2026 ASEE ANNUAL CONFERENCE June 21–24, 2026 | Charlotte Convention Center, NC
Host Sponsor UNC Charlotte
ASEE AMERICAN SOCIETY FOR ENGINEERING EDUCATION
Paper ID #53797

From MATLAB II to Computational Decision Making – Redesigning Numerical Methods Courses in the Age of Generative AI

Dr. David Edward Torello, Georgia Institute of Technology

Dr. David Torello graduated with his B.S. in mechanical engineering from UC Berkeley and his M.S. and Ph.D. from the Georgia Institute of Technology. He is currently a Senior Academic Professional at the Georgia Institute of Technology, where he serves two primary roles: instructional faculty in mechanics, computational techniques, and laboratory courses as well as the executive director of the John H. Martinson Honors Program. His engineering education research interests lie in alternative assessment techniques and course change best practices.

Jill Fennell, Georgia Institute of Technology

Jill Fennell, the Frank K. Webb Chair in Communication Skills at the George W. Woodruff School of Mechanical Engineering at Georgia Tech, focuses on advancing written, visual, and verbal communication skills. Her research centers on affect theory and its application to technical communication, specifically information design. Jill studies how to enhance the effectiveness of pedagogical documents by incorporating principles from affect theory. Through her work, she aims to empower students, fostering an environment where they actively shape their communication interactions, including teamwork and ethical discussions. By integrating these principles, she goes beyond traditional methods, ensuring that students not only learn but also take an active role in shaping their communication experiences.

©American Society for Engineering Education, 2026

From MATLAB II to Computational Decision Making – Redesigning Numerical Methods Courses in the Age of Generative AI

Context and Problem Statement

With widespread access to generative artificial intelligence (GenAI) and large language models (LLMs), the ability to generate complete numerical algorithms in common engineering programming languages such as MATLAB, Python, and C requires minimal effort and coding understanding [1]-[5]. These systems are often faster than undergraduates at producing these algorithms in code at a similar or higher level of competency [1]-[4]. Given that a long-standing assumption that coding proficiency and assessing algorithm output accuracy is a reliable proxy for conceptual understanding of this topic, the modern numerical methods classroom is experiencing an existential upheaval, manifesting as a troubling symptom: students appear to be "coding without thinking," commonly referred to as "vibe coding." For a numerical methods course, this means students may easily create syntactically correct programs with plausible results while demonstrating limited ability to justify modeling choices and assumptions, interpret outcomes, or assess solution validity. This is supported by research showing that AI-assisted programming results in learners frequently accepting generated code without fully understanding it, often obscuring understanding of underlying concepts [2]-[6]. Compounding this issue with the well-recognized fact that LLM-generated outputs are non-deterministic and susceptible to hallucinations, there is a strong risk that student work that looks correct may rest on faulty logic or contain unjustified engineering decision-making [6]-[9]. Traditional assessments that emphasize code creation or numerical correctness therefore risk rewarding effort that is based on minimal understanding and no longer indicate meaningful learning.

These developments raise a foundational question for numerical methods education: What counts as learning when AI can do the coding? This work attempts to justify building a modern numerical methods course around "durable" competencies such as problem framing, computational decision making, model validation, and communication that are central to engineering practice and resistant to automation [10]-[14]. Foregrounding the teaching and assessment of these skills reframes GenAI as a forcing function that compels a reconsideration of course purpose, learning objectives, and assessment practice instead of an external threat to policed. This model of a numerical methods classroom allows for the exploration of modern tools for computing tasks while emphasizing capacities that remain essential to engineering as code generation becomes increasingly trivialized by GenAI advancements.

Course Before Redesign: What Learning Looked Like in MATLAB II Mode

Prior to the redesign at hand, this course in numerical methods was focused on the mathematical derivations and implementations of common numerical methods and the coding of these methods in MATLAB. Course enrollments are typically 60-70 students that are predominantly second year students, although first- and third-year students may enroll in the course depending on curricular factors, and all students have had an introductory course in MATLAB programming. The course design involved homework for practicing coding and hand-calculation of methods, one or two projects that applied these methods to a complicated problem or case study, and three paper examinations in which students solved math problems using class concepts with some exposure to conceptual questions about underlying assumptions in the tested methods. The core

learning activity in the course revolved around writing code to implement numerical algorithms, and as a result many students would refer to the course as "MATLAB II." While many of the assignments and examinations promote learning at useful levels for the implementation of numerical methods in engineering practice, the dominant time on task was spent on the implementation or understanding of computer code, which is precisely the problem: in the age of generative AI, accurate code is cheap to produce with little cognitive engagement from students. Because of this new reality, it would be unwise to consider the core learning experience of the course (writing code) to be a durable skill, and thus this implementation of the course now serves a less than useful purpose in the modern engineering curriculum.

