Among our greatest challenges in incorporating computational skills in math classes is the variability of our students' backgrounds. Because our courses are not centered on the computational aspect, it becomes a byproduct of the central mathematical topics, and this is reflected in the time which is dedicated to computational skills. Although math majors are required to complete a computational course, not all of our classes are comprised of math majors. Nonetheless, our department would like to incorporate computation into most low level courses. We have found that our students tend to work on computational projects in groups with the common trait that at least one student can/will deliberately avoid practicing any computation, preferring instead to observe.

While an obvious method to address balance among students' work is to insist on individual projects, we find that the time required to support students with weaker computational backgrounds is then siphoned from the more mathematically relevant class topics. This causes confusion among all students about our priorities in class. During our calculus courses, we have instead begun to perform sample computations in front of the students before they begin their projects. We then expect them to replicate what they have seen. This "quick and dirty" approach is indeed quick, and bypasses some of the problems a beginner would have, but the less desirable outcome is that students of all backgrounds view our computational projects as banal. In fact, they lose the very thing the labs were designed to promote: seeing classroom topics 'in action' and analyzing the resulting computer output with respect to current topics.

An alternative solution has been to design courses around computation as a significant portion of the instruction. I have co-developed such a course for low-level undergraduate students. In it, we were able to offer more scaffolding steps to our students, at times sharing nearly complete code or complete code which the students must edit to achieve a different goal. Since it is understood that computation is one of our primary goals, we do not feel class time is 'lost' when it is spent on computation techniques. Indeed, this class was designed for computation to support the mathematical concepts rather than the converse. One major benefit to this class is that it has been co-taught, allowing one professor to focus on instruction, while the other can attend to students and their individual computational needs.

A second challenge we have faced in teaching computation is the struggle to see the 'big picture' of their techniques/results after working with the more minute details. In our exploratory course on computational mathematics, we offered an eclectic collection of problems as motivating examples from field like economics, traffic engineering, and hydrology. We then designed exercises to highlight the power and common pitfalls of computational techniques with emphasis on interpretation of results and recognition of nonsensical results. Following the exercises, we applied the results to our original example. We did *not*, however, discuss the methods of troubleshooting when our results are nonsensical, which leaves an incomplete picture for our students.

In the ideal scenario, we would use textbooks that support computational projects at several scales (single problems in a problem set, sections in a chapter, and even entire chapters). Most of our mathematical textbooks include several (at most 3) sections dedicated to computation (usually with a graphing calculator). Each of these is always the last section in a chapter and is nearly always ignored or at the very least deprioritized. Without supporting materials, students are entirely reliant on the professor and each other to help with skill development. Such a textbook would offer excellent support to our students and our instruction.