Matrix algorithms

Agenda

- Master theorem
- Multiplication: numbers and matrices
- Other notable matrix problems
  - Inverse: naive, LU
  - Pseudoinverse: SVD, QR
  - Compression: SVD

Analysing recursion

Dynamic programming

```
# homogenous linear recurrence relation with
# constant koefficients - use characteristic equation

def fib(n):
    if n < 2: return 1
    return fib(n - 1) + fib(n - 2)

[fib(i) for i in range(10)]
```

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

Chain method

```python
# T(n) = n * T(n-1) + c -- use chain method

import numpy as np

def minor(A, row, col):
    return np.delete(np.delete(A, col, axis=1), row, axis=0)

def det(A):
    assert A.shape[0] == A.shape[1], "Dimensions don't match"
    if A.shape == (1, 1):
        return A[0, 0]
    s = 0.0
    for i in range(A.shape[0]):
        s += (-1) ** i * A[0, i] * det(minor(A, 0, i))
    return s

A = np.matrix([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
print(A)
det(A)
```

```
[[1 0 0]
 [0 1 0]
 [0 0 1]]

1.0
```

Master theorem

Integer multiplication

Fixed size **integer numbers** (32, 64 bits) multiplication is limited by square root of max value. Thus, for **unsigned long long** in will be just **INT_MAX**

Bignum arithmetics (+,-,*,/) allows bigger numbers, but depends on the length of numbers. How?

Stolbik :)

```
def mult(first, second, base=10):
  result = 0
  for c in second:
    subresult = 0
    for d in first:
      subresult *= base
      subresult += int(d, base) * int(c, base)
    result *= base
    result += subresult
  return result
```

Karatsuba’s method

$$(ax+b)(cx+d) = acx^2 + axd + cxb + bd = acx^2 + x(ad+bc) + bd$$

$$(a \underset{1}{+} b\underset{1}{)(}c \underset{2}{+} d) \underset{3}{-} ac \underset{4}{-} bd = \underset{1}{ad} \underset{1}{+} \underset{2}{bc}$$

Karatsuba multiplication

```python
# T(n) = a*T(n/b) + O(n^c) - use Master theorem

def karatsuba(x, y):
    if len(str(x)) == 1 or len(str(y)) == 1:
        return x * y
    else:
        n = max(len(str(x)), len(str(y)))
        nby2 = n // 2
        a = x // (10 ** (nby2))
        b = x % (10 ** (nby2))
        c = y // (10 ** (nby2))
        d = y % (10 ** (nby2))
        ac = karatsuba(a, c)
        bd = karatsuba(b, d)
        ad_plus_bc = karatsuba(a + b, c + d) - ac - bd
        return ac * 10**(2*nby2) + (ad_plus_bc * 10**nby2) + bd


karatsuba(100001, 54321)
```

```
5432154321
```

Master theorem $T(n) = a\,T\left(\frac{n}{b}\right) + f(n),$

Let's call $c_{\text{crit}} = \log_b a$

We have 3 options:

