AC 2007-1548: IMPLEMENTING A VIDEO GAME TO TEACH PRINCIPLES OF MECHANICAL ENGINEERING

Brianno Coller, Northern Illinois University

Dr. Coller is an Associate Professor of Mechanical Engineering at Northern Illinois University. He received his Ph.D. in Theoretical & Applied Mechanics from Cornell University, and did postdoctoral research at the California Institute of Technology. In addition to research in education, Dr. Coller is interested in nonlinear dynamical systems and control.
Implementing a video game to teach principles of mechanical engineering

Abstract

The paper describes how a video game is used to teach numerical methods to mechanical engineering undergraduates. The video game provides an authentic and engaging context in which to learn computational techniques and concepts that are often dry and uninspiring. After outlining a study demonstrating that students in the video game-based course learn more deeply than students in more traditional textbook-based courses, we describe how learning outcomes are integrated into the game-play. We contrast the game-based assignments to typical textbook problems.

1 Introduction

For the past two years, we have been experimenting with a new way of teaching a numerical methods course to our undergraduate mechanical engineering students. Rather than designing the course around a textbook in which the students’ assignments are culled from the end-of-chapter exercises, we have built our new numerical methods course around a video game. Our goal has been to use the video game as an anchor for the course, providing an engaging context for almost all subsequent instruction, homework, and group projects.

Our motivation for doing so stems from the fact that annual revenues for the video game industry have long surpassed box office receipts from the movie industry. According to a recent report from the Kaiser Family Foundation, 83% of children between the ages of 8 and 18 have at least one video game console in their home; 31% have three or more. The statistics across all categories of race, gender and economic status are compelling. Children in this age group spend considerable time playing video games: 68 minutes per day on average. Unlike the passive medium of television, playing a video game usually means engaging one’s mind in complex problem solving exercises. The most successful video games are often ones that are the most challenging, mentally. They are also ones that have highly effective learning
environments built into the game-play. Several scholars\textsuperscript{1,13,17,23} argue that we educators can learn a lot from how video games engage players/learners.

After recognizing the potential motivating power of video games, however, the task of developing or adapting a video game to address specific academic outcomes in a traditional university course is daunting. There is a good reason why modern video games cost many millions of dollars to develop.\textsuperscript{7} In this paper, we describe a specific video game-based course that we have created to teach an undergraduate numerical methods course.

Our video game is car racing game called \textit{NIU-Torcs} that we co-developed. We show a screen-shot of the game in Figure 1. On its surface, \textit{NIU-Torcs} has much of the look, feel and adrenaline-pumping action of a modern video game (\textit{i.e.} the \textit{Need for Speed} and \textit{Gran Turismo} series of games). At its heart, however, the game is a sophisticated automobile simulation, faithfully capturing the physics and interactions of the major automotive subsystems. These technical elements allow us to link video game to engineering education outcomes. They provide the attachment points from which we can hang the scaffolding of an undergraduate course.

In this article, we outline the way in which we have built the scaffolding, providing details of how course content is integrated into the game. We do so in light of a companion study\textsuperscript{10} which has demonstrated that students taking the game-based numerical methods course gain a deeper understanding of course material than their counterparts who take a traditional course. Herein, we compare and (mostly) contrast the types of tasks we give our students with the those found in typical textbooks.

2 Course Overview

Our video game-based course is called \textit{MEE 381: Computational Methods and Programming in Engineering Design}. At Northern Illinois University, all undergraduate mechanical engineering students must either take this course or a more traditional textbook-based numerical methods course. Generally speaking, the goal in both classes is for students to learn how to get a computer to perform engineering calculations that are often too difficult or too cumbersome to perform by hand. Course content includes root finding, solving systems of linear algebraic equations, curve fitting, numerical differentiation and integration, and more. Before taking the computational methods course, students take four courses in calculus, including differential equations; basic engineering mechanics courses (statics, dynamics, and strength of materials); and an introductory computer programming course in C/C++.

