

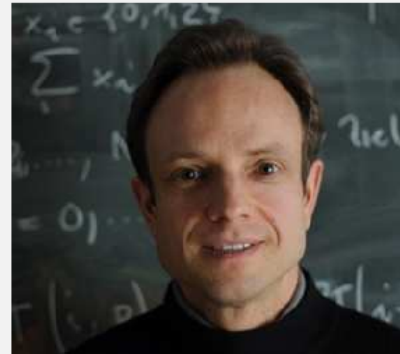
Course Video

Section 1.4: Computational Effort

Prof. R. Hiptmair, SAM, ETH Zurich

Date: August 24, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- Familiarity with basic matrix-vector operations

Dependencies. None

Duration: 55 minutes



Note: Possible minor *mismatch of video and tablet notes!*

[Corrections and updates can be incorporated into tablet notes only]

I. Computing with Matrices and Vectors

1.4. Computational Effort

Traditional: no. of elementary operations $+$, $-$, $*$, $\backslash$, $\sqrt{\quad}$

"Computational effort $\neq$ runtime"



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

Mainly determined by:

- memory access pattern
- vectorization / pipelining

1.4.1. Asymptotic complexity

Definition 1.4.1.1. (Asymptotic) complexity

The **asymptotic complexity** of an algorithm characterises the worst-case dependence of its computational effort ($\rightarrow$ Def. 1.4.0.1) on one or more **problem size parameter(s)** when these tend to ∞ .

$\text{Cost}(n)$

Numerical LA: vector length / matrix dimension

Notation: "Landau-0"

$$\text{Cost}(n) = O(n^\alpha), \quad \alpha > 0$$

for $n \rightarrow \infty$

②

$\Leftrightarrow \exists C > 0, n_0 \in \mathbb{N} : \text{cost}(n) \leq C n^\alpha \quad \forall n > n_0$

Tacit *sharpness* assumption:

$\text{cost}(n) \neq O(n^\beta), \quad \forall \beta < \alpha$

Meaning: Asymptotic complexity predicts the dependence of runtime on problem size for

- large problems
- a concrete implementation

e.g.: $\text{cost}(n) = O(n^2)$

then $n \rightarrow 2 * n$ leads to $\text{cost} \rightarrow 4 * \text{cost}$

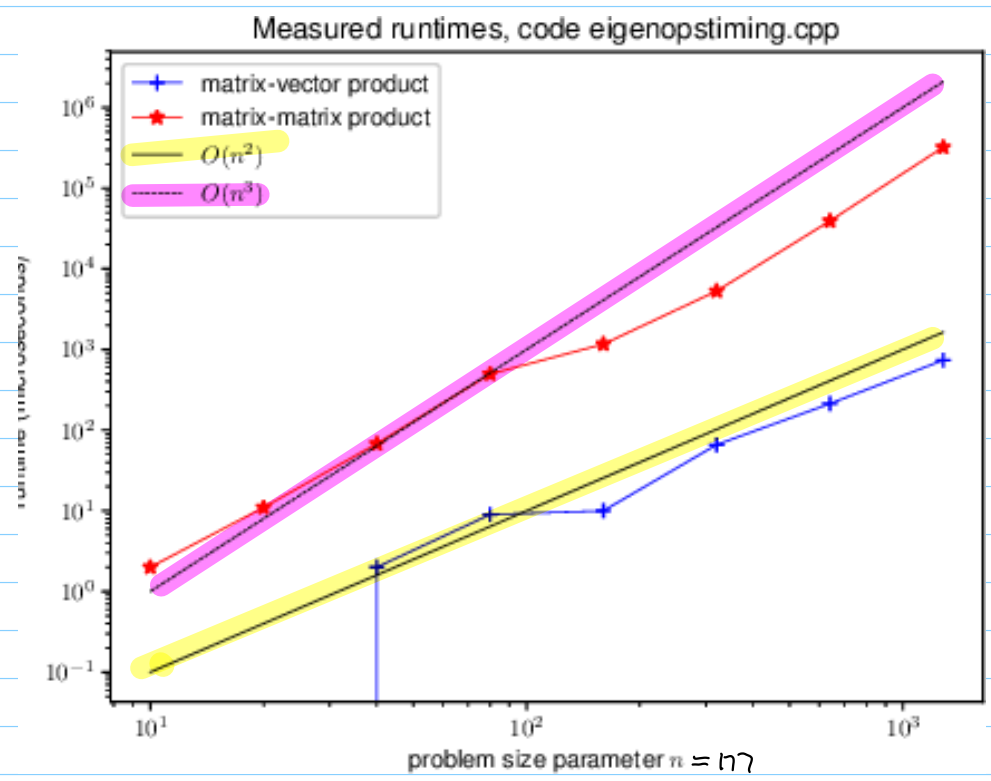
1.4.2. Computational Cost of basic numerical LA operations

