanging nodes



Computer graphics



GIS

Definition 1.7.22. Planar triangulation

A planar triangulation (mesh) $\mathcal{M}$ consists of a set $\mathcal{N}$ of $N \in \mathbb{N}$ distinct points $\in \mathbb{R}^2$ and a set $\mathcal{T}$ of triangles with vertices in $\mathcal{N}$, such that the following two conditions are satisfied:

1. the interiors of the triangles are mutually disjoint (“no overlap”),
2. for every two *closed* distinct triangles $\in \mathcal{T}$ their intersection satisfies exactly one of the following conditions:
   (a) it is empty
   (b) it is exactly one vertex from $\mathcal{N}$,
   (c) it is a common edge of both triangles

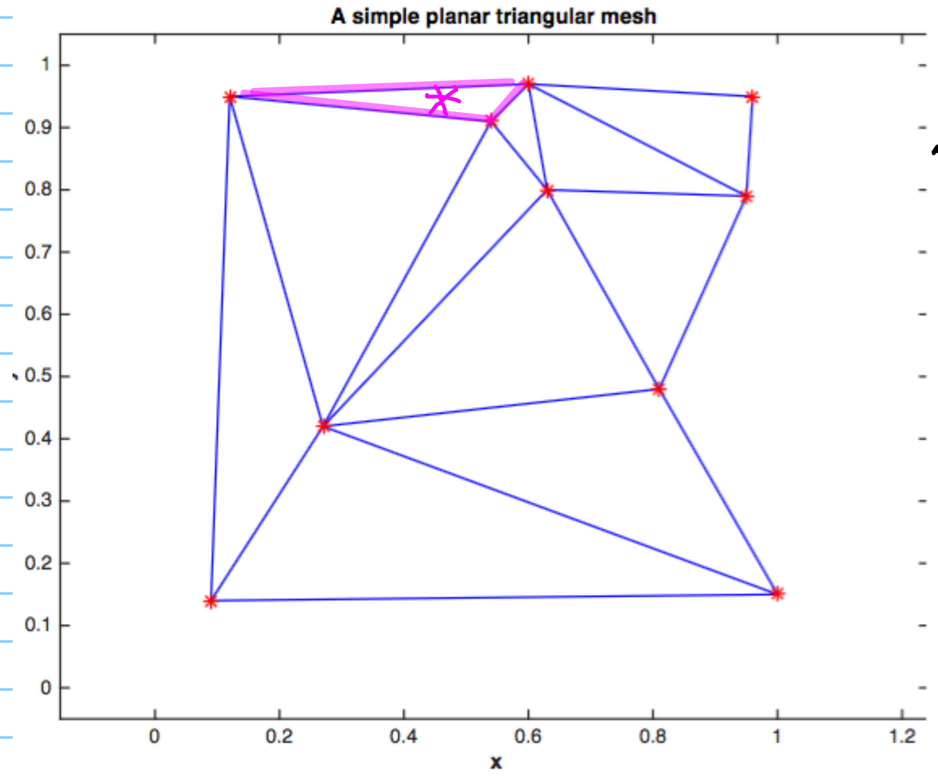
```
% MATLAB demonstration for visualizing a planes triangular mesh
% Initialize node coordinates
% First the x-coordinates
x = [1.0;0.60;0.12;0.81;0.63;0.09;0.27;0.54;0.95;0.96];
% Next the y-coordinates
y = [0.15;0.97;0.95;0.48;0.80;0.14;0.42;0.91;0.79;0.95];
% Then specify triangles through the indices of their vertices.  These
% indices refer to the ordering of the coordinates as given in the
% vectors x and y.
T = [8 2 3;6 7 3;5 2 8;7 8 3;7 5 8;7 6 1;...
     4 7 1;9 5 4;4 5 7;9 2 5;10 2 9];
% Call the plotting routine; draw mesh with blue edges
triplot(T,x,y,'b-'); title ('A simple planar triangular mesh');
xlabel('{\bf x}'); ylabel('{\bf y}');
axis([-0.05 1.05 -0.05 1.05]); axis equal;
% Mark nodes with red stars
hold on; plot(x,y,'r*');

% Save plot a vector graphics
print -depsc2 'meshplot.eps';
```

NCSE15

MATLAB "data structure" for triangulations:

- Coordinate vectors of length N ($\hat{=}$ no. of nodes)
- Node-triangle incidence matrix $T \in \mathcal{N}^{M,3}$ ($M \hat{=}$ no. of triangles)



