Mathematical Software Programming (02635)

Lecture 1 — September 4, 2025

Instructor: Martin S. Andersen

Fall 2025

About me

Martin S. Andersen

- Associate Professor at DTU Compute, Section for Scientific Computing
- PhD from the University of California, Los Angeles
- MSc Eng from Aalborg University
- Research interests
  - optimization (I also teach *02611 Optimization for data science*)
  - numerical linear algebra
  - applications (signal and image processing, inverse problems, machine learning, control, ...)
- Co-developer of several mathematical software packages
- My preferred programming languages are ... (but I dislike that ...)
  - Python (Numpy arrays are awkward for linear algebra)
  - Julia (plotting can be slow)
  - MATLAB (it is closed-source and clunky)
  - C (I have to do *everything*)

Practical information

Format

- 5 ECTS (1 ECTS $\sim$ 28 hours on average)
- Lectures and exercises
- Weekly reading assignments
- Autolab exercises (10%) and one assignment (15%)
- Final exam (75%) — written exam, Dec. 9, 2025

Instructors

- Martin S. Andersen (`mskan`), DTU Compute
- Bernd Dammann (`beda`), DTU Compute/DCC

Teaching assistants

- Jonas Amtoft (s234948)
- Paula Barho (s242926)

Help!?

Instructors/teaching assistants

- Be prepared
- Write down questions
- Get feedback

DTU Learn

- Post your (anonymous) questions on the DTU Learn discussion board
- Learn from and help your peers
- Enable notifications (click on “Notifications” and change “How often?”)

Email

- Please use email for personal matters only

Mathematical software

- Computer representation of numbers
- Finite precision arithmetic
- Conditioning of a problem
- Stability of an algorithm
- Data structures for mathematical objects (polynomials, vectors, matrices, . . .)
- Time/space complexity
- Memory hierarchy
- Parallel computing
- Recursive functions

Example 1: matrix chain multiplication (MATLAB)

- matrix multiplication is associative: $A(BC) = (AB)C$
- many ways to compute $y = UU^Tx$ ($U \in \mathbb{R}^{n\times k}$ and $x \in \mathbb{R}^n$)

```
n = 8000; k = 100;
U = randn(n,k);
x = randn(n,1);
tic; y1 = U*U'*x; toc      % version 1 (left-to-right)
tic; y2 = U*(U'*x); toc    % version 2
fprintf(1,'norm(y1-y2) = %.2e\n', norm(y1-y2))
```

```
Elapsed time is 0.448834 seconds.
Elapsed time is 0.001667 seconds.
norm(y1-y2) = 2.70e-10
```

- $y_2 = U(U^Tx)$ is more than 250 times faster than MATLAB's default $y_1 = (UU^T)x$
- the results are different (i.e., $\|y_1 - y_2\|_2 \neq 0$)

Example 2: finite precision arithmetic

Evaluate the following expressions

$$10^{12} - (10^6 - 7)(10^6 + 7) \qquad\qquad 3 \cdot 0.1 - 0.3$$

Maple

$1e12 - (1e6 - 7) \cdot (1e6 + 7)$

```
0.
```

$3 \cdot 0.1 - 0.3$

```
0.
```

Python

```
>>> 1e12 - (1e6-7)*(1e6+7)
49.0
>>> 3*0.1-0.3
5.551115123125783e-17
```

Example 3: sparse vectors and matrices (MATLAB)

Storage-efficient representation of vectors and matrices with few nonzero elements

```
%  Unit vector v = (1,0,...,0) of length 4,000,000,000
v = sparse([1],[1],1.0,4e9,1);    % v is a "column" vector
whos
u = v';       % the transpose of v is a "row" vector
```

```
  Name                  Size                Bytes  Class     Attributes

  v            4000000000x1                    32  double    sparse

Error using  '
Requested 1x4000000000 (29.8GB) array exceeds maximum array
size preference (16.0GB).
```

