Example Class 1 — Numbers, complexity, the shell and Python

C1 Research Computing — Michaelmas 2026
MPhil in Data Intensive Science and MPhil in Computational Biology

Exercises

The exercises are done without a calculator, as in the examination.

Bit patterns of float32 and bf16

A `float32` stores $(-1)^s \times 1.f \times 2^e$ in 1 sign bit s, 8 exponent bits holding $e + 127$, and 23 fraction bits f. bf16 keeps the sign, the exponent bits and the first 7 fraction bits, rounding to nearest; a tie goes to the value whose last bit is 0. Write patterns as sign | exponent | fraction.

(a) Give the 32 bits of -0.40625.
(b) Give the value of `0 | 10000011 | 00101000000000000000000`.
(c) Give the bf16 bits and value of each `float32` below.

nearest to $e = 2.71828\ldots$	`0 | 10000000 | 01011011111100001010100`
$1 + 2^{-8}$	`0 | 01111111 | 00000001000000000000000`

Running sums in low precision

fp16 has 1 sign bit, 5 exponent bits and the rest fraction bits; bf16 has 1 sign bit, 8 exponent bits and 7 fraction bits. With f fraction bits, numbers in $[2^e, 2^{e+1})$ are 2^{e-f} apart.

Readings of 0.25 are added one at a time to a sum that starts at 0 and is rounded at each step to nearest, ties to even. A sum of a constant c stops changing once the spacing reaches $2c$.

(a) Give the number of fraction bits of fp16, and the value at which an fp16 sum stops.
(b) Give the value at which a bf16 sum stops.
(c) Give the mean of 1000 readings, as the bf16 sum divided by 1000, and the true mean.
(d) Give the value at which a `float32` sum stops (23 fraction bits) and, as one bullet point, why a `float32` sum gives the right mean for up to ten million readings.

Memory and operation counts

A fourth-order second derivative on $N = 2^{18}$ grid points is an $N \times N$ `float64` matrix A with 5 nonzero diagonals. It acts on a vector ψ of N `complex128` values, each made of two `float64`. A is stored densely, as an $N \times N$ array, or as its bands, a $5 \times N$ array.

(a) Give the size of ψ in bytes and in MiB.
(b) Give the size of the dense A in GiB, and of its bands in MiB.
(c) Give, in Θ notation in N, the cost of $A\psi$ with each storage.

(d) Count N^2 operations for the dense product, one per matrix element, and $5N$ for the banded one. Give both counts, and how many times more the dense product costs, to two significant figures.

(e) Repeats among $n = 300\,000$ numbers are found by comparing every pair, or by a merge sort and one pass over neighbours. The sort first pads the list with $+\infty$ to $M = 2^{\lceil \log_2 n \rceil}$ values. Count n^2 operations for the pairs and $M \log_2 M$ for the sort. Give M, both counts and their ratio, to two significant figures, and the cost of the sort in Θ notation in n.

Recursion trees and the Master theorem

Splitting a problem of size N into a subproblems of size N/b, with $\Theta(N^d)$ work outside the recursion, costs $T(N) = a\,T(N/b) + \Theta(N^d)$, the Master-theorem form. Level j of the recursion tree, the root being level 0, holds a^j subproblems of size N/b^j, which do $N^d(a/b^d)^j$ work in all. Below, $T(1) = 1$.

(a) Draw the recursion tree of $T(N) = 4\,T(N/4) + 1$ for $N = 16$. Label each node with its size, and each level with its work outside the recursion, counting 1 for each leaf. Give the total, and $T(N)$ in Θ notation.

(b) For $T(N) = 4\,T(N/2) + N$ and $T(N) = 3\,T(N/3) + N$, give a, b and d, compare a with b^d, and give $T(N)$ in Θ notation.

(c) Which of the three recurrences does the same work at every level?

Environment variables and child processes

The script `show.py` prints `os.environ.get("NTHREADS", "unset")`. Each line below runs in a new `bash` shell in which `NTHREADS` is unset. A child process inherits the exported variables of its parent. `NAME=value` written before a command sets `NAME` for that command only, and parentheses run their commands in a subshell, a child process of the shell.

```
NTHREADS=4; python show.py
NTHREADS=4 python show.py; echo "[$NTHREADS]"
export NTHREADS=4; python show.py; unset NTHREADS; python show.py
(export NTHREADS=4); python show.py
```

(a) Give what each line prints.

(b) Give, as one bullet point, why the last line prints `unset`.

Code run at import

The three files below share a directory, and `SIM_NSTEPS` is unset when `python main.py` starts. A module's code runs when it is first imported, and Python keeps the module in the dictionary `sys.modules` under its name. `reload`, from `importlib`, executes a module's code again, in the same module object.

config.py

```
import os

print("config.py runs")
NSTEPS = int(os.getenv("SIM_NSTEPS", "1000"))


def nsteps():
    return int(os.getenv("SIM_NSTEPS", "1000"))
```

analysis.py

```
import config as settings


def steps():
    return settings.NSTEPS
```

main.py

```
import os, sys
from importlib import reload
from config import NSTEPS
os.environ["SIM_NSTEPS"] = "250"
import analysis
print(NSTEPS, analysis.steps(), sys.modules["config"].nsteps())
reload(analysis.settings)
print(NSTEPS, analysis.steps(), analysis.settings is sys.modules["config"])
```

(a) Give the four lines that `python main.py` prints.
(b) Give, as one bullet point, why `NSTEPS` in `main.py` keeps its value after the reload.
(c) Did `import config as settings`, in `analysis.py`, run `config.py` again? Give, as one bullet point, what in the output shows it.

Late binding in closures

A least-squares fit uses the basis functions $1, x, x^2, x^3$, built as below.

```
basis = [lambda x: x**k for k in range(4)]
print([f(2.0) for f in basis])
```

(a) Give the list printed.
(b) Give, as one bullet point, the reason.
(c) Correct the first line in two ways, with a default argument and with `partial` from `functools`, and give the new output.