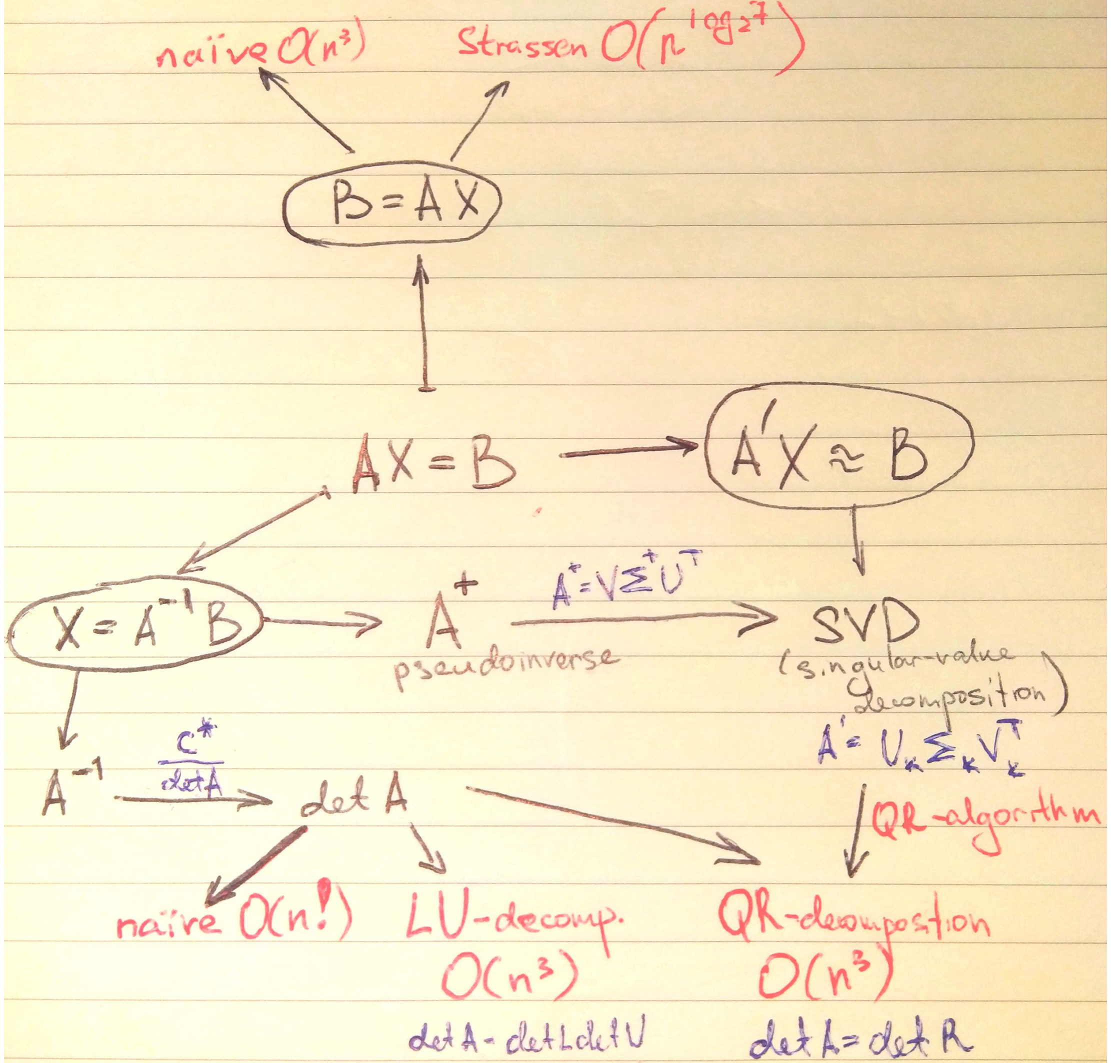
$f(n) = O(n^c)$

1. **Branching** dominates ***c < c**crit*, then $T(n) = \Theta\left(n^{c_{\text{crit}}}\right)$

$f(n) = \Omega(n^c)$

2. "**Tail**" dominates ***c > c**crit*, then $T(n) = \Theta\left(f(n)\right)$

3. **Comparable** computation $f(n) = \Theta(n^{c_{\text{crit}}} \log^k n)$, then $T(n) = \Theta\left(n^{c_{\text{crit}}} \log^{k+1} n\right)$


naïve $O(n^3)$
Strassen $O(n^{\log_2 7})$
$B = AX$
$AX = B$
$A'X \approx B$
$X = A^{-1}B$
A^+
pseudoinverse
$A^+ = V\Sigma^+U^T$
SVD
(singular-value decomposition)
$A' = U_k \Sigma_k V_k^T$
QR-algorithm
A^{-1}
$\frac{C^*}{\det A}$
det A
naïve $O(n!)$
LU-decomp.
$O(n^3)$
QR-decomposition
$O(n^3)$
$\det A = \det L \det U$
$\det A = \det R$

Matrix multiplication: $B = AX$

Naive matrix multiplication

Input: matrices A and B

Let C be a new matrix of the appropriate size

For i from 1 to n:

- For j from 1 to p:
  - Let $\text{sum} = 0$
  - For k from 1 to m:
    - Set $\text{sum} \leftarrow \text{sum} + A_{ik} \times B_{kj}$
  - Set $C_{ij} \leftarrow \text{sum}$

Return C

Cache-friendly naive multiplication. Distributed

Input: matrices A and B

Let C be a new matrix of the appropriate size

Pick a tile size $T = \Theta(\sqrt{M})$

For I from 1 to n in steps of T:

- For J from 1 to p in steps of T:
  - For K from 1 to m in steps of T:
    - Multiply $A_{I:I+T,\, K:K+T}$ and $B_{K:K+T,\, J:J+T}$ into $C_{I:I+T,\, J:J+T}$, that is:
    - For i from I to $\min(I + T, n)$:
      - For j from J to $\min(J + T, p)$:
        - Let $\mathrm{sum} = 0$
        - For k from K to $\min(K + T, m)$:
          - Set $\mathrm{sum} \leftarrow \mathrm{sum} + A_{ik} \times B_{kj}$
        - Set $C_{ij} \leftarrow C_{ij} + \mathrm{sum}$