operation	description	#mul/div	#add/sub	asympt. complexity
dot product	$(\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^n) \mapsto \mathbf{x}^H \mathbf{y}$	n	$n - 1$	$O(n)$
tensor product	$(\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n) \mapsto \mathbf{xy}^H$	nm	0	$O(mn)$
Matrix $\times$ vector	$(\mathbf{x} \in \mathbb{R}^n, \mathbf{A} \in \mathbb{R}^{m,n}) \mapsto \mathbf{Ax}$	mn	$(n - 1)m$	$O(mn)$
matrix product(*)	$(\mathbf{A} \in \mathbb{R}^{m,n}, \mathbf{B} \in \mathbb{R}^{n,k}) \mapsto \mathbf{AB}$	mnk	$mk(n - 1)$	$O(mnk)$

} optimal

$\hookrightarrow$ triple-loop implementation
(faster implementations available, cf Rem 1.4.2.2)

Exp. 1.4.2.1: Runtimes of M/V operation Implementation based on Eigen, $m = n$



$\rightarrow$ A doubly logarithmic plot

[If $\text{cost}(n) = O(n^d) \Rightarrow$ data points aligned in a doubly logarithmic plot]

③

1.4.3. Some Tricks to Improve Complexity

Ex 1.4.3.1: Exploit associativity

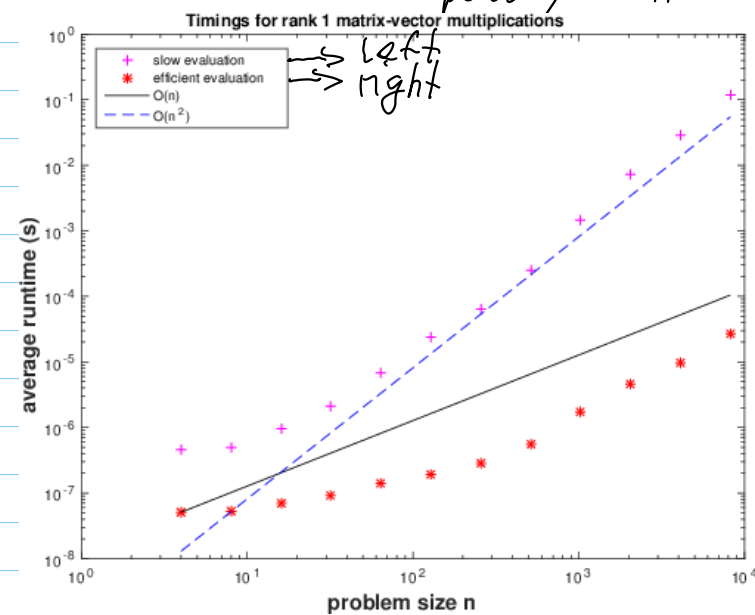
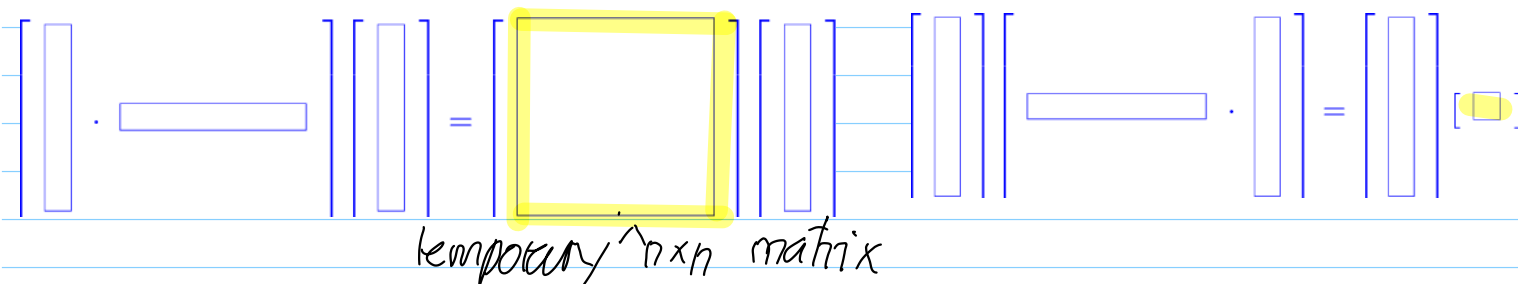
 $a, b, x \in \mathbb{R}^n$ (column vectors)

