Module Code	**EEP55C36**
Module Name	Computational Methods and Algorithms
ECTS Weighting[1]	5 ECTS
Semester taught	Semester 1
Module Coordinator/s	Dr. Hossein Javidnia
Module Learning Outcomes with reference to the Graduate Attributes and how they are developed in discipline	On successful completion of this module, students should be able to: LO1. Translate mathematical models into algorithms by reformulating engineering problems (e.g. signal, image, biomedical and geophysical data) into well-posed computational tasks. LO2. Implement and validate numerical, signal-processing and machine-learning solutions in Matlab and Python, making effective use of core libraries such as NumPy, SciPy, PyTorch and Matplotlib. LO3. Integrate domain-specific libraries and toolboxes to build end-to-end systems that ingest data, compute results and visualise outputs on appropriate platforms. LO4. Design and execute automated unit and integration tests that verify correctness, robustness and performance of code under realistic operating conditions. LO5. Manage code collaboratively with Git, applying branching strategies, pull requests and continuous-integration workflows for peer review and quality assurance. LO6. Quantify and optimise resource trade-offs—speed, memory footprint, energy usage and power dissipation—when selecting algorithms or hardware targets. LO7. Adapt and deploy computational pipelines at scale, using GPU acceleration, cloud services or on-premise clusters as required. LO8. Evaluate the deployment suitability of algorithms for real-time, batch or edge/cloud settings, justifying architectural choices. LO9. Communicate methods, assumptions and results clearly through concise technical reports, well-annotated code and informative visualisations.

[1] TEP Glossary

Graduate Attributes: levels of attainment

To act responsibly - Attained
To think independently - Attained
To develop continuously - Enhanced
To communicate effectively - Enhanced

Module Content

Please provide a brief overview of the module of no more than 350 words written so that someone outside of your discipline will understand it.

Computational Methods is a one-semester module that demonstrates how mathematical reasoning is converted into reliable, efficient software for real-world engineering and scientific tasks.

The syllabus is organised around a four-stage problem-solving cycle—problem analysis, algorithm design, implementation, and verification. It begins with vector and matrix representations of data, establishing the linear-algebra tools needed to manipulate images, audio snippets, biomedical traces, or seismic readings. Building on this foundation, the class explores least-squares techniques for regression and modelling, followed by linear and nonlinear optimisation methods that formalise goals such as cost minimisation under constraints.

Attention then turns to software architecture. Principles of decomposition and abstraction show how large computational systems are partitioned into self-contained modules with clear interfaces. Classic algorithmic patterns—recursion, backtracking, and divide-and-conquer—illustrate systematic ways to reduce complex tasks to simpler sub-problems.

The final block introduces time-series analysis, covering stationarity, differencing, and autoregressive modelling for forecasting and anomaly detection in ordered data streams. Throughout the module, implementations are developed in Python, leveraging widely adopted libraries (NumPy, SciPy, PyTorch, Matplotlib). Code is maintained under Git version control, accompanied by unit tests and profiling tools that expose speed, memory, and energy trade-offs across deployment targets.

By the end of the semester, participants will be able to translate mathematical formulations into executable algorithms, select appropriate library routines rather than duplicating existing solutions, evaluate resource constraints when scaling to large data sets, and communicate computational results clearly through concise technical writing and visualisation.

Teaching and Learning Methods

e.g., lectures, seminars, online learning via VLE, field trips, laboratories, practice-based etc…

- **Lectures (weekly, 2 hrs):** Structured, problem-led presentations that introduce the mathematical foundations, algorithmic techniques and software tools. Short live-coding segments and worked examples model the full "analyse → design → implement → verify" cycle.
- **Laboratories (weekly, 2 hr):** Supervised coding sessions in Jupyter/Python notebooks where students apply the week's techniques to progressively more challenging data-driven problems, receiving immediate feedback from demonstrators**.**

Assessment Details[2]
Please include the following:

- **Assessment Component**
- **Assessment description**
- **Learning Outcome(s) addressed**
- **% of total**
- **Assessment due date**

Assessment Component	Assessment Description	LO Addressed	% of total	Week due
Lab Portfolio A	Five short, individual coding tasks (vectors, matrices, least-squares, introductory optimisation). Each task is submitted via Git with clear comments plus a brief "AI-usage note" (≤ 150 words) describing any generative-AI help and how the output was verified. Unit-testing is not required at this stage.	LO 1, LO 2, LO 3, LO 5	30%	Weeks 1 – 5 (rolling)
Invigilated Lab Exam	Two-hour, supervised practical in the lab (Internet and AI tools blocked). Students implement a specified numerical routine on a supplied data set, then add basic unit tests and run a profiling script provided with the exam.	LO 1, LO 2, LO 4, LO 6	30%	Week 10
Oral Viva & Live Demo	15-minute, individual session: 5-min live demo of one Lab-Portfolio-B task*; 5-min code walkthrough focused on resource trade-offs and testing; 5-min Q & A probing algorithm choices and any documented AI usage. Examiners may change an input file or comment out a function to confirm understanding.	LO 6, LO 7, LO 8, LO 9	40%	University Exam Period

[2] TEP Guidelines on Workload and Assessment

	*Lab Portfolio B = the four tasks completed after week 6 (recursive algorithms, abstraction, unit-testing and profiling). These are part of the same continuous portfolio but introduced later, once unit-testing concepts have been taught.
Reassessment Requirements	Reassessment with a problem solving exercise in lab. Assessment based on an oral presentation and demonstration of the solution.
Contact Hours and Indicative Student Workload[2]	**Contact hours:** 44 • **Lectures**: 11 weeks × 2 hrs = 22 hrs • **Labs**: 11 weeks × 2 hrs = 22 hrs **Independent Study (preparation for course and review of materials):** 44 **Independent Study (preparation for assessment, incl. completion of assessment):** 39
Recommended Reading List	• Numerical Recipes in C • Computational Science and Engineering by Gilbert Strang • Introduction to Computational Models with Python by Jose M Garrido
Module Pre-requisite	Undergraduate Numerical Methods, Signals and Systems
Module Co-requisite	None
Module Website	
Are other Schools/Departments involved in the delivery of this module? If yes, please provide details.	No
Module Approval Date	
Approved by	
Academic Start Year	
Academic Year of Date	**2025-26**