Return C

Divide-and-Conquer

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

$$T(n) = 8T(n/2) + \Theta(n^2)$$

Strassen algorithm idea

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

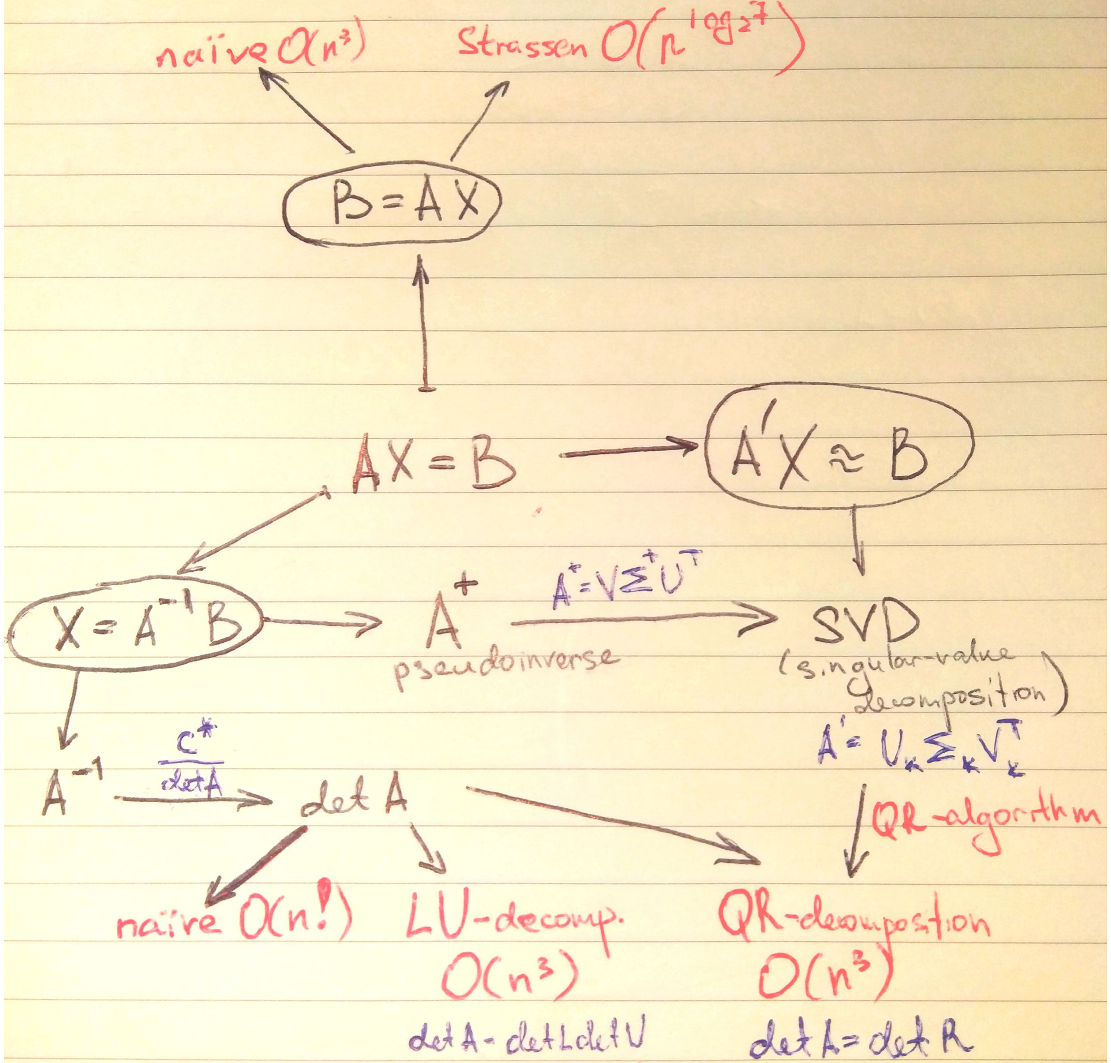
$$\begin{aligned} \mathbf{C}_{1,1} &= \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1} \\ \mathbf{C}_{1,2} &= \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2} \\ \mathbf{C}_{2,1} &= \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1} \\ \mathbf{C}_{2,2} &= \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2} \end{aligned}$$

8 multiplications
4 sums

$$\begin{aligned} \mathbf{P}_1 &:= (\mathbf{A}_{1,1} \underset{1}{+} \mathbf{A}_{2,2})\underset{1}{}(\mathbf{B}_{1,1} \underset{2}{+} \mathbf{B}_{2,2}) \\ \mathbf{P}_2 &:= (\mathbf{A}_{2,1} \underset{3}{+} \mathbf{A}_{2,2})\underset{2}{}\mathbf{B}_{1,1} \\ \mathbf{P}_3 &:= \mathbf{A}_{1,1}\underset{3}{}(\mathbf{B}_{1,2} \underset{4}{-} \mathbf{B}_{2,2}) \\ \mathbf{P}_4 &:= \mathbf{A}_{2,2}\underset{4}{}(\mathbf{B}_{2,1} \underset{5}{-} \mathbf{B}_{1,1}) \\ \mathbf{P}_5 &:= (\mathbf{A}_{1,1} \underset{6}{+} \mathbf{A}_{1,2})\underset{5}{}\mathbf{B}_{2,2} \\ \mathbf{P}_6 &:= (\mathbf{A}_{2,1} \underset{7}{-} \mathbf{A}_{1,1})\underset{6}{}(\mathbf{B}_{1,1} \underset{8}{+} \mathbf{B}_{1,2}) \\ \mathbf{P}_7 &:= (\mathbf{A}_{1,2} \underset{9}{-} \mathbf{A}_{2,2})\underset{7}{}(\mathbf{B}_{2,1} \underset{10}{+} \mathbf{B}_{2,2}) \end{aligned}$$

$$\begin{aligned} \mathbf{C}_{1,1} &= \mathbf{P}_1 \underset{11}{+} \mathbf{P}_4 \underset{12}{-} \mathbf{P}_5 \underset{13}{+} \mathbf{P}_7 \\ \mathbf{C}_{1,2} &= \mathbf{P}_3 \underset{14}{+} \mathbf{P}_5 \\ \mathbf{C}_{2,1} &= \mathbf{P}_2 \underset{15}{+} \mathbf{P}_4 \\ \mathbf{C}_{2,2} &= \mathbf{P}_1 \underset{16}{-} \mathbf{P}_2 \underset{17}{+} \mathbf{P}_3 \underset{18}{+} \mathbf{P}_6 \end{aligned}$$