Students in MEE 381 have (or have access to) a traditional numerical methods textbook\textsuperscript{20} for reference. However, the bulk of the assigned reading, class discussions, and homework assignments are built around the \textit{NIU-Torcs} video game. The reader is encouraged to view the 6 minute video (\url{http://www.ceet.niu.edu/faculty/coller/video.htm}) which depicts students’ accomplishments in the course.

At the beginning of the course, each student receives his or her own car. Initially, the car just sits motionless on the track. To get one’s car to move, a student must write a C++ program that gives the car its driving commands: how much to step on the gas pedal; how much to step on the brake
pedal; which gear the transmission should be in; and how much the steering wheel should be turned to the left or the right. In addition, the driving program may query from the simulation important information such as the car’s distance from the center line of the track; the heading angle of the car relative to the heading angle of the track; wheel rotation rates; and copious information about the track itself which students may use in computing their driving strategies. Students compile their driving programs into a library and link them to the main NIU-Torcs code. Then, students are able to see the fruit of their effort. The car simulation runs in real time, displaying the behavior of the car in full 3D graphics.

Our program NIU-Torcs is more than a simulation; it is a video game. Like most video games, it contains “levels” which players/students must complete in order to succeed at the game. These levels are clearly defined tasks that serve as homework assignments. As described in more detail in Section 4.1, the player/student starts off at the lowest (easiest) level in which she must simply get the car to steer itself around the track. As in the commercial video games Need for Speed and Gran Turismo, our game rewards its player/students when they successfully complete levels by giving them new cars to drive and new tracks to drive upon. Also, completion of a level, brings on a new challenge that requires students to learn new numerical methods and to apply them. The course climaxes with an open-ended project in which students form teams and participate in a friendly competition.

2.1 Additional Learning Opportunities

In addition to the learning outcomes related specifically to numerical methods as outlined in the first paragraph of Section 2, we have additional learning outcomes in MEE 381 related directly to improving students’ programming skills. If new (multi-platform) textbooks\textsuperscript{8,11,12,16,20,21} are any indication of national trends, it appears that we are swimming against the current. Instead of being exposed to a sampling of commercial software packages with canned numerical routines and libraries, students in MEE 381 create their own numerical routines and libraries in C++. In many ways, the game-based numerical methods course is a continuation of the students’ introductory programming course. In MEE 381, we introduce them to object oriented programming, data structures, dynamic memory allocation, and complexity analysis. Software engineering issues are always at the forefront as we strive to create code that is scalable and reusable. Because of these additional learning goals, we spend little more than half the in-class time discussing specific numerical methods content.

3 Course Effectiveness

Before describing the details of how we integrate course content into the video game, it is instructive to outline the results of a companion study\textsuperscript{10} in which we investigated the effect of the video game on student learning.
3.1 Time on Task

Regardless of how good an instructional medium is, students will not learn unless they invest their own time and effort to do so. According to Bransford et al.,\(^6\,19\) motivation to learn affects the amount of time students are willing to devote to learning. Furthermore, we humans are motivated to develop competence and solve problems when we can see the usefulness of what we are learning, and can apply it.

In the Spring of 2005, we surveyed all students taking core undergraduate mechanical engineering courses. Among other questions, we asked them how much time, outside of lecture/lab, they spend on each of their courses. The chart in Figure 2 summarizes results that are discussed in more detail in.\(^10\) Each bar in the figure represents the average number hours reported by students in a particular course. The hours are normalized by the average over all courses. The value of 2.07 for MEE 381 indicates that students spend roughly twice as much time on their game-based coursework than the average reported over all other courses. With the next highest value in the chart at 1.51, the MEE 381 students are investing roughly 35\% more time in the class than the next highest course.