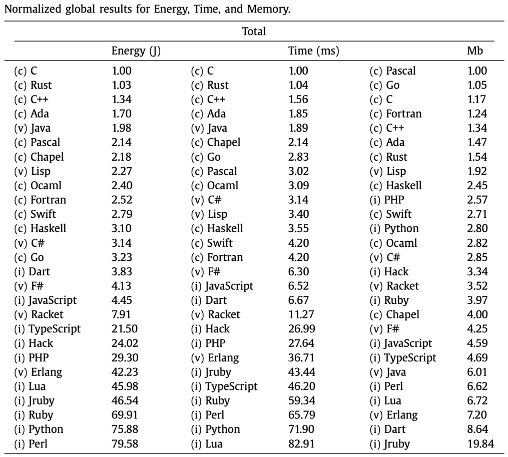
Learning objectives

- Evaluate discrete and continuous mathematical expressions.
- Describe and use data structures such as lists, arrays, and sparse matrices.
- Choose appropriate data types and data structures for a given problem.
- Compare iterative and recursive solutions for simple problems.
- Analyze the runtime behavior and the time and space complexity of simple programs.
- Call external (third party) programs and libraries.
- Design, implement, and document a program that solves a mathematical problem.
- Debug and test mathematical software.
- Describe and use basic object-oriented programming concepts such as classes and objects.
- Explain rounding errors and floating point number representation of real numbers.

Why C?

- Widely used and mature programming language (developed in the early 1970s)
- Industry standard (ANSI C (C89) / ISO C (C90), C95, C99, C11, C17, C23)
- Many newer programming languages are syntactically similar to C (e.g., C++, C#, Objective C, Java, PHP, Go, ...)
- Cross-platform support
- Low-level control (direct access to low level hardware/APIs)
- Low overhead (high performance)
- Statically typed language
- Understanding of memory management (no "magic" under the hood)
- Embedded systems (IoT)
- C *powers* the world (OS kernels, Python, MATLAB, ...)

Fast and efficient



Normalized global results for Energy, Time, and Memory.

Total					
	Energy (J)		Time (ms)		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

Pereira et al. (2021) (DOI: 10.1016/j.scico.2021.102609)

Fast and efficient (cont.)

ChatGPT: *Compare runtime of a matrix-vector multiplication routine in C and in pure Python.*

```
Elapsed time in C: 5.53 ms
Elapsed time in Python: 296.21 ms
```

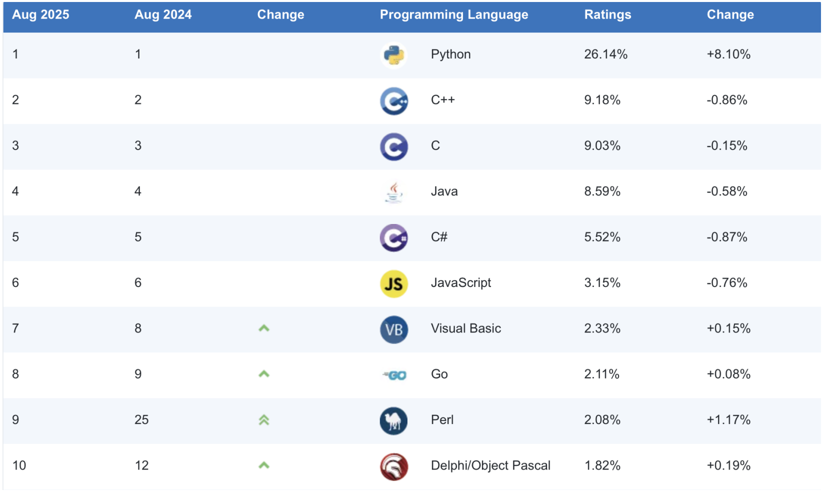
ChatGPT: *Make a version of the Python code that uses NumPy.*

```
Elapsed time in Python with NumPy: 1.41 ms
```

ChatGPT: *Make a version of the C code that uses BLAS.*

```
Elapsed time in C with BLAS: 1.32 ms
```