Matrix inverse. $X = A^{-1}B$


naïve O(n^3)
Strassen O(n^log2 7)
B = AX
AX = B
A'X ≈ B
X = A^-1 B
A^+
pseudoinverse
A^+ = VΣ^+U^T
SVD
(singular-value decomposition)
A' = U_k Σ_k V_k^T
A^-1
C* / det A
det A
QR-algorithm
naïve O(n!)
LU-decomp.
O(n^3)
det A = det L det U
QR-decomposition
O(n^3)
det A = det R

Matrix inverse

*C** - adjugate matrix, consists of **minors**

$$A^{-1} = \frac{C^*}{\det(A)}$$

Dodgson condensation

$$\begin{vmatrix} -2 & -1 & -1 & -4 \\ -1 & -2 & -1 & -6 \\ -1 & -1 & 2 & 4 \\ 2 & 1 & -3 & -8 \end{vmatrix}.$$

1. Construct (N-1)x(N-1) matrix ***B*** of 2x2 minors

$$B = \begin{bmatrix} \begin{vmatrix} -2 & -1 \\ -1 & -2 \end{vmatrix} & \begin{vmatrix} -1 & -1 \\ -2 & -1 \end{vmatrix} & \begin{vmatrix} -1 & -4 \\ -1 & -6 \end{vmatrix} \\ \begin{vmatrix} -1 & -2 \\ -1 & -1 \end{vmatrix} & \begin{vmatrix} -2 & -1 \\ -1 & 2 \end{vmatrix} & \begin{vmatrix} -1 & -6 \\ 2 & 4 \end{vmatrix} \\ \begin{vmatrix} -1 & -1 \\ 2 & 1 \end{vmatrix} & \begin{vmatrix} -1 & 2 \\ 1 & -3 \end{vmatrix} & \begin{vmatrix} 2 & 4 \\ -3 & -8 \end{vmatrix} \end{bmatrix} = \begin{bmatrix} 3 & -1 & 2 \\ -1 & -5 & 8 \\ 1 & 1 & -4 \end{bmatrix}.$$

