NUR A lecture 1

In this lecture:

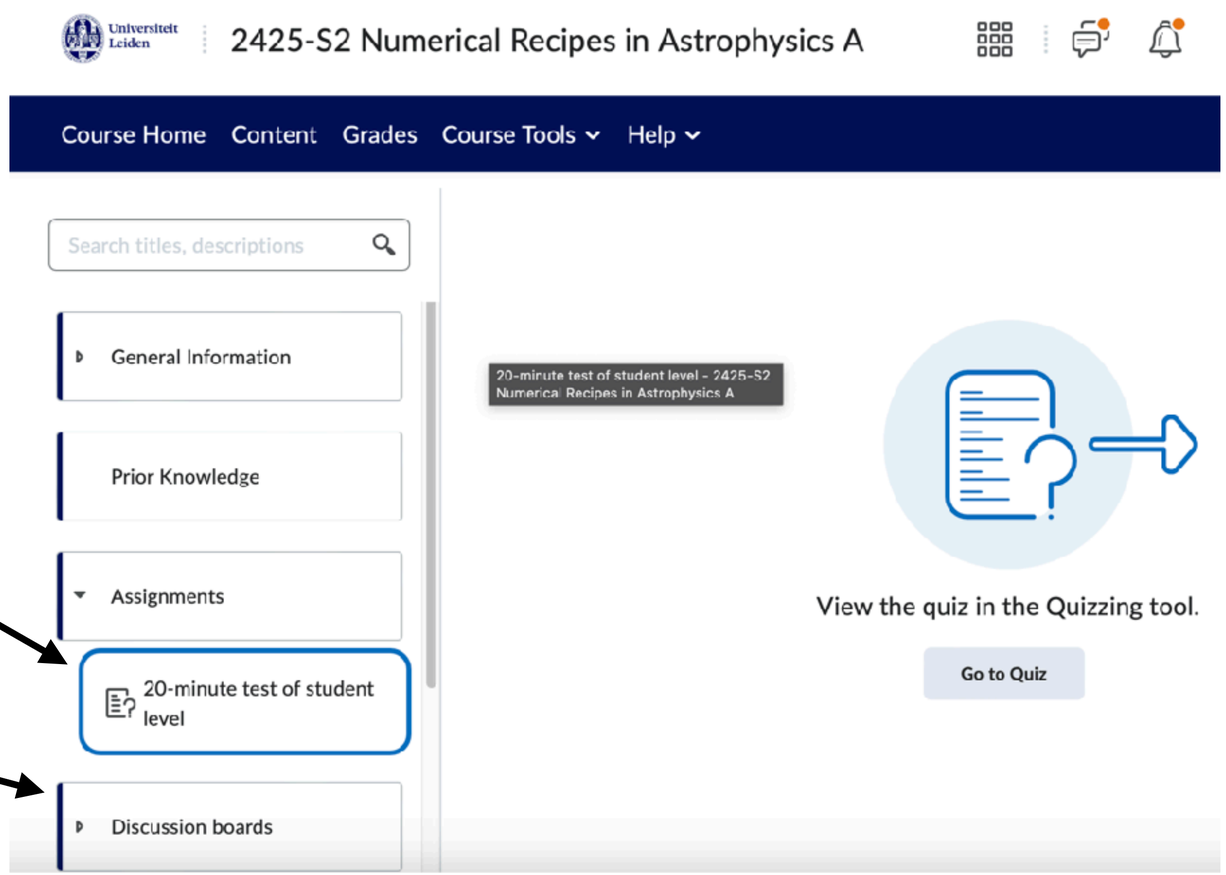
Introduction to the course
Integer representation
Integer arithmetic and logic
Binary operations
Floating point numbers
Sources of numerical error
Efficiency vs accuracy

Numerical Recipes for Astrophysics A

Spring semester 2025

Things to know

- Lecturer: dr. Marcel van Daalen
- Teaching assistants: Maria Marinichenko & Konstantinos Kilmetis
- Course website: Brightspace
- Please do the 20-minute programming test on Brightspace!
- Outside class hours you can ask questions on the discussion boards
- Note: two lectures this week, two tutorials + one lecture next week



Things to know

- The lectures discuss theory and specific algorithms
- After today, we start each lecture/half with a Mentimeter quiz (menti.com)
- During the tutorials you will solve problems by coding up the algorithms discussed (collaboration with each other is encouraged – but not with LLMs)
- Three larger exercises should be handed in (individually!) and count towards 50% of your grade (total)
- Participating in the tutorials means the hand-ins (and studying for the exam) are MUCH less work!
- At the end of the course a theory exam will determine the final 50% of your grade

Introduction

- Algorithms

A series of steps to get from A to Z

initial state

instructions

solution

E.g. Runge-Kutta

Sieve of Eratosthenes

long division

- Broad scientific libraries: GSL and SciPy

GSL

SciPy

Python

- Stability, accuracy and efficiency

"how reliable"

