Numerical Methods for Computational Science and Engineering

Fall Semester 2017 (HS17)

Prof. Rima Alaifari, SAM, ETH Zurich

1. Computing with Matrices & Vectors

1.0. Numerics & Error Analysis (cf. 1.5.2 & 1.5.4)

From int / long (discrete alg.) we move on to float / double (numerics) for comp. tasks

↳ We can no longer expect exact solutions!

Real-world quantities: $\mathbb{R}$ / $\mathbb{C}$

Computers: can't compute properly in $\mathbb{R}$

↳ Set of machine numbers $\mathbb{M}$ is finite & discrete

$$\mathbb{M} \subsetneq \mathbb{R}$$

$\mathbb{M}$ is not closed under arithmetic operations

$$op \in \{'*', '/', '+', '-'\}$$

$$op: \mathbb{R} \times \mathbb{R} \to \mathbb{R}$$

$$\mathbb{M} \times \mathbb{M} \not\to \mathbb{M}$$

```
double a = 1.0;
double b = a/9.0;
if (a==b*9.0) cout << "They are equal";
else cout <<"They are not equal";
```

$\frac{1}{9} = 0.\dot{1}$ — will print: not equal!

Instead of == queries, ask for approximate equality with some given tolerance:

```
double a = 1.0;
double b = a/9.0;
if (fabs(a-b*9.0)<numeric_limits<double>::epsilon)
cout << "They are equal";
else cout <<"They are not equal";
```

machine precision

→ introduces tradeoff between accuracy & tolerance

Fixed point representation:

fixed decimal point → most straightforward way to store frac. number

$k, l \in \mathbb{Z}$ range 10^{-k} to 10^{l}

$k+l+1$ digits k of which appear after dec. point

Pro: arithmetic op.: almost as if working with integers

e.g.: $a+b = (a \cdot 10^k + b \cdot 10^k) \cdot 10^{-k}$

Con: precision issue

e.g. $k=1$ $\quad 0.1 * 0.1 = 0.01$
$\cong 0$ (truncated)

Fixed point repr.: used in systems that favor time over accuracy (e.g. some GPU systems)

"Default"

Floating Point Representation:

in various applications: frequent change of scales

↳ requires unified representation

chemists: 10^{-31} to 10^{24}

Floating Point Representation:

$$\underbrace{0.73125}_{\text{digits of mantissa}} \cdot \underbrace{10}_{\text{base}}^{\overbrace{12}^{\text{exponent } \in \mathbb{Z}}}$$

Definition 1.5.15. Machine numbers/floating point numbers → [?, Sect. 2.1]

Given
- basis $B \in \mathbb{N} \setminus \{1\}$,
- exponent range $\{e_{\min}, \ldots, e_{\max}\}$, $e_{\min}, e_{\max} \in \mathbb{Z}$, $e_{\min} < e_{\max}$,
- number $m \in \mathbb{N}$ of digits (for mantissa),

the corresponding set of machine numbers is

$$\mathbb{M} := \{d \cdot B^E \colon d = i \cdot B^{-m}, i = B^{m-1}, \ldots, B^m - 1, E \in \{e_{\min}, \ldots, e_{\max}\}\}$$

$$x = \underbrace{\pm}_{\text{sign}} 0.\underbrace{\boxed{}\boxed{}\boxed{}\boxed{}\boxed{}\boxed{}\ldots\boxed{}\boxed{}}_{\text{mantissa}} \cdot B^{\overbrace{\boxed{}\boxed{}\boxed{}\boxed{}\ldots\boxed{}\boxed{}}^{\text{digits of exponent}}}$$

first digit: $\neq 0$

Machine numbers $\mathbb{M}$: standard for floating pt repr.:

IEEE 754: 5 basic formats

- 3 binary formats
  - binary32 : single
  - binary64 : double
  - binary128 : quadruple

  most common

- 2 decimal formats
  - decimal64 : double
  - decimal128 : quadr.