Course After Redesign

The course redesign shifts the focus of learning in the course to durable skills by emphasizing engineering logic and decision-making alongside communicating numerical results because these are skills or practices that AI is not yet competent at reproducing in a niche engineering contexts. The new implementation revolves around creating assignments that make students' engineering logic visible, rather than allowing reasoning to remain implicit or embedded solely in code. To support the desired cognitive shift, we embedded the numerical methods assignment within a professional scenario in which students act as interns at a consulting firm responding to a client's needs, as summarized by a senior engineer (Anthony). The primary deliverable—a dual-audience genre consisting of a client-facing slide deck accompanied by internal presenter notes—was selected intentionally to mirror professional engineering communication, where different stakeholders require different levels of detail and justification.

This genre choice foregrounds metacognition by requiring students to make explicit decisions about what counts as evidence, what assumptions must be defended, and how computational results should be interpreted to support action. By separating client-facing slides from internal presenter notes, the assignment requires students to reason about the function of their computations: which methods require justification because they shape model validity, which results are decision-relevant because they intersect with client thresholds, and which technical details can remain internal or are not important enough to share. In this way, audience awareness is not treated as a stylistic concern but as a proxy for engineering logic—students demonstrate understanding by aligning methods, results, and conclusions with the constraints, decisions, and risks that define the problem space. Unlike traditional problem sets, where correct implementation can mask shallow reasoning, this genre forces students to externalize how and why their computational choices support a defensible engineering decision. Figure 1 uses Bloom's Taxonomy to illustrate how this design relocates cognitive effort from code production to higher-order reasoning—shifting assessment away from procedural application toward analysis of constraints, evaluation of method fit, and creation of decision-ready justifications.

To operationalize this reasoning across the course, the slide templates and associated prompts were structured to scaffold decision-making across phases of engineering work. Distinct sections for *Methods & Assumptions*, *Results*, and *Conclusions* are paired with guiding questions that cue justification rather than description. For example, students are prompted not simply to list methods used, but to explain how those methods align with the client's constraints or operational needs. This structure encodes expectations about engineering logic directly into the deliverable, reducing reliance on implicit norms. Instructional support was provided to help students develop