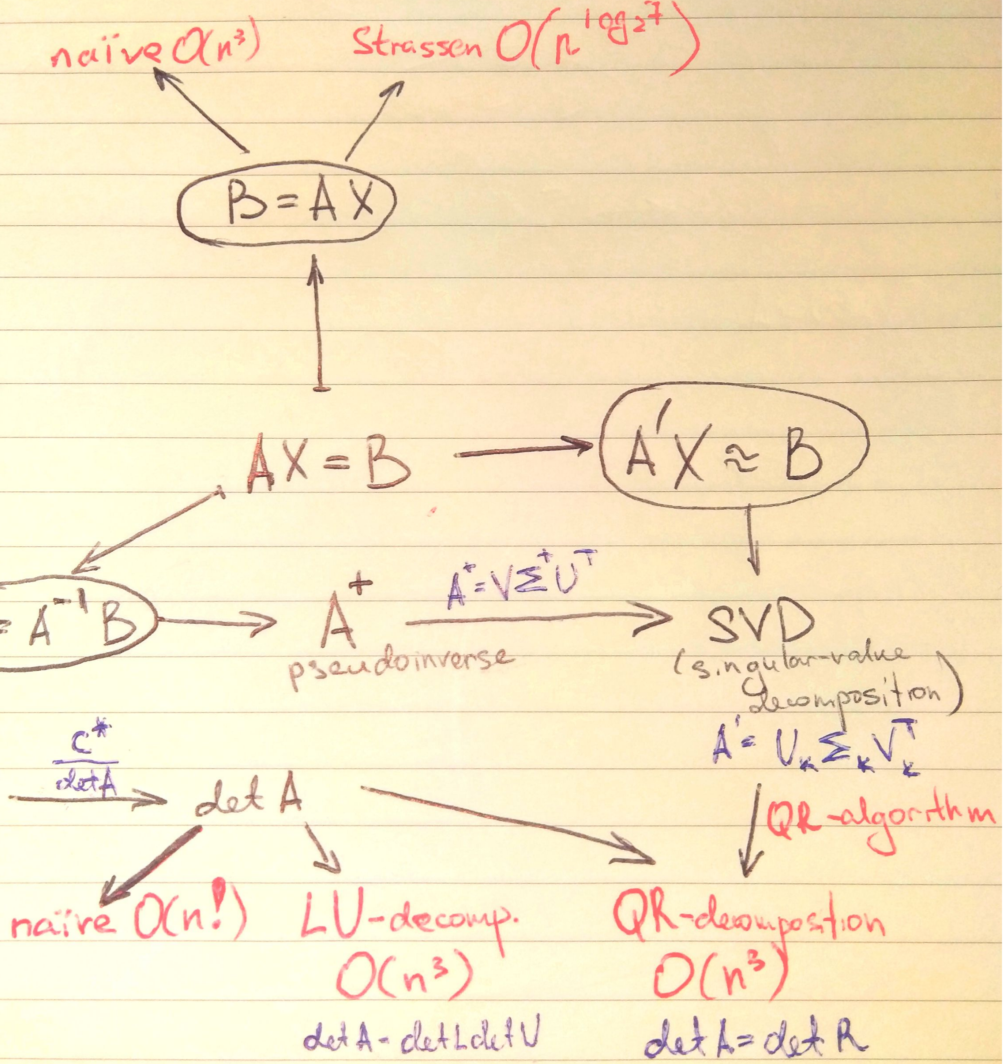
2. Repeat to get matrix ***C***

$$\begin{bmatrix} \begin{vmatrix} 3 & -1 \\ -1 & -5 \end{vmatrix} & \begin{vmatrix} -1 & 2 \\ -5 & 8 \end{vmatrix} \\ \begin{vmatrix} -1 & -5 \\ 1 & 1 \end{vmatrix} & \begin{vmatrix} -5 & 8 \\ 1 & -4 \end{vmatrix} \end{bmatrix} = \begin{bmatrix} -16 & 2 \\ 4 & 12 \end{bmatrix}.$$

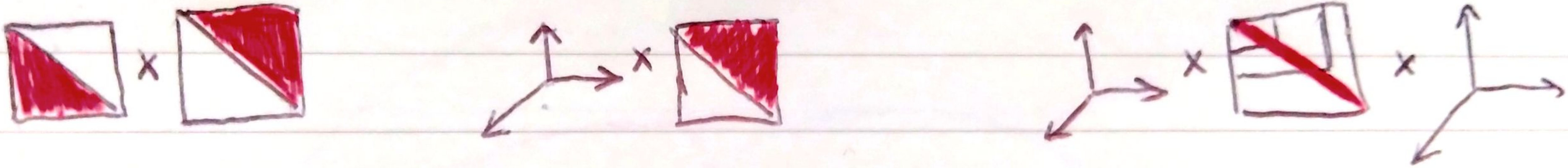
3. Divide ***C*** by internal elements of ***A***
4. ***A*=*B*, *B*=*C***. Repeat until C is ***1x1***

$$C = \begin{bmatrix} 8 & -2 \\ -4 & 6 \end{bmatrix}.$$

Decompositions


naïve $O(n^3)$
Strassen $O(n^{\log_2 7})$
$B = AX$
$AX = B$
$A'X \approx B$
$X = A^{-1}B$
A^+
pseudoinverse
$A^+ = V\Sigma^+U^T$
SVD
(singular-value decomposition)
$A' = U_k \Sigma_k V_k^T$
QR-algorithm
A^{-1}
$\frac{C^*}{\det A}$
$\det A$
naïve $O(n!)$
LU-decomp.
$O(n^3)$
$\det A = \det L \det U$
QR-decomposition
$O(n^3)$
$\det A = \det R$

LU QR SVD = $U \Sigma V^T$



LU decomposition

Lower Triangular

$$\begin{bmatrix} A00 & A01 & A02 \\ A10 & A11 & A12 \\ A20 & A21 & A22 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ L10 & 1 & 0 \\ L20 & L21 & 1 \end{bmatrix} \begin{bmatrix} U00 & U01 & U02 \\ 0 & U11 & U12 \\ 0 & 0 & U22 \end{bmatrix}$$

Upper Triangular

$$\boxed{i = 1}$$

1. $u_{1j} = a_{1j},\ j = 1 \dots n$
2. $l_{j1} = \dfrac{a_{j1}}{u_{11}},\ j = 1 \dots n\ (u_{11} \neq 0)$

$$\boxed{i = 2 \dots n}$$

1. $u_{ij} = a_{ij} - \displaystyle\sum_{k=1}^{i-1} l_{ik} u_{kj},\ j = i \dots n$
2. $l_{ji} = \dfrac{1}{u_{ii}}\left(a_{ji} - \displaystyle\sum_{k=1}^{i-1} l_{jk} u_{ki}\right),\ j = i \dots n$