TIOBE Index August 2025



Aug 2025	Aug 2024	Change	Programming Language	Ratings	Change
1	1		Python	26.14%	+8.10%
2	2		C++	9.18%	-0.86%
3	3		C	9.03%	-0.15%
4	4		Java	8.59%	-0.58%
5	5		C#	5.52%	-0.87%
6	6		JavaScript	3.15%	-0.76%
7	8	⌃	Visual Basic	2.33%	+0.15%
8	9	⌃	Go	2.11%	+0.08%
9	25	⌃⌃	Perl	2.08%	+1.17%
10	12	⌃	Delphi/Object Pascal	1.82%	+0.19%

Resources

Textbooks

- K. N. King, “C Programming: A Modern Approach”, 2. edition, 2008 (ISBN: 9780393979503). Available at Polyteknisk boghandel.

Supplementary resources (optional)

- I. Horton, “Beginning C”, 5th ed., 2013 (ISBN: 9781430248811)
  - E-book available through DTU Library
  - Source code available for examples
- M. Olsson, C quick syntax reference, 2015
- I. Horton, Beginning C++, 2014
- M. Olsson, C++ quick syntax reference, 2013
- OnlineProgrammingBooks.com
- Big-O Cheat Sheet
- Learn to Solve It: C programming exercises

Documentation and reference manuals

- DevDocs C documentation
- GNU C Library
- GNU C Library - function index
- GNU Compiler Collection (GCC) Manual
- Wikipedia: C mathematical functions
- GNU Scientific Library
- Cplusplus.com
- Cprogramming.com
- Boost C++ Library

Compilers

Compiler installation guide available on DTU Learn

- Linux/Unix
  - GCC (Ubuntu/Debian: `sudo apt-get install build-essential gdb`)
  - `clang` (`sudo apt-get install clang`)
- Mac OS X
  - Clang (`xcode-select --install`)
  - GCC (e.g., via Homebrew)
- Windows
  - GCC via Windows Subsystem for Linux (WSL)
  - GCC via MSYS2
  - Visual Studio C++ (no support)

Software

Cross-platform editors & IDEs

- Visual Studio Code
- Atom
- GNU Emacs
- Vim
- Eclipse

Tools

- GNU Make
- GNU debugger
- GNU profiler
- Valgrind profiler

DTU Resources

- DTU Computing Center
- gBar DataBar
- gBar GitLab

Historical perspective

C89	C99	C11	C18
Intel 80486 $350 0.03 GFLOPS	Intel Pentium III $800 1-2 GFLOPS	Intel Core i7 (1st gen) $600 80 GFLOPS	Intel Core i9 7980XE $2000 950 GFLOPS
Macintosh Portable $6,500 No FPU	PS2 $299 6 GFLOPS	iPhone 4s $650 12 GFLOPS	iPhone 8 $699 300 GFLOPS

- Clang 7.0 and GCC 8.1 support C18
- Clang 9.0 and GCC 9.0 have experimental support for C2x
- Microsoft Visual Studio 2022
  "MSVC is compatible with the ANSI C89 and ISO C99 standards, but not strictly conforming." (VS 2022)

Today's exercises

Available on DTU Learn:

- Part I: install a C compiler and a text editor
- Part II: do the exercises (individually or in small groups)

If you finish early, start preparing for next week!

Numeral systems

Positional notation for a number x

$$x = s\,(c_{n-1}c_{n-2}\cdots c_0.d_1d_2\cdots)_b$$

- $s \in \{-1,+1\}$ represents the sign
- digits $c_{n-1},\dots,c_0,d_1,d_2,\dots$
- radix point "."
- the base (aka radix) b is an integer (≥ 2)
  - decimal numbers ($b = 10$)
  - binary numbers ($b = 2$)
  - octal numbers ($b = 8$)
  - hexadecimal numbers ($b = 16$)
- classification: terminating, non-terminating, and repeating (depends on base)

Expanded form of x

$$x = s\left(\underbrace{\sum_{j=0}^{n-1} c_j \cdot b^j}_{\text{integral part}} + \underbrace{\sum_{j=1}^{\infty} d_j \cdot b^{-j}}_{\text{fractional part}}\right)$$