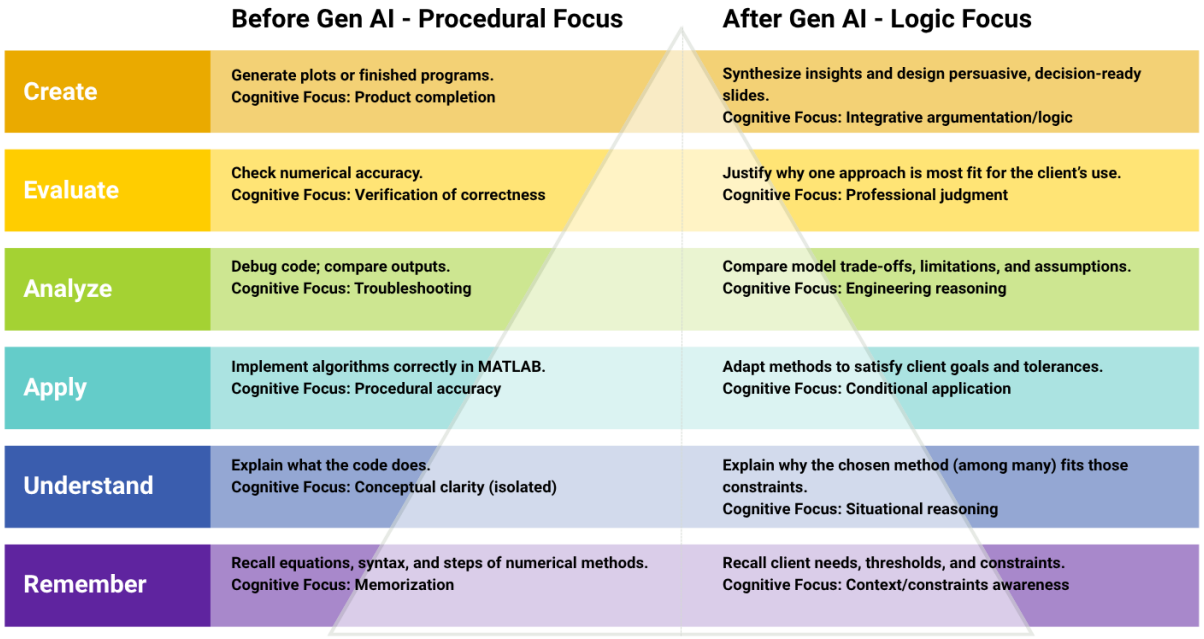

Figure 1: Conceptual mapping of cognitive demands before and after the course redesign.

justification as an explicit skill. In-class workshops, modeling examples, and short activities asked students to transform procedural statements into context-driven explanations, link client constraints to method choices, and revise slide titles to reflect claims rather than categories. Feedback language consistently prioritized reasoning quality—why a choice was made and how it supports a decision—over procedural completeness alone.

Finally, the assessment scheme was revised to reinforce these priorities. While computational correctness remains necessary, grades are weighted toward the clarity and defensibility of students' reasoning, including their ability to articulate assumptions, interpret results relative to client thresholds, and justify recommendations. Together, these moves reposition computation as evidence within an argument rather than as the sole object of evaluation—an intentional design choice that also makes students' underlying reasoning visible.

Tragedy and Triumph: Navigating Shifts in Practice and Cognitive Focus

The following are our lessons learned from teaching the course and rely on two predominant sources of assessment: student feedback in the form of email, MS Teams messages, personal conversations, and Course Instructor Opinion Survey results, as well as close reading of the deliverables and measured progression on learning objectives through grading practices.

Tragedy: Skipping Context in Favor of Tasks

Despite the scenario-based framing, many students reported skimming or skipping the opening paragraphs describing the client's needs and instead moving directly to the numbered list of technical tasks outlined by Anthony. In meetings with instructional staff, students frequently arrived with completed code and calculations but struggled to determine how to populate the slide deck template in a meaningful way. When asked to explain why particular numerical

methods were selected, students often defaulted to procedural descriptions rather than reasoning grounded in the client's goals, constraints, or decision thresholds. This behavior revealed a deeper issue than inattentive reading. Students interpreted the task list as a traditional programming assignment—treating each item as an isolated requirement to be executed correctly—rather than as components of a larger engineering decision-making process. The client context, which was intended to function as a set of design constraints, was instead perceived as narrative background with little bearing on "real" engineering work. As a result, students were unable to distinguish between *method justification* (why a particular approach was appropriate for the problem) and *result interpretation* (what the computed values meant for the client's decision). Without anchoring their work in client-defined needs, all outputs collapsed into undifferentiated data reporting. Importantly, this outcome was pedagogically diagnostic. The genre did not fail to communicate expectations; rather, it exposed how students conceptualized engineering logic itself. By privileging task execution over contextual reasoning, students demonstrated a model of engineering grounded in procedural completion rather than in judgment under constraint. This misalignment clarified the need for additional scaffolding to help students recognize that understanding the client's problem is not preparatory work, but central to defining what counts as an appropriate method, a meaningful result, and a defensible conclusion.

Triumph: Identifiable Engineering Logic Development

The revised assignment structure successfully shifted students away from a narrow conception of correctness centered on numerical accuracy and toward higher-order reasoning focused on method justification, result interpretation, and engineering decision-making. Although students initially struggled to distinguish between method justification and result interpretation, nearly all reported being "forced to think about what information was useful to prove that my answer was reasonable." Many also noted that they had not previously considered figures as tools of communication or justification but came to appreciate their role in rapidly conveying meaning and guiding interpretation for external audiences. Students with prior engineering work experience further observed that the assignment closely mirrored professional practice, particularly in its emphasis on defensible reasoning and contextual judgment. Together, these outcomes suggest that the redesigned assignments more effectively support analysis, evaluation, and creation than traditional code-focused tasks, leading to more authentic and professionally relevant engineering learning.