C++11 code 1.5.21: Querying characteristics of `double` numbers → GITLAB

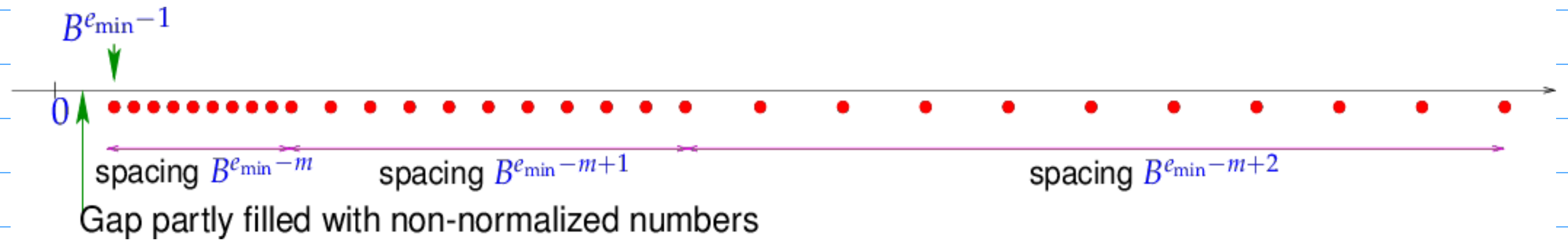
```cpp
#include <limits>
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
    cout << std::numeric_limits<double>::is_iec559 << endl
    << std::defaultfloat << numeric_limits<double>::min() << endl
    << std::hexfloat << numeric_limits<double>::min() << endl
    << std::defaultfloat << numeric_limits<double>::max() << endl
    << std::hexfloat << numeric_limits<double>::max() << endl;
}
```

Output:

```
true
2.22507e-308
0010000000000000
1.79769e+308
7fefffffffffffff
```

Note: Machine numbers are not evenly spaced

gaps are bigger for larger numbers



Recall: $op : \mathbb{M} \times \mathbb{M} \not\to \mathbb{M}$

On computer $\widetilde{op} : \mathbb{M} \times \mathbb{M} \to \mathbb{M}$

implementation: $\widetilde{op} = rd \circ op$

↳ rounding to nearest machine number

Definition 1.5.27. Correct rounding

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

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

Error analysis:

C++11 code 1.5.23: Demonstration of roundoff errors ➺ GITLAB

```cpp
#include <iostream>
int main() {
    std::cout.precision(15);
    double a = 4.0/3.0, b = a-1, c = 3*b, e = 1-c;
    std::cout << e << std::endl;
    a = 1012.0/113.0; b = a-9; c = 113*b; e = 5+c;
    std::cout << e << std::endl;
    a = 83810206.0/6789.0; b = a-12345; c = 6789*b; e = c-1;
    std::cout << e << std::endl;
}
```

Output:

```
2.22044604925031e-16
6.75015598972095e-14
-1.60798663273454e-09
```

Absolute & relative error:

Definition 1.5.24. Absolute and relative error → [?, Sect. 1.2]

Let $\tilde{x} \in \mathbb{K}$ be an approximation of $x \in \mathbb{K}$. Then its absolute error is given by

$$\epsilon_{\text{abs}} := |x - \tilde{x}| \, ,$$

and its relative error is defined as

$$\epsilon_{\text{rel}} := \frac{|x - \tilde{x}|}{|x|} \, .$$

Approximation $\tilde{x}$ of x has $\ell \in \mathbb{N}_0$ correct digits if

$$\varepsilon_{rel} := \frac{|x-\tilde{x}|}{|x|} \leq 10^{-\ell}$$

Maximal relative error of rounding:

$$\mathrm{EPS} = \max_{x \in \mathbb{R}} \frac{|rd(x) - x|}{|x|}$$

machine precision

Assumption 1.5.32. "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* $f \in \{\exp, \sin, \cos, \log, \dots\}$ 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 $|\delta| < \mathrm{EPS}$.