$\triangle$ Output of above code

$*$ distorted triangle

$\downarrow$

Move interior nodes to improve shape ("smoothing")

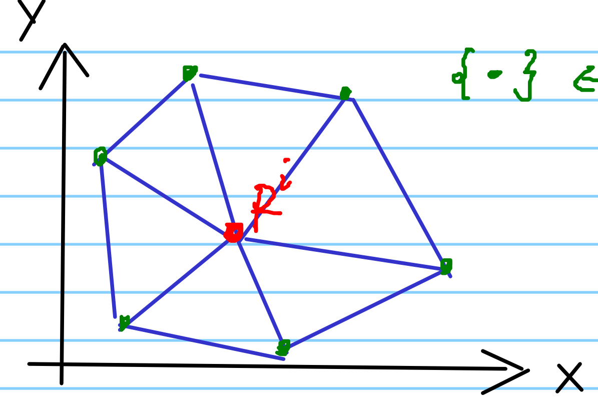
$S(i) := \{j \in \{1,\dots,N\}$: nodes i and j are connected by an edge$\}$, [Set of neighbors]

Definition 1.7.26. Smoothed triangulation

A triangulation is called smoothed, if [$\mathbf{p}^i \hat{=}$ position of node #i]

$$\mathbf{p}^i = \frac{1}{\sharp S(i)} \sum_{j \in S(i)} \mathbf{p}^j \quad \Leftrightarrow \quad \sharp S(i) p_d^i = \sum_{j \in S(i)} p_d^j,\ d = 1,2, \quad \text{for all} \quad i \in \{1,\dots,N\} \setminus \Gamma, \quad (1.7.27)$$

that is, every interior node is located in the center of gravity of its neighbours.



$\{\cdot\} \Leftrightarrow S(i)$

Note: $(1.7.27) \Leftrightarrow$ LSE $[\mathbf{C}\underline{\mathbf{z}} = 0]$

$\hookrightarrow$ describes two rows of linear system

$n \hat{=}$ no. of interior nodes : $\mathbf{C} = \mathbb{R}^{2n,2N}$

$$\underline{\mathbf{z}} \in \mathbb{R}^{2N}, \quad z_j := \begin{cases} p_x^j, & 1 \le j \le N \\ p_y^{j-N}, & N+1 \le j \le 2N \end{cases}$$

$\uparrow$ vector of node coordinates

Note: x,y-coordinates averaged independently

$$\Rightarrow \quad \mathbf{C} = \left[\begin{array}{c|c} \mathbf{A} & \mathbf{O} \\ \hline \mathbf{O} & \mathbf{A} \end{array}\right] \qquad \mathbf{A} \in \mathbb{R}^{N,N}$$

$$(\mathbf{A})_{i,j} = \begin{cases} -\sharp S(i) & \text{if } i = j \\ 1 & \text{if } j \in S(i) \\ 0 & \text{else.} \end{cases}$$

$\hookrightarrow$ (sparse) combinatorial graph Laplacian

Note: Position of boundary nodes are known

Assume: Boundary nodes numbered before interior nodes

$$\mathbf{z}^T = \begin{bmatrix} \mathbf{z}_1^{\text{int}} \\ \mathbf{z}_1^{\text{bd}} \\ \mathbf{z}_2^{\text{int}} \\ \mathbf{z}_2^{\text{bd}} \end{bmatrix} := [z_1,\dots,z_n,z_{n+1},\dots,z_N,z_{N+1},\dots,z_{N+n},z_{N+n+1},\dots,z_{2N}]^{\top}$$

$$\Updownarrow$$

$C\underline{z} = 0 \iff$

$$\begin{bmatrix} \mathbf{A}_{\text{int}} & \mathbf{A}_{\text{bd}} & & \\ & & \mathbf{A}_{\text{int}} & \mathbf{A}_{\text{bd}} \end{bmatrix} \begin{bmatrix} z_1^{int} \\ z_1^{bd} \\ z_2^{int} \\ z_2^{bd} \end{bmatrix} = \mathbf{0}\,.$$

Move known values to r.h.s. $\longrightarrow$ $\Updownarrow$

$$\left[\begin{array}{c|c} A_{int} & 0 \\ \hline 0 & A_{int} \end{array}\right] \begin{bmatrix} z_1^{int} \\ z_2^{int} \end{bmatrix} = \begin{bmatrix} -A_{bd}\, z_1^{bd} \\ -A_{bd}\, z_2^{bd} \end{bmatrix}$$