QR decomposition

$$\mathrm{proj}_{\mathbf{u}}\,\mathbf{a} = \frac{\langle \mathbf{u}, \mathbf{a} \rangle}{\langle \mathbf{u}, \mathbf{u} \rangle}\mathbf{u}$$

then:

$$\begin{aligned}
\mathbf{u}_1 &= \mathbf{a}_1, & \mathbf{e}_1 &= \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|} \\
\mathbf{u}_2 &= \mathbf{a}_2 - \mathrm{proj}_{\mathbf{u}_1}\,\mathbf{a}_2, & \mathbf{e}_2 &= \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|} \\
\mathbf{u}_3 &= \mathbf{a}_3 - \mathrm{proj}_{\mathbf{u}_1}\,\mathbf{a}_3 - \mathrm{proj}_{\mathbf{u}_2}\,\mathbf{a}_3, & \mathbf{e}_3 &= \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|} \\
&\vdots & &\vdots \\
\mathbf{u}_k &= \mathbf{a}_k - \sum_{j=1}^{k-1} \mathrm{proj}_{\mathbf{u}_j}\,\mathbf{a}_k, & \mathbf{e}_k &= \frac{\mathbf{u}_k}{\|\mathbf{u}_k\|}
\end{aligned}$$

$$\boxed{Q = [\mathbf{e}_1, \cdots, \mathbf{e}_n]}$$

$$\boxed{R = \begin{pmatrix} \langle \mathbf{e}_1, \mathbf{a}_1 \rangle & \langle \mathbf{e}_1, \mathbf{a}_2 \rangle & \langle \mathbf{e}_1, \mathbf{a}_3 \rangle & \ldots \\ 0 & \langle \mathbf{e}_2, \mathbf{a}_2 \rangle & \langle \mathbf{e}_2, \mathbf{a}_3 \rangle & \ldots \\ 0 & 0 & \langle \mathbf{e}_3, \mathbf{a}_3 \rangle & \ldots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}}$$

QR algorithm

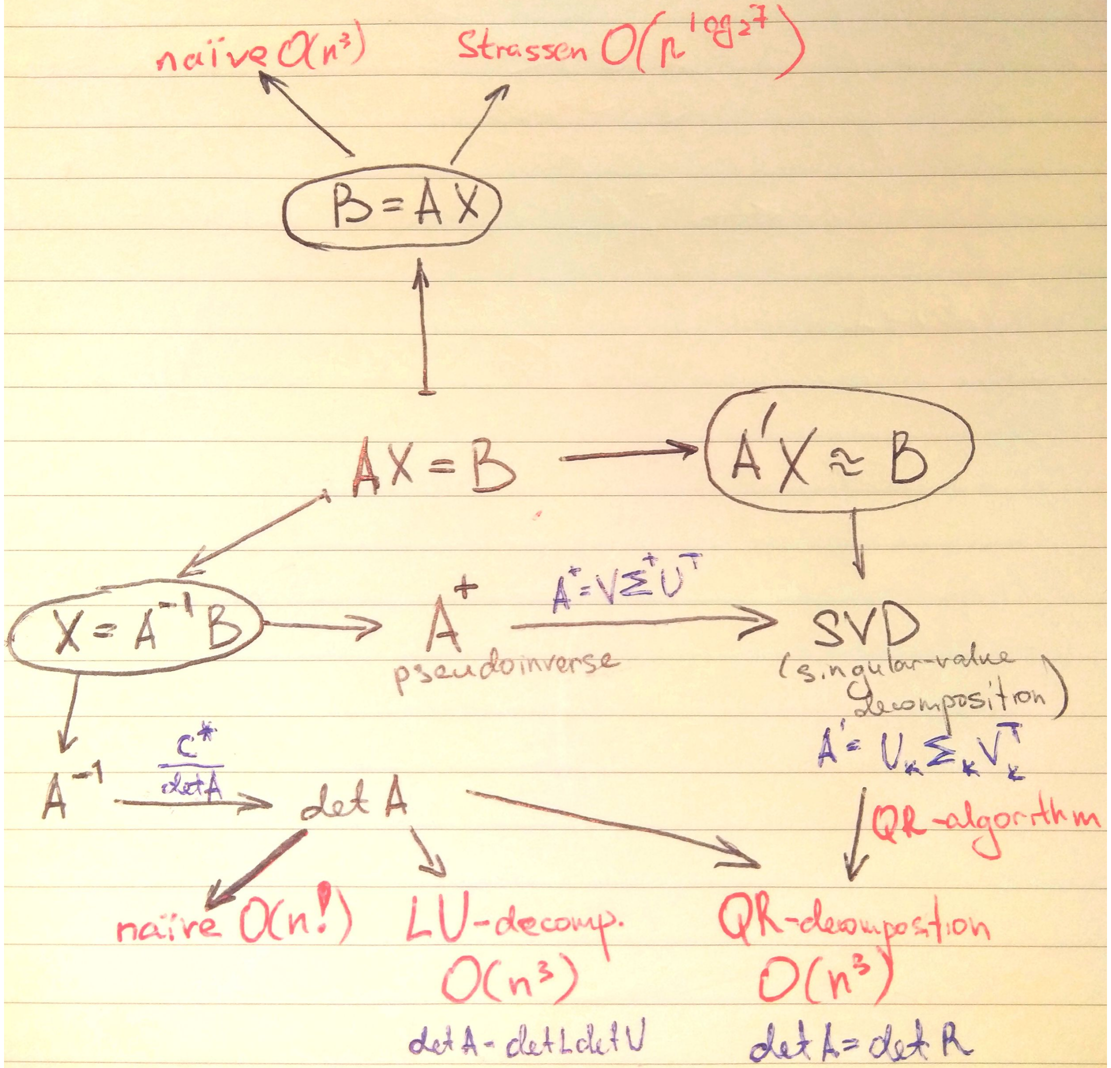
Idea is to exploit a fact, that eigenvalues of QR and RQ matrices are the same