Querying EPS in C++:

C++11-code 1.5.34: Finding out EPS in C++ ➩ GITLAB

```cpp
#include <iostream>
#include <limits> // get various properties of arithmetic types
int main() {
  std::cout.precision(15);
  std::cout << std::numeric_limits<double>::epsilon() << std::endl;
}
```

Output:

```
2.22044604925031e-16
```

Alternative definition:

smallest possible pos. number s.t.

$$1 \tilde{+} \mathrm{EPS} \neq 1 \qquad (\text{note: e.g. } 10 \tilde{+} \mathrm{EPS} = 10)$$

Note: other sources of errors exist such as:

discretization error
modelling error
measurement error
etc.

Note: Rel. or abs. error: in general not computable

(don't know $\underline{\underline{\text{true}}}$ solution!)

1. worst case estimates

2. Compute backward error:

Suppose $Ax_{ex} = b$ (solve for x)

compute x_{app} (appr. of x_{ex})

$b_{app} := A x_{app}$

compare b_{app} to b

$x_{ex} - x_{app}$ forward error

$b - b_{app}$ backward error $\Leftarrow$ CAN COMPUTE THIS!

In practice: Stop when $A x_{app} - b$ is small

However: A small backward error $\not\Rightarrow$ small forward error!

x_{app} can be far off from x_{ex}!

Condition number   Example matrix A

$\frac{\sigma_{max}}{\sigma_{min}}$ ratio of largest & smallest sing. value

Recall toy example of 3×3 matrix

$b - b^{\delta} = 0.007$ $x - x^{\delta}$ huge

cond. number: $\underline{\underline{10^5}}$

1.1 Fundamentals

1.1.1 Notation

$$A := \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \\ \vdots & & \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \in \mathbb{K}^{m,n}$$

$\mathbb{K}$: $\mathbb{R}/\mathbb{C}$

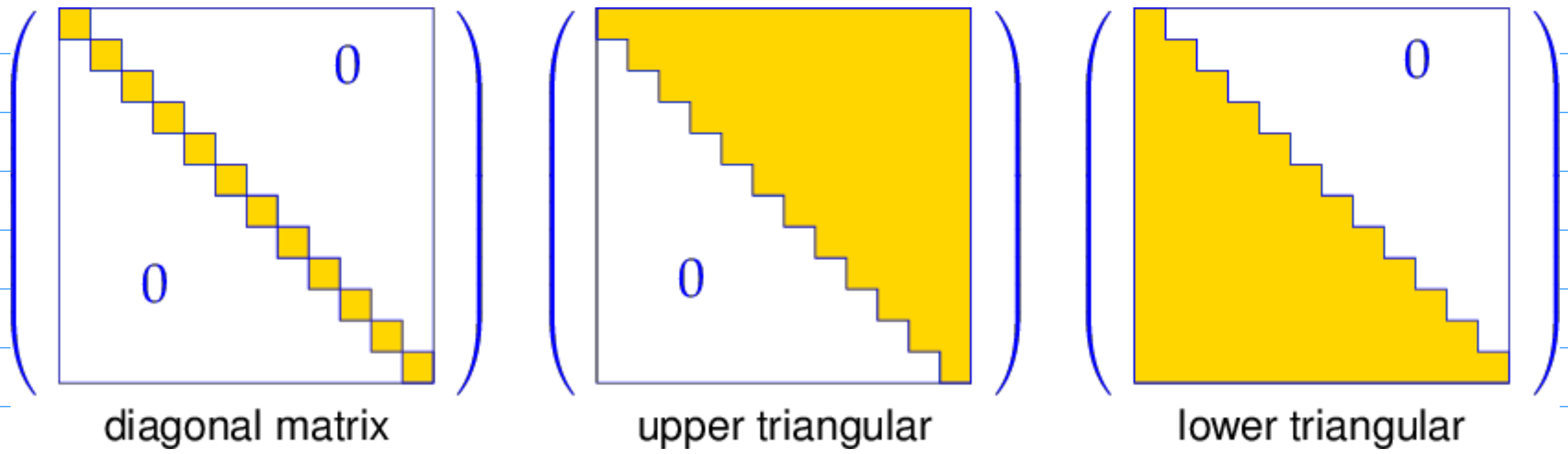
$(A)_{i,j} = a_{ij}$ $i \in \{1, \dots, m\}$, $j \in \{1, \dots, n\}$