![Figure 2: Hours students spend, outside of class, on their coursework.](image)

3.2 Depth and Breadth of Learning

To assess whether this extra time translates to improved learning, we compared students who took the game-based numerical methods course to a control group of students who took more traditional non-game-based numerical methods courses from four different instructors at two different universities.

Since different instructors emphasize different techniques, and since they motivate the material differently, it is difficult to devise an objective test that could be used to distinguish the two groups of learners. Therefore, rather than test students on specific course content, we asked them to tell us, in the form of a concept map, what they learned in their numerical methods course.

The purpose of the exercise is to get a snapshot of the structure of students’ knowledge of the course material within the last two weeks of the semester, before they typically begin studying for finals. As Chi et al.\(^9\) explain, “the structure of knowledge [has a] significant influence on intelligence and high-level cognitive performance.” In summarizing several decades of cognitive science research, Bransford et al.,\(^6\) explain that “knowing more” means:

A. “having more conceptual chunks in memory;”

B. having more “relations or features defining each chunk;”
In the concept maps, students are asked to express all the concepts and techniques they learned in the course, and do so in a hierarchical manner that articulates the dependencies, relations, and interrelations among topics. In light of the definition of “knowing more” above, assessing breadth and depth of learning in the two groups is a matter counting specific features of the concept map. Results are outlined as follows:

1. Students in the game-based course were able to recall roughly the same number of major numerical methods topics as their counterparts taking more traditional numerical methods courses. Major numerical methods topics include root finding, solving systems of linear algebraic equations, curve fitting, numerical differential equations, et cetera. Programming topics and applications of the numerical methods are not counted.

2. The two groups of students were able to recall roughly the same number of specific techniques for each major topic. For example, in relationship to the major topic of root finding, a student might list the techniques: Newton-Raphson method, bisection method, secant method and more. To get credit, students must associate the techniques to the correct major topics.

3. Students in the game-based course were significantly more likely to demonstrate that they know how the methods work, and they are more likely to demonstrate appropriate uses and/or limitations of the techniques. For example, within the topic of root-finding, a student would get credit if she mentions that the Newton-Raphson method generally converges faster than bisection. She would get credit if she mentions the bisection is guaranteed to converge to a root, while Newton-Raphson can diverge wildly.

4. Students in the game-based course were significantly more likely to recognize how techniques depend on one another. For example, the technique for performing finite differences of unequally spaced data requires that one first fit the data with a polynomial. There are dozens of other examples that game-based students are much more likely to recognize than their counterparts who take traditional textbook-based numerical methods courses.

Results (1) and (2) correspond to item (A) in the definition of “knowing more” above. They reflect the students’ ability to regurgitate a list of chapters an sections that they covered in their texts. Given that students taking the traditional textbook-based courses are exposed to more numerical methods topics, we had anticipated that they would list more topics and techniques on their concept maps, compared to students in the game-based courses. This turns out not to be the case.

In contrast, results (3) and (4) above indicate higher levels of learning. While characteristics (B) and (C) are almost absent in the control group, the game-based students show ample evidence of deeper understanding. Our interpretation of the results, within this framework, leads us to postulate that the game-based course in computational methods builds a better foundation upon which students can develop expertise in the field.
4 Integrating Course Content into the Game

The primary purpose of this article is to describe the new video game-based computational methods course we have created. In Section 1, we mentioned that we “co-developed” NIU-Torcs. The video game we use is derived from an open-source video game called Torcs (http://www.torcs.org), which is available under the GNU Public License. NIU-Torcs borrows most of the graphics engine of Torcs. Among other things, we have contributed to the game a higher fidelity simulation of the car’s physics, including the engine, transmission, differential, suspension, tire mechanics, and more. Furthermore, we gave the game a series of “levels.” Specifically, we programmed a sequence of events – combinations of cars, tracks, and objectives – which the player/student must work through in order to succeed. In creating NIU-Torcs, we sought to straddle the boundary between rigorous engineering simulation and an accessible video game that could guide students through engaging and authentic engineering problems.