$$A_{k+1} = R_k Q_k = Q_k^{-1} Q_k R_k Q_k = Q_k^{-1} A_k Q_k = Q_k^T A_k Q_k,$$

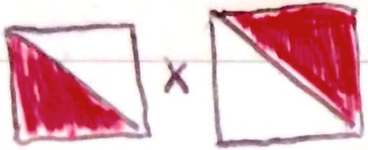
Eigenvectors of A are found in $\prod \boldsymbol{Q}_i$

Eigenvalues of A are found on diagonal of $\boldsymbol{A}_k$

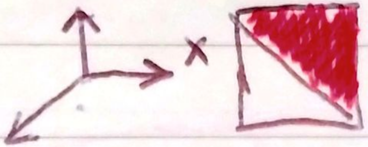
Pseudoinverse: $X=A^{+}B$


naïve $O(n^3)$
Strassen $O(n^{\log_2 7})$
$B = AX$
$AX = B$
$A'X \approx B$
$X = A^{-1}B$
A^+
pseudoinverse
$A^+ = V\Sigma^+U^T$
SVD
(singular-value decomposition)
$A' = U_k \Sigma_k V_k^T$
QR-algorithm
A^{-1}
$\frac{C^*}{\det A}$
$\det A$
naïve $O(n!)$
LU-decomp.
$O(n^3)$
$\det A = \det L \det U$
QR-decomposition
$O(n^3)$
$\det A = \det R$