$a_{i,:} = (A)_{i,:}$ i-th row

$a_{:,j} = (A)_{:,j}$ j-th column

$(a)_{\substack{i=k,\dots,l \\ j=r,\dots,s}} = (A)_{k:l,\, r:s}$ $1 \le k \le l \le m$, $1 \le r \le s \le n$ submatrix

Matrix types:



Transpose of A: A^T

Adjoint of A: $A^H = \begin{bmatrix} \overline{a_{11}} & \dots & \overline{a_{m1}} \\ \vdots & & \\ \overline{a_{1n}} & \dots & \overline{a_{mn}} \end{bmatrix}$

$A \in \mathbb{R}^{m,n}$: $A^T = A^H$

Symmetric: $A^T = A$

Hermitian: $A^H = A$

Definition (s.p.d. matrix):

The matrix $A \in \mathbb{K}^{n,n}$, $n \in \mathbb{N}$, is symm. (Hermitian) (semi-) pos.-def. (s.p.d.) if

$A = A^H$ and $\forall x \in \mathbb{K}^n$: $x^H A x \in \mathbb{R}$ and

$x^H A x > 0$ $(\geq)$ $\Leftrightarrow$ $x \neq 0$

A matrix $A \in \mathbb{K}^{n,n}$ is (semi-) s.p.d. iff all its eigenvalues are positive. (non-neg.)

1.2. Libraries

1.2.1 EIGEN

header-only C++ library for numerical computations

provides
- data structures
- (standard) operations [on matrices/vectors]

fundamental data type: matrix

Matrix < typename scalar, int nrows, int ncols >

type of coeffs [double, long, ...]

fixed dynamic

Example: Matrix<double, Dynamic, Dynamic>

→ size not known at compile time but treated as runtime variable

convenience typedefs:

MatrixXd double

dynamic

Matrix3f

fixed size 3x3

Matrix3d x; ← 3x3 matrix with array of uninitialized coefficients

MatrixXf y; ← dynamic size current size 0-by-0 (no array of coeffs allocated)

MatrixXf y(6,9);

VectorXd x(12); dynamic size

array of coeffs. allocated with given size

x.resize(5);

C++11 code 1.2.14: Initializing special matrices in EIGEN

```cpp
#include <Eigen/Dense>
// Just allocate space for matrix, no initialisation
Eigen::MatrixXd A(rows,cols);
// Zero matrix. Similar to matlab command zeros(rows, cols);
Eigen::MatrixXd B = MatrixXd::Zero(rows, cols);
// Ones matrix. Similar to matlab command ones(rows, cols);
Eigen::MatrixXd C = MatrixXd::Ones(rows, cols);
// Matrix with all entries same as value.
Eigen::MatrixXd D = MatrixXd::Constant(rows, cols, value);
// Random matrix, entries uniformly distributed in [0,1]
Eigen::MatrixXd E = MatrixXd::Random(rows, cols);
// (Generalized) identity matrix, 1 on main diagonal
Eigen::MatrixXd I = MatrixXd::Identity(rows,cols);
std::cout << "size of A = (" << A.rows() << ',' << A.cols() << ')' <<
    std::endl;
```

1.2.4. Dense Matrix Storage Formats

$A \in \mathbb{K}^{m,n}$: stored as array of length $m \cdot n$

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Row major (C-arrays, bitmaps, Python):

A_arr	1	2	3	4	5	6	7	8	9

Column major (Fortran, MATLAB, EIGEN):

A_arr	1	4	7	2	5	8	3	6	9

Indexing in EIGEN start 0!