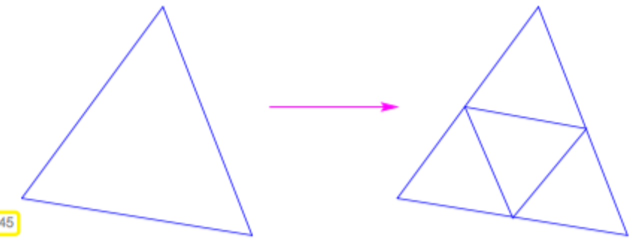
$\hookrightarrow$ square linear system with sparse system matrix

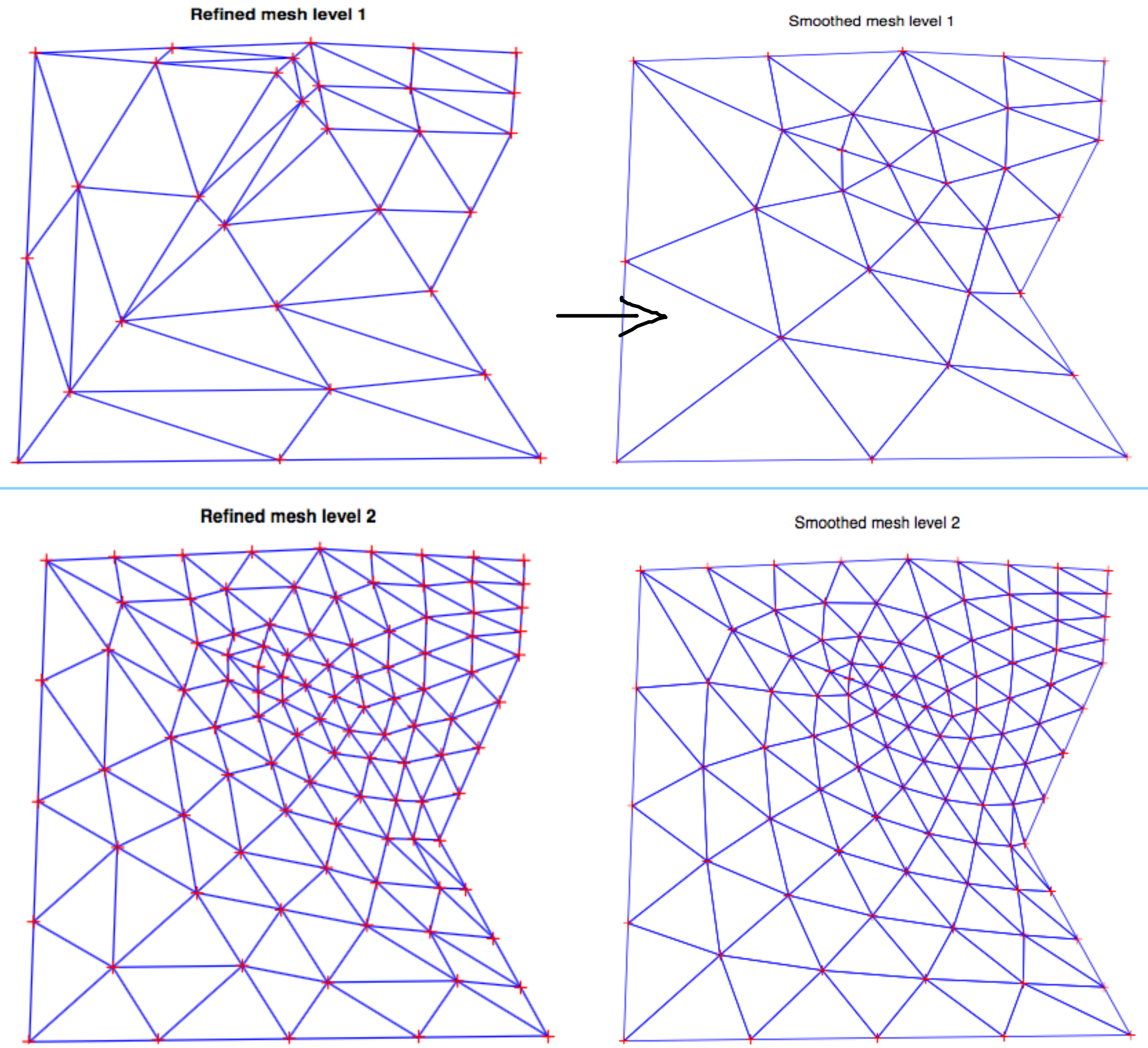
Experiment

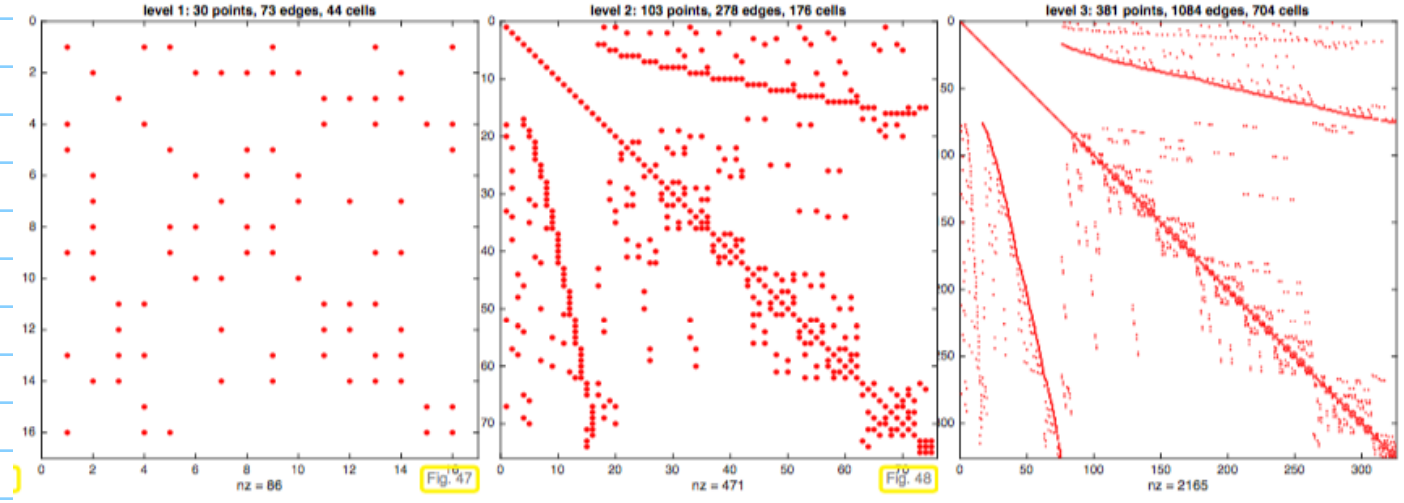
Definition 1.7.33. Regular refinemnent of a planar triangulation

The planar triangulation with cells obtained by splitting all cells