Implications and Ongoing: Supporting Student Justification Internalize

Future steps should focus on aligning in-class activities with the new deliverable format and strengthening assessment to support course redesign goals. Although assignments were restructured to promote durable skills, content delivery remained largely lecture-based, with limited attention to epistemic cognition. This material would benefit from scaffolded instruction using "I do, we do, you do" approaches, along with a grading-for-growth model that emphasizes feedback and revision. Additionally, evidence of success is primarily qualitative and anecdotal. A key aim of this work is to develop a more rigorous evaluation process through consultation with peers and the ASEE community. By offering clearer evidence of impact and implementing stronger feedback and assessment of student learning and epistemic decision-making, we aim to build students' confidence and capacity for higher-level cognitive engagement in numerical methods within engineering contexts while providing a replicable design framework.

References

1. Y. Li *et al.*, “Competition-level code generation with AlphaCode,” *Science*, vol. 378, no. 6624, pp. 1092–1097, Dec. 2022.
2. S. Imai, "Is GitHub Copilot a Substitute for Human Pair-programming? An Empirical Study," *2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, Pittsburgh, PA, USA, 2022, pp. 319-321.
3. A. Mastropaolo, L. Pascarella, E. Guglielmi, M. Ciniselli, S. Scalabrino, R. Oliveto, and G. Bavota, “On the robustness of code generation techniques: An empirical study on GitHub Copilot,” in *Proc. IEEE/ACM 45th Int. Conf. Software Engineering (ICSE)*, Melbourne, VIC, Australia, 2023, pp. 2149–2160.
4. Y. Xue *at al.*, “Does ChatGPT help with introductory programming? An experiment of students using ChatGPT in CS1,” in *Proc. IEEE/ACM 46th Int. Conf. Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, Lisbon, Portugal, 2024, pp. 331–341.
5. Q. Guo *et al.*, “Exploring the potential of ChatGPT in automated code refinement: An empirical study,” in *Proc. IEEE/ACM 46th Int. Conf. Software Engineering (ICSE)*, Lisbon, Portugal, 2024, Art. no. 34, pp. 1–13.
6. S. Ouyang, J. M. Zhang, M. Harman, and M. Wang, “An empirical study of the non-determinism of ChatGPT in code generation,” *ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 2, Art. no. 42, pp. 1–28, Feb. 2025.
7. Z. Ji *et al.*, “Survey of hallucination in natural language generation,” *ACM Comput. Surv.*, vol. 55, no. 12, Art. no. 248, pp. 1–38, Dec. 2023.
8. S. Lin, J. Hilton, and O. Evans, “TruthfulQA: Measuring how models mimic human falsehoods,” in *Proc. 60th Annu. Meeting Assoc. Comput. Linguistics (ACL)*, Dublin, Ireland, 2022, pp. 3214–3252.
9. E. M. Bender *et al.*, “On the dangers of stochastic parrots: Can language models be too big?” in *Proc. ACM Conf. Fairness, Accountability, and Transparency (FAccT)*, Virtual Event, 2021, pp. 610–623.
10. L. J. Shuman, M. Besterfield-Sacre, and J. McGourty, “The ABET ‘professional skills’—Can they be taught? Can they be assessed?” *J. Eng. Educ.*, vol. 94, no. 1, pp. 41–55, Jan. 2005.
11. H. J. Passow, “Which ABET competencies do engineering graduates find most important in their work?” *J. Eng. Educ.*, vol. 101, no. 1, pp. 95–118, Jan. 2012.
12. ABET, *Criteria for Accrediting Engineering Programs, 2025–2026*. Baltimore, MD, USA: ABET, Inc., 2024. [Online]. Available: https://www.abet.org
13. N. Oreskes, K. Shrader-Frechette, and K. Belitz, “Verification, validation, and confirmation of numerical models in the Earth sciences,” *Science*, vol. 263, no. 5147, pp. 641–646, Feb. 1994.
14. World Economic Forum, *The Future of Jobs Report 2025*. Geneva, Switzerland: World Economic Forum, 2025. [Online]. Available: https://www.weforum.org