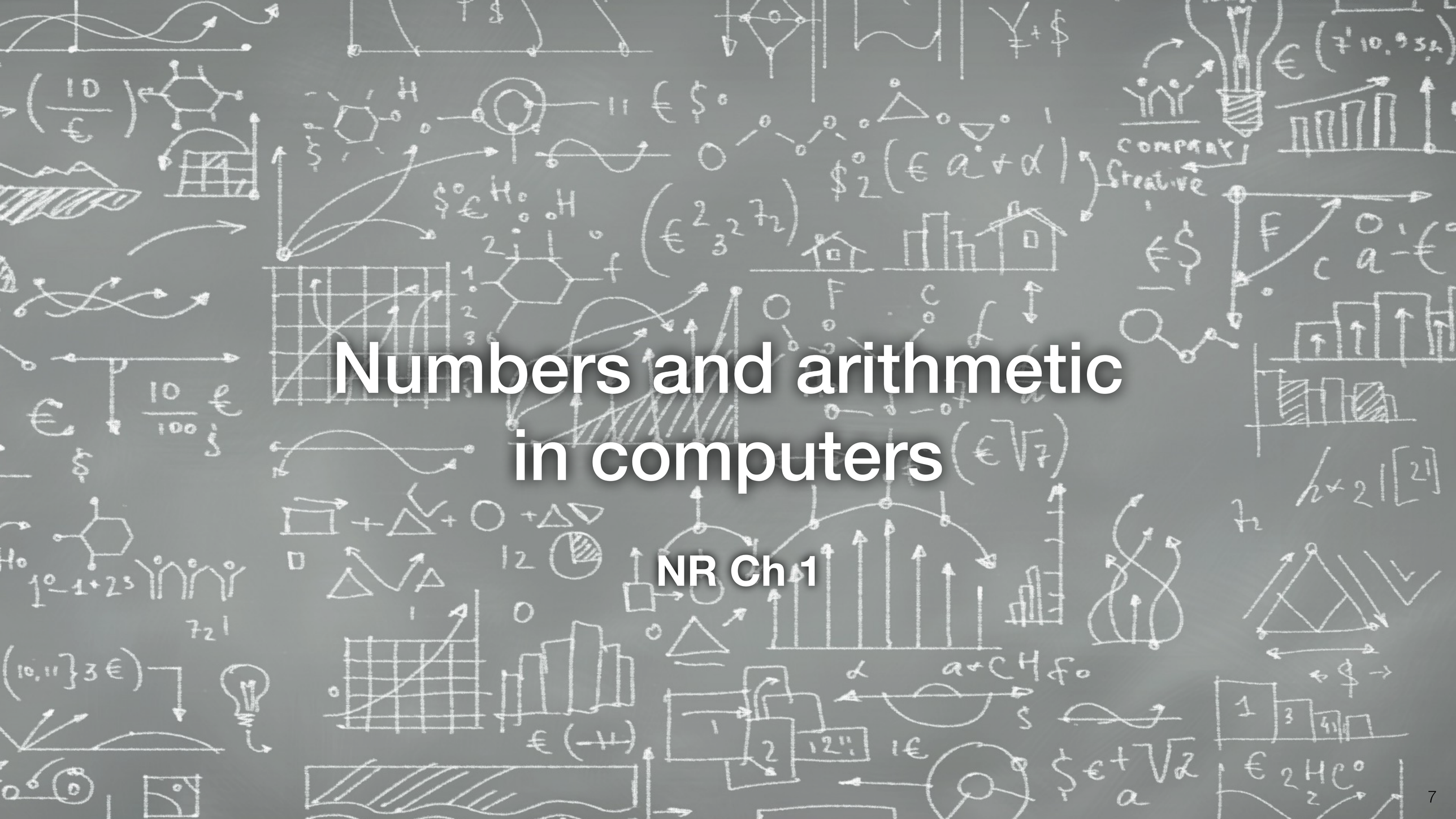
"how close"

"how quickly"

reduce steps for CPU

Topics

L1: Numbers and arithmetic in computers
L2: Inter- and extrapolation
L3: Solving linear equations
L4: Function integration
L5: Differentiation
L5: Random numbers and sampling
L6: Sorting and selection
L6: Root-finding/intersection
L7: Minimization/maximization/optimization
L8 & L9: Fitting and modelling data
L9: Statistical tests
(L10: Q&A)



Numbers and arithmetic in computers

NR Ch 1

Bits, bytes and ints

Bit: 0, 1 → byte = 8 bits → 2^8 values

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
0	0	1	0	1	1	0	1	= 45

values: [0, 255] or [-128, 127]

- Binary representation

32-bit (4 bytes): $2^{32} \simeq 4.3 \times 10^9$

64-bit (8 bytes): $2^{64} \simeq 1.8 \times 10^{19}$

- Integer addition and multiplication

```
  00001011  (11)
+ 00000110  (6)
  --------
  00010001  (17)

  11111111  (255)
+ 00000001  (1)
  --------
  ...00
```