of a planar triangulation $\mathcal{M}$ into four congruent triangles is called the regular refinement of $\mathcal{M}$.

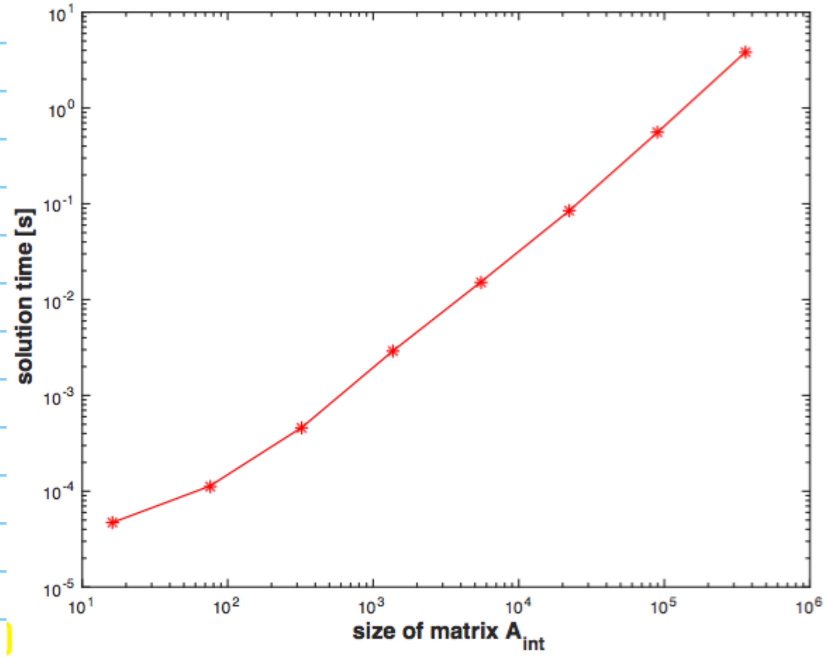
Fig. 45







"spy-plot" $\hat{=}$ sparsity pattern of comb. graph Laplacian



◁ tic-toc timing for "$y = A_{int} \backslash b$"

Empiric complexity $\approx O(n^{1.6})$