Numeral systems (continued)

- decimal digits: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- binary digits: $\{0, 1\}$
- octal digits: $\{0, 1, 2, 3, 4, 5, 6, 7\}$
- hexadecimal digits: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}, \mathsf{f}\}$

Number	Decimal ($b = 10$)	Binary ($b = 2$)	Hexadec. ($b = 16$)
18	18 or $17.\overline{9}$	10010 or $10001.\overline{1}$	12 or $11.\overline{\mathrm{f}}$
11	11 or $10.\overline{9}$	1011 or $1010.\overline{1}$	b or $\mathrm{a}.\overline{\mathrm{f}}$
4	4 or $3.\overline{9}$	100 or $11.\overline{1}$	4 or $3.\overline{\mathrm{f}}$
1	1 or $0.\overline{9}$	1 or $0.\overline{1}$	1 or $0.\overline{\mathrm{f}}$
$1/3$	$0.\overline{3}$	$0.\overline{01}$	$0.\overline{5}$
$1/4$	0.25 or $0.24\overline{9}$	0.01 or $0.00\overline{1}$	0.4 or $0.3\overline{\mathrm{f}}$
$1/10$	0.1 or $0.0\overline{9}$	$0.0\overline{0011}$	$0.1\overline{9}$
$-1/2$	-0.5 or $-0.4\overline{9}$	-0.1 or $-0.0\overline{1}$	-0.8 or $-0.7\overline{\mathrm{f}}$

Compile and run “Hello 02635!” program

Create a plain text file `hello.c` with the following code:

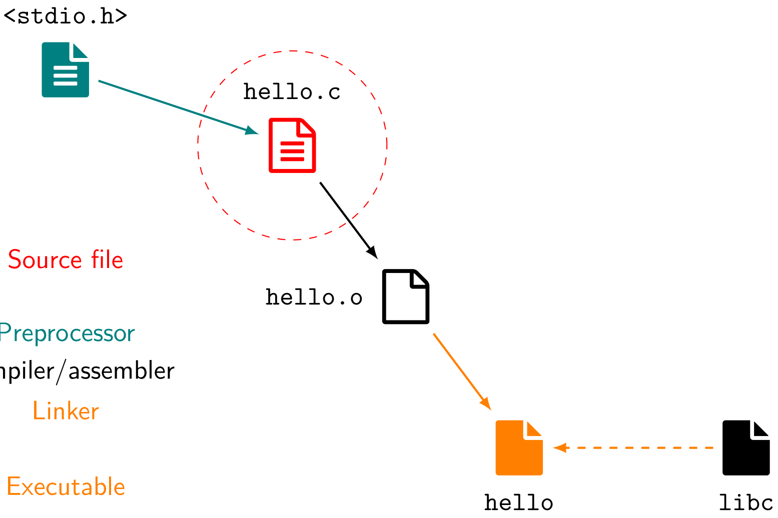
```c
#include <stdio.h>

int main(void) {
    printf("Hello 02635!\n");
    return 0;
}
```

Compile and run your program:

```
$ gcc -Wall -std=c11 hello.c -o hello
$ ./hello
```

Creating a C program



Source file

Preprocessor

Compiler/assembler

Linker

Executable

Compiling “Hello World” with GNU Make

```
$ make hello
gcc     hello.c   -o hello
```

```
$ make "CFLAGS=-std=c11 -Wall" hello
gcc -std=c11 -Wall    hello.c   -o hello
```

Makefiles

Create a plain text file and call it `Makefile` (no extension!)

```
CC=gcc                  # C compiler
CFLAGS= -std=c11 -Wall  # Extra flags for the C compiler
LDLIBS=                 # Extra library flags (e.g. -lm)
```

```
$ make hello
gcc -std=c11 -Wall    hello.c   -o hello
```

Visual Studio Code (VS Code)

- modern and customizable text editor
- multiple platforms (Windows, macOS, Linux)

Extensions

- auto-formatting, auto-complete, linter: `C/C++`
- WSL support: `WSL` extension