$$y = (ab^T)x. \quad (1.4.3.2)$$

$$y = a(b^T x). \quad (1.4.3.3)$$

 $T = (a*b.transpose())*x; \rightarrow O(n^2)$ $t = a*b.dot(x); \rightarrow O(n)$ 

Asymptotic cost = ?



Ex. 1.4.3.5 (Hidden summation)

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

```
Eigen::MatrixXd AB = A*B.transpose();
```

 $\rightarrow O(n^2 p)$

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

 $\rightarrow O(n^2)$
 $\rightarrow O(n^2 p)$

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

$\hookrightarrow$ upper triangular part



We can do better?

- Case $p = 1$: $A, B \leftrightarrow a, b \in \mathbb{R}^n$
 $(ab^T)_{i,j} = a_i b_j, \quad 1 \leq i, j \leq n$

Trick: Matrix factorization

$$y = \text{triu}(ab^T)x =$$

scaling matrix

$$\begin{bmatrix} a_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & a_n \end{bmatrix}$$

reverse cumulative summation

$$\begin{bmatrix} 1 & 1 & \dots & \dots & 1 \\ 0 & 1 & 1 & \dots & 1 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & 1 \end{bmatrix}$$

scaling matrix: $M \times V = O(n)$

$$\begin{bmatrix} b_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$



Efficient algorithm for $\begin{bmatrix} 1 & & 1 \\ & \ddots & \\ 0 & & 1 \end{bmatrix} x$

$$\text{Cost}(n) = O(n)$$

$$\triangleright \text{Total cost} = O(n)$$

• Case $p > 1$:

Trick: $AB^T = \sum_{\ell=1}^p \underbrace{(A)_{:, \ell} (B)_{:, \ell}^T}_{\in \mathbb{R}^{n, n}}$

triu-operation is linear: $\text{triu}(AB^T) = \sum_{\ell=1}^p \text{triu}((A)_{:, \ell} (B)_{:, \ell}^T)$

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

total cost $O(np)$

Ex. 1.4.3.8 (Reuse of intermediate results)

Definition 1.4.3.7. Kronecker product

The **Kronecker product** $A \otimes B$ of two matrices $A \in \mathbb{K}^{m,n}$ and $B \in \mathbb{K}^{l,k}$, $m, n, l, k \in \mathbb{N}$, is the $(ml) \times (nk)$ -matrix

$$A \otimes B := \begin{bmatrix} (A)_{11}B & (A)_{12}B & \dots & \dots & (A)_{1n}B \\ (A)_{21}B & (A)_{22}B & & & \vdots \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ (A)_{m1}B & (A)_{m2}B & \dots & \dots & (A)_{mn}B \end{bmatrix} \in \mathbb{K}^{ml, nk}.$$