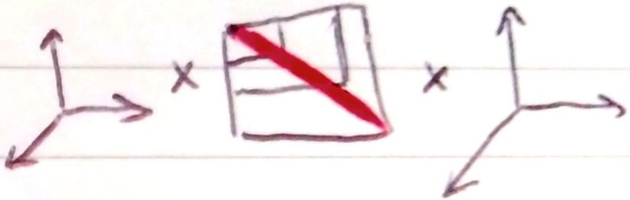
LU



QR



SVD $= U\Sigma V^T$

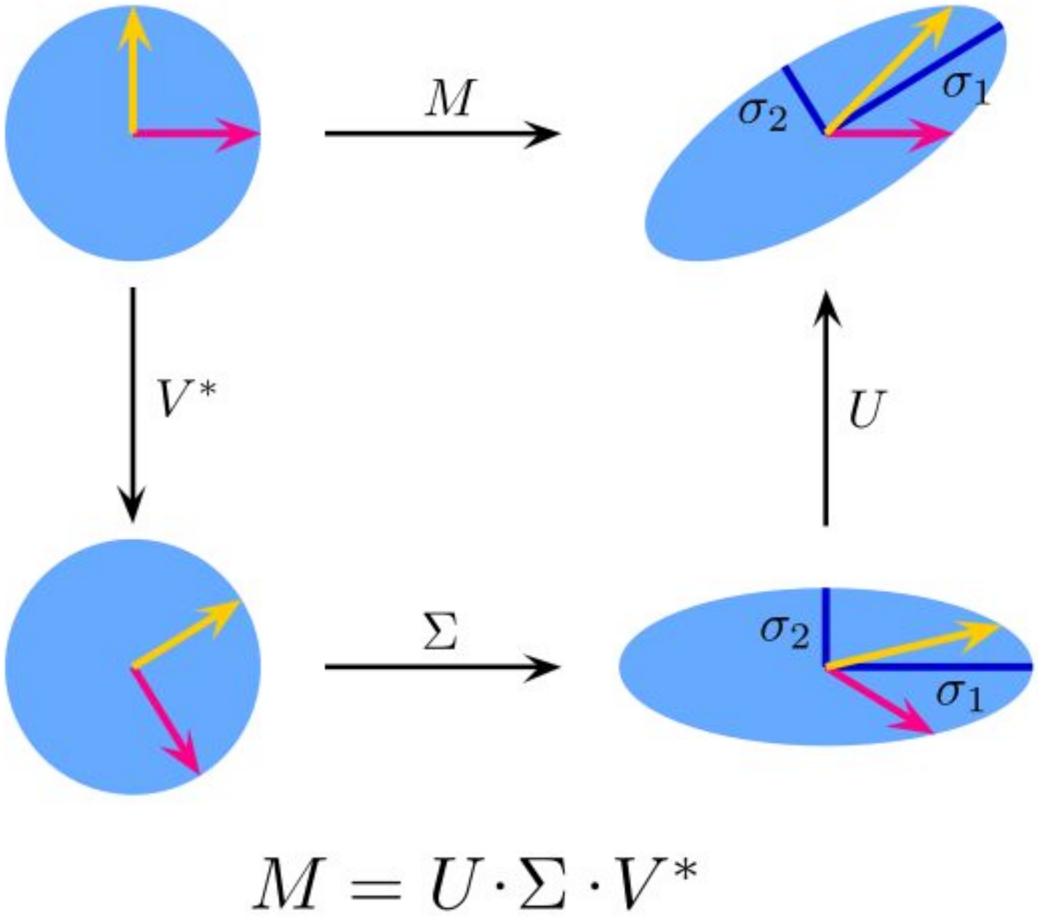


SVD

U is eigenvectors for $\mathbf{MM^T}$

$\mathbf{V^T}$ is eigenvectors for $\mathbf{M^TM}$

Σ is diagonal with square roots of non-negative eigenvalues of $\mathbf{M^TM}$



Pseudoinverse using SVD

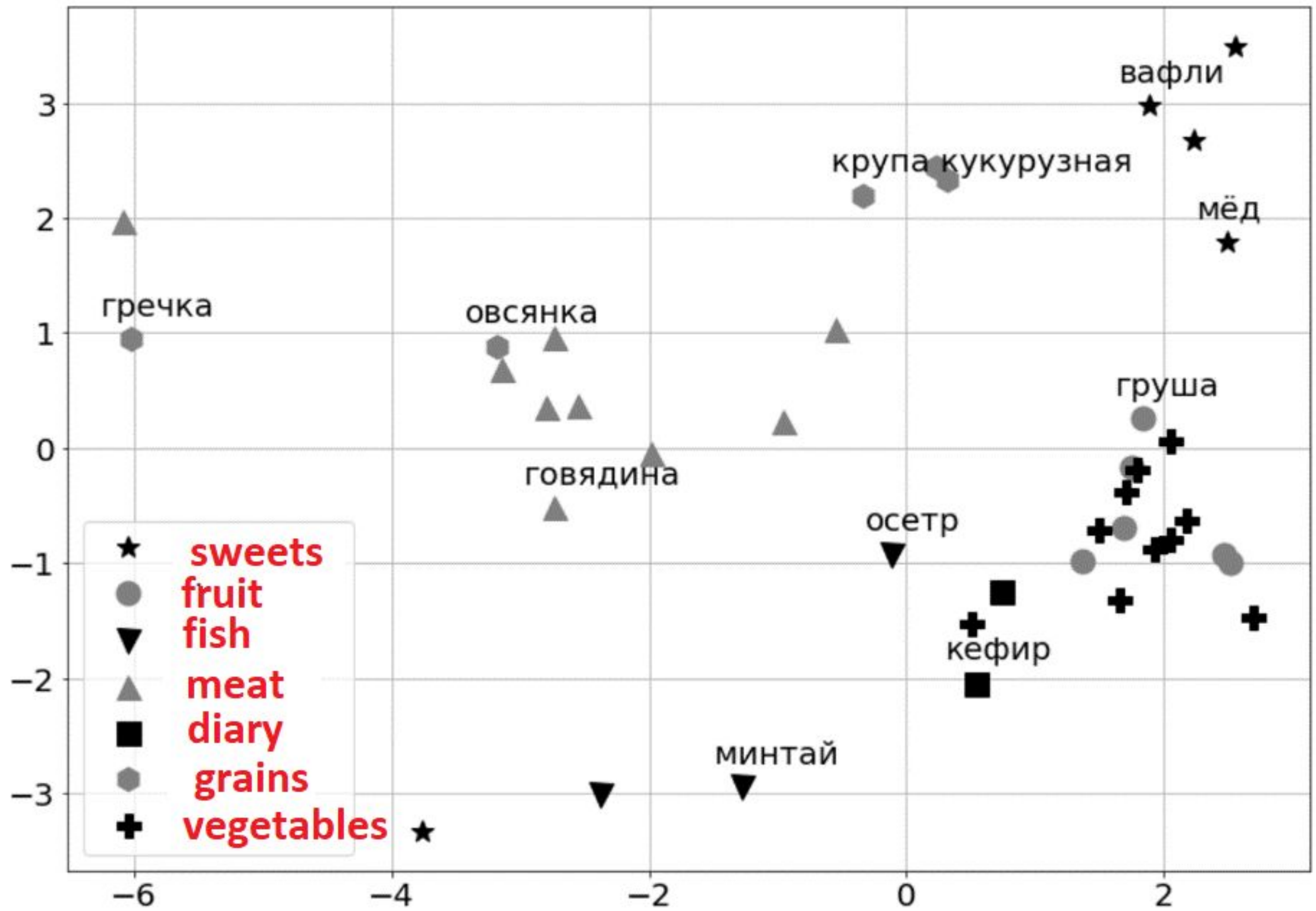
$$M = U\Sigma V^T$$

$$M+ = V\Sigma^+ U^T$$

$$\Sigma^+ \text{-?}$$

Matrix approximation

$$M = U_R \Sigma_R V_R^T$$



вафли
мёд
крупа кукурузная
гречка
овсянка
груша
говядина
осетр
кефир
минтай
sweets
fruit
fish
meat
diary
grains
vegetables
3
2
1
0
−1
−2
−3
−6
−4
−2
0
2