To access coeffs $A(\text{int } a, \text{int } b)$

$v(\text{int } a)$

$A(\text{int } a)$

$A(4) = 5$

format : column major default

→ can be changed

C++11 code 1.2.22: Single index access of matrix entries in EIGEN ➜ GITLAB

```cpp
void storageOrder(int nrows=6,int ncols=7)
{
  cout << "Different matrix storage layouts in Eigen" << endl;
  // Template parameter ColMajor selects column major data layout
  Matrix<double,Dynamic,Dynamic,ColMajor> mcm(nrows,ncols);
  // Template parameter RowMajor selects row major data layout
  Matrix<double,Dynamic,Dynamic,RowMajor> mrm(nrows,ncols);
  // Direct initialization; lazy option:  use int as index type
  for (int l=1,i= 0; i< nrows; i++)
    for (int j= 0; j< ncols; j++,l++)
      mcm(i,j) = mrm(i,j) = l;

  cout << "Matrix mrm = " << endl << mrm << endl;
  cout << "mcm linear = ";
  for (int l=0;l < mcm.size(); l++) cout << mcm(l) << ',';
  cout << endl;

  cout << "mrm linear = ";
  for (int l=0;l < mrm.size(); l++) cout << mrm(l) << ',';
  cout << endl;
}
```

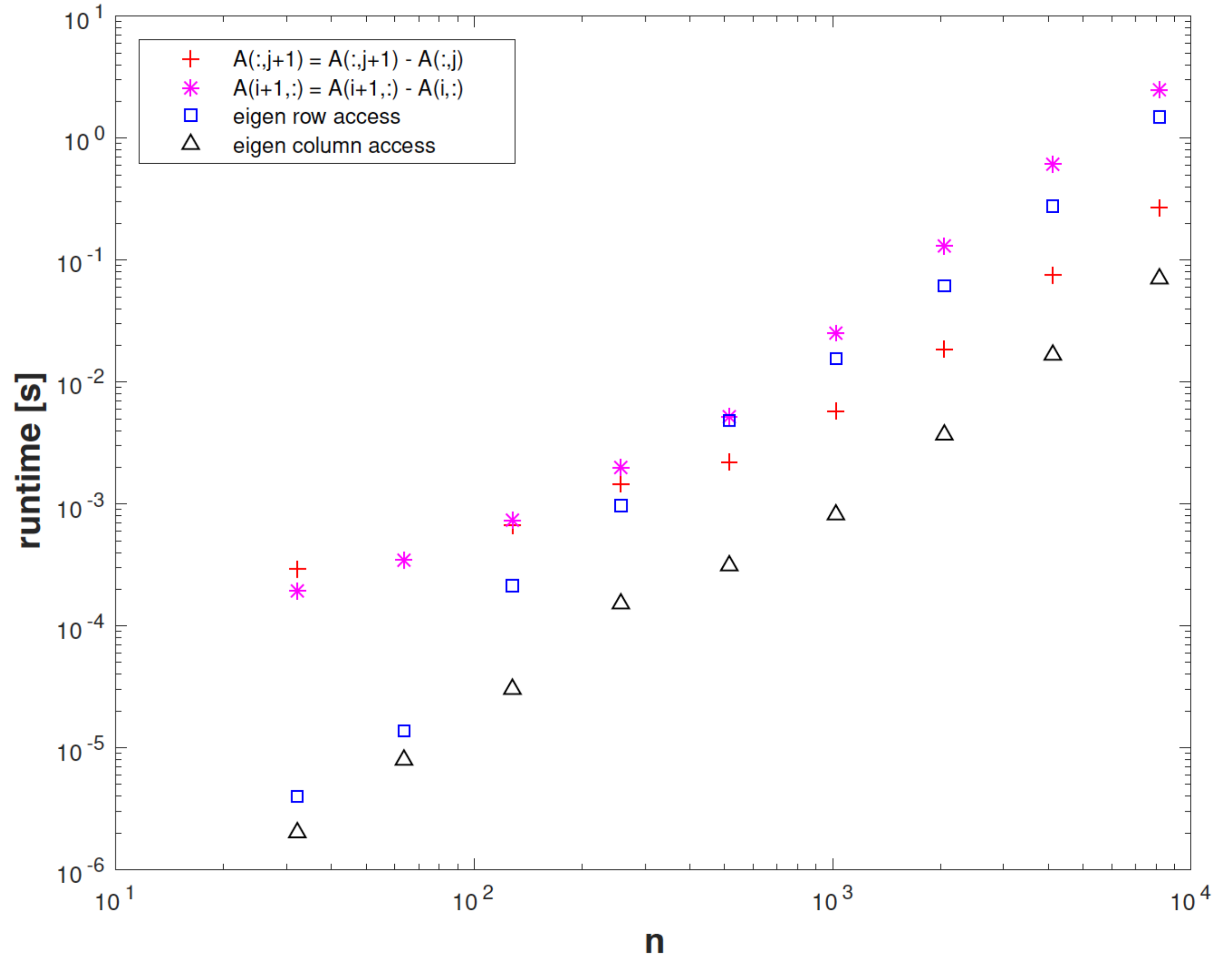
```
Different matrix storage layouts in Eigen
Matrix mrm =
1 2 3
4 5 6
7 8 9
mcm linear = 1,4,7,2,5,8,3,6,9,
mrm linear = 1,2,3,4,5,6,7,8,9,
```

