Teaching Statement

Approach

I teach computational mathematics the way it is practised: by building, running and scrutinising computations. A convergence result becomes convincing when students observe the predicted rate in code they wrote themselves, and an implementation becomes trustworthy when students can explain why it works. I therefore teach every topic on two tracks, the mathematical foundations (approximation, stability, convergence, complexity) and their realisation in tested software on current hardware, and let each track motivate the other.

I set clear and transparent standards. Learning goals and assessment criteria are stated in the first lecture; frequent low-stakes checkpoints, short concept questions, automatically tested exercises and code reviews give students early and actionable feedback; students who want more get stretch goals such as performance challenges or small open questions. I aim for an inclusive classroom through structured participation, accessible materials, and by making the reasoning behind an error, not only the correct answer, part of the discussion.

Methods in practice

My lectures start from a concrete problem, for example a structure in contact, a valve in a flow, or a large sparse system on a GPU, and then unpack the algorithms and the theory beneath it. I use live coding and live derivations, short think–pair–share questions, and exercise sessions focused on debugging and conceptual clarity. Materials stay concise: worked examples, annotated code, and small unit tests that encourage minimal reproducible examples.

Assignments are scaffolded. Students first implement and verify a method on a model problem, with convergence studies and unit tests; then they analyse its cost; then they apply it to a larger problem where the cost matters. This sequence builds the professional habits of a computational mathematician: version control, testing, reproducibility, performance awareness, and critical interpretation of numerical results. It also makes the mathematics stick, because a student who has watched an unstable scheme diverge in their own code rarely forgets the stability condition.

Experience

At the Università della Svizzera italiana I am course director and lecturer of *Numerical Programming* in the BSc in Data Science, where I integrated testing, performance awareness and reproducibility into the numerical curriculum. I lectured in the MSc course *Software Atelier: Partial Differential Equations*, and I held five teaching-assistant assignments in MSc and PhD courses, including *Linear and Nonlinear Multiscale Solution Techniques* and *Software Engineering for Computational Science*. I also designed and taught five outreach courses on parallel computing and n-body simulation for high-school students and teachers.

Mentoring is the part of teaching closest to research. I have mentored one postdoctoral researcher and one doctoral student, with a second in progress, and supervised or co-supervised eight master's theses, three bachelor's theses and four undergraduate internships, on topics from high-performance finite element kernels and spectral schemes to physics-informed neural networks. Several of these projects led to publications or to software that my research still uses. I onboard students gradually, from guided literature reviews and replication studies to independent ownership of a research question, with regular one-to-one meetings, milestone plans and written feedback. I have also served as an external member of three doctoral defence committees at the University of Pavia.

Outside the university, I served as an officer in the Swiss Army, where I co-organised training and operations for units of 100 to 150 people. That experience taught me how repeated practice in progressively

harder settings consolidates skills, and how much clarity about expectations matters when people are learning under pressure.

Graduate courses

Courses I am prepared to offer at graduate level, connected to my research:

- **Multilevel and domain-decomposition methods**: multigrid, Schwarz and substructuring methods, their convergence analysis and their parallel implementation.
- **Numerical methods for contact and coupled problems**: variational inequalities, mortar and Lagrange-multiplier methods, and fluid–structure interaction.
- **High-performance scientific computing**: matrix-free methods, mixed precision, performance modelling on GPUs, and the software practices that make large computations reproducible.
- **Scientific software engineering**: testing numerical code, reproducibility, continuous benchmarking, and the design of libraries that outlive their first user.
- **Scientific machine learning for simulation**, a course I would add once that research direction matures rather than one I teach today: simulation as a generator of training data, learned components inside numerical pipelines, and what can and cannot be guaranteed about them.

English is my working language in teaching and research; Italian is my mother tongue.