Mult. X by 2^Y ≡ shifting bits of X left by Y positions

00010110 (22) × 00000101 (5) =

00010110 (22) + 01011000 (88) = 1110

More integer arithmetic

- Bit-wise operators
- Examples

Bit-shifts: $x \ll y$ $(\equiv x \cdot 2^y)$

$x \gg y$ $(\equiv x / 2^y)$

$x \,\&\, y$: AND, 1 if both bits are 1, 0 otherwise

$x \mid y$: OR, 1 if at least 1 of 2 bits is 1

$x \wedge y$: XOR, 1 if exactly 1 of 2 bits is 1

$\sim x$: NOT

↳ complement of x $(0 \leftrightarrow 1)$

$(7 \gg 2) \ll 2 = 4$

$(4 \wedge 13) \mid 10 = 11$

$(3 \ll 2) \,\&\, \sim 5 = 8$

$x \,\%\, 2^y = x \,\&\, (2^y - 1)$

More integer arithmetic

- Negative numbers $-X = \sim(X-1) = \sim X + 1$ (2's complement)

$-X + X = 0 \rightarrow \sim X + 1 + X = 0$

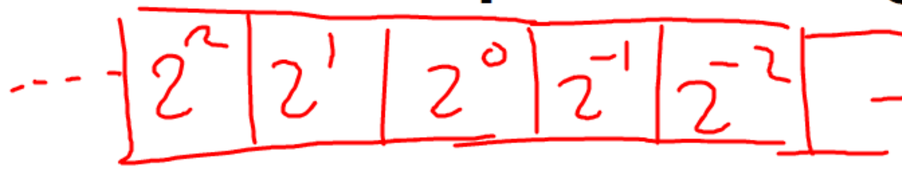
- Subtraction and division

$Y - X = Y + (-X) = Y + \sim X + 1$ | $X / 2^Y \equiv$ shift x right by y pos

in general:

```
00010110 (22) / 00000011 (3) = 2^2 + 2^1 + 2^0 = 7
 01100
 ----
 01010
 00110
 ----
  0100
  0011
  ----
  0001
```

Representing floating point numbers



? no: inefficient ⇒ scientific notation

base 10: $a \times 10^b$, $a = X.xxxx\ldots$

⇒ base 2: $1.xxxx\ldots \cdot 2^E$ (Fraction F; mantissa M; E ← exponent (int))

S-bit: $(-1)^S$

F: 2^{-1} 2^{-2} 2^{-3} ...

IEEE 754

- 32-bit (float, single-prec.): S: 1 bit, E: 8, F: 23; $2^{[-126, 127]}$
- 64-bit (double, -prec.): S: 1, E: 11, F: 52; $2^{[-1022, 1023]}$

1.625 → E = 0

0.625 → F = 10100

FLOPs and reducing operations

FLoating point OPeration

Addition, subtraction, multiplication = 1 FLOP

Division: 4-16 FLOPs, sqrt: 6-32 FLOPs

X/2.0 → X*0.5

X/9.0 many times → define oneNinth=1.0/9.0 ← multiply!

$\sqrt{dx^2+dy^2+dz^2} < 5 \rightarrow dx^2+dy^2+dz^2 < 25 (=5^2)$

Sources of numerical error

- Overflow and underflow

"too large to represent" — ints: wraps around
floats: ±∞ (NaN, not a number, e.g. x/0)

"too small to represent" → rounded to zero

$$P_\lambda(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

single prec.

$P_5(34) \approx 1.33 \times 10^{-17}$ (correct)

$P_5(35) = 0$ $(2^{-127} \sim 10^{-39})$

correct: 1.9×10^{-18}

Solve: rewrite equations, order of operations, or log space

Sources of numerical error

- Machine precision

Fractional machine accuracy: ε_m

e.g. $1.0 + 10^{-100} = 1.0$

ε_m: smallest number we can add to 1.0 and not get 1.0

ε_m: single prec. $\sim 10^{-7}$, double: $\sim 10^{-16}$

e.g. 5 can be represented exactly, but 4.9 and 0.1 cannot

if((4.9+0.1)==5).. ✗

Sources of numerical error

N = # of operations

- Round-off, stability and truncation
  - accumulation of machine error
    - if random: frac. error $\sqrt{N}\,\varepsilon_m$
    - in practice: $\sim N\,\varepsilon_m$
  - unstable: error is magnified

e.g. $(1.0002 \pm \varepsilon_m) - (1.0001 \pm \varepsilon_m) = 10^{-4} \pm \sim \varepsilon_m$

frac. error $\times 10^4$!

under our control!

Continuous $\rightarrow$ discrete

$\int \rightarrow \sum_n$

Efficiency vs accuracy

- At odds or hand-in-hand?

e.g. truncation error

e.g. round-off

$$\text{E.g. } \bar{X} = \sum_i \left(\frac{x_i}{n}\right) \quad \underline{\text{or}} \quad \left(\sum_i x_i\right)/n$$

in general better

- Vectorized operations/avoiding loops

Array X: $X^* = 0.2$ is better than a loop and/or divide by 5

avoid loops in Python

Thursday:
Inter- and extrapolation