Data storage format impacts runtime

Example: row / column-wise access of matrix

$$A.row(j+1) -= A.row(j)$$

vs. $$A.col(j+1) -= A.col(j)$$



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.

"Computational effort $\not\sim$ runtime"

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

out of scope of this course:
- parallelization / vectorization of execution
- memory hierarchies (organization of algorithm accordingly)
- optimizing code wrt hardware resources

1.4.1 Asymptotic complexity

How does computational effort <u>scale</u> with the problem size?

↳ tells us something about performance & comparison of algorithms

Definition 1.4.4. (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 ∞.

typical parameter: dimension of input of vector/matrix

worst case: maximal effort over set of admissible inputs

Landau O-notation

Consider $f, g: \mathbb{N} \to \mathbb{R}$

We write $f(n) = O(g(n))$

if $\exists\, C > 0,\ n_* \in \mathbb{N}$ s.t.

$$\forall n \geq n_*: \quad f(n) \leq C \cdot g(n)$$

implicit assumption: sharpness of Θ-bound
valid provable

asymptotic complexity: Does not predict runtime!

BUT: predicts dependence of runtime on size of the problem

Ex.: $\text{cost}(n) = \Theta(n^2)$

if n is doubled, complexity increases by $*4$

Ex.: Conjecture: $t_i \approx C \cdot n_i^{\alpha}$

t_i: runtimes

n_i: problem sizes

$i = 1, \ldots, N$

log-log-plot: $\log t_i \approx \log C + \alpha \log n_i$

data points (t_i, n_i) lie roughly on straight line with slope α

1.4.2. Cost of basic operations

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	$n-1$	$O(n)$
tensor product	$(\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n) \mapsto \mathbf{x}\mathbf{y}^H$	nm	0	$O(mn)$
matrix product(*)	$(\mathbf{A} \in \mathbb{R}^{m,n}, \mathbf{B} \in \mathbb{R}^{n,k}) \mapsto \mathbf{AB}$	mnk	$mk(n-1)$	$O(mnk)$

1.4.3. Some tricks to reduce complexity

1. Example: Rank-1-matrix $B \in \mathbb{K}^{m,n}$

Compute $y = Bx$

Note: B can be written as

$$B = a b^T \qquad a \in \mathbb{K}^m, \; b \in \mathbb{K}^n$$

$$y = a b^T x \qquad b^T x \quad \text{scalar}$$

$$\underbrace{a(\underbrace{b^T x}_{\mathcal{O}(n)})}_{\mathcal{O}(m)} \Big\} \; \mathcal{O}(m+n) \qquad \mathcal{O}(n)$$