↑ matrix-block notation

Task: Given $A, B \in \mathbb{R}^{n, n}$, $x \in \mathbb{R}^{n^2}$ we want

to compute $y = \underbrace{(A \otimes B)}_{\in \mathbb{R}^{n^2, n^2}} x$

[Naive implementation: $\text{cost} = O(n^4)$]

$$(A \otimes B)x = \begin{bmatrix} (A)_{11}Bx^1 + (A)_{12}Bx^2 + \dots + (A)_{1n}Bx^n \\ (A)_{21}Bx^1 + (A)_{22}Bx^2 + \dots + (A)_{2n}Bx^n \\ \vdots \\ \vdots \\ (A)_{m1}Bx^1 + (A)_{m2}Bx^2 + \dots + (A)_{mn}Bx^n \end{bmatrix}.$$

5 with partitioned vector $\underline{x} = \begin{bmatrix} x^1 \\ x^2 \\ \vdots \\ x^2 \end{bmatrix}$, $x_\ell \in \mathbb{R}^n$

Idea: precompute Bx^ℓ , $\ell = 1, \dots, n$

$$Z = B \cdot [\underline{x}^1, \dots, \underline{x}^n] = [Bx^1, \dots, Bx^n]$$

↓
requires $O(n^3)$ operations $\in \mathbb{R}^{n \times n}$

$$(A)_{i1} Bx^1 + (A)_{i2} Bx^2 + \dots + (A)_{in} Bx^n = Z (A)_{i,:}^T$$

$$\rightarrow \text{Form: } Y = ZA^T \rightarrow \text{cost } O(n^3)$$

Result vector

$$y = \begin{bmatrix} (Y)_{:,1} \\ \vdots \\ (Y)_{:,n} \end{bmatrix} \in \mathbb{R}^{n^2}$$

$$\text{Total cost: } O(n^3)$$

Remark: Reshaping operations at no cost!
Just a reinterpretation of arrays

C++ code 1.4.3.9: Efficient multiplication of Kronecker product with vector in EIGEN → GITLAB

```

2 template <class Matrix, class Vector>
3 Vector kronmultv(const Matrix &A, const Matrix &B, const Vector &x){
4     unsigned int m = A.rows(); unsigned int n = A.cols();
5     unsigned int l = B.rows(); unsigned int k = B.cols();
6     // 1st matrix mult. computes the products Bx^j
7     // 2nd matrix mult. combines them linearly with the coefficients of
8     Matrix t = B * Matrix::Map(x.data(), k, n) * A.transpose(); //
9     return Matrix::Map(t.data(), m*l, 1);
10 }

```

$O(n^3)$

$O(n^3)$

6 Review questions 1.4.3.11

A:

Explain why the classical concept of “computational effort” (= computational cost) is only loosely related to the runtime of a concrete implementation of an algorithm.

B:

We are given two dense matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n,p}$, $n, p \in \mathbb{N}$, $p < n$ fixed. What is the asymptotic complexity of each of the following two lines of code in terms of $n, p \rightarrow \infty$?

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

C:

Given a vector $\mathbf{u} \in \mathbb{R}^n$, $n \in \mathbb{N}$, we consider the matrix

$$\mathbf{A} \in \mathbb{R}^{n,n}: \quad (\mathbf{A})_{i,j} = u_i + u_j + u_i u_j, \quad i, j \in \{1, \dots, n\}.$$

Outline an efficient algorithm for computing the matrix-vector product $\mathbf{A}\mathbf{x}$, $\mathbf{x} \in \mathbb{R}^n$.

▷ Additional review questions can be found in the lecture document

7