4.1 Simple Driving

The first task we give the students with the video game is to complete one lap around the track without crashing into anything. The students’ begin by writing driving programs that turn the steering wheel to the right as the car enters a right turn, and vice versa when the car enters a left turn.

Of course, the appropriate way to do this is to construct a feedback controller. This may sound complicated for a group of undergraduates who have not taken their first control course. However, almost all students drive. They don’t need any elaborate calculations to keep their own (real) car on the road. The solution is intuitive: when the car is driving to the left of the desired path, the driver should simply nudge the steering wheel to the right, and vice versa. By continuously monitoring where the car is in relation to where it ought to be and taking corrective action, it is possible to navigate the serpentine track and complete the circuit.

To drive the car around the track (nothing too fancy) requires about a dozen lines of computer code. We work it out during lecture. And when the proportional controller fails for all but the slowest car speed, we trouble-shoot in class. We determine that the driver needs an anticipatory effect. The remedy is relatively simple to intuit and implement, thus creating a stable steering controller.

After class, students must try it out on their own car. Then, to demonstrate that they truly understand the simple driving program, students are given the task of getting their car to drive around the track in the opposite direction (clockwise when viewed from above). The task is not trivial. Students must realize (on their own) that they must redefine the program’s representation of “left” and “right”; also, they must compensate for a discontinuity in relative heading angle.

4.1.1 What the exercise accomplishes. In this first task, there is no numerical methods content. However, the exercise, does serve several important purposes. First, it gets students back in the hang of programming, similar to Chapter 2 of Chapra and Canale, one of the most popular textbooks.

The programming also serves the goals for MEE 381 outlined in Section 2.1. The assignment gives students their first exposure to object oriented programming. They are given a skeleton to a C++
class for their driver. In order to get their driver to work, they must program one of the methods of
the class.

Perhaps most important, though, this first assignment motivates and sets the stage for future learn-
ing. Naturally, the engineering students want to see how fast they can get their cars to go. After
completing their homework assignment, they begin experimenting. In the straight sections of track,
they put the pedal to the metal, giving the car its maximum acceleration command. They know,
only vaguely, the best time to shift gears. So they do what they can. Also, before a straight section
of track ends and a turn begins, the driving program must slow the car to a speed at which it can
navigate the curve without sliding off the road.

Early on, we give students tools with which they can observe the behavior of their car in more detail
than simply watching a computer-generated animation of the vehicle on the screen. We provide an
interface to the students through which they can specify run-time data to be saved to a telemetry
file. We also provide Matlab scripts that allow students to closely inspect performance data. An
example is shown in Figure 3.

![Graphical tools to inspect driver performance data.](image)

With a healthy dose of competitive encouragement from their classmates, students begin encoding
new driving strategies on their own. The strategies are ad hoc, based on trial and error. Quickly,
students learn the potential value of pursuing a more formal and mathematical approach based on
physical principles. The exercise provides authentic motivation to learn numerical methods that
lasts throughout the semester.

4.2 Shifting Gears I

The most challenging aspect of driving on a straight section of track in NIU-Torcs is deciding when
to shift gears. For each pair of consecutive gears, there is an optimal speed at which to shift gears.
The optimal shift point is that which produces the greatest possible acceleration.
To find the optimal shift points is not a simple self-contained exercise that fits conveniently within the confines of a single homework assignment. Like real-world engineering problems, to achieve the goal requires students to conquer several technical hurdles and then piece together the facets into properly functioning whole. Below, we describe these separate pieces, the computational techniques students must learn, and how the pieces fit together to reveal the optimal instances at which to shift gears.

4.2.1 **Step #1: Creating a speedometer.** Having a good estimate of the car’s speed is a critical element of determining optimal shift points, as well as other driving tasks such as braking, cornering and general maneuvering.

In the process of simulating the car’s dynamics *NIU-Torcs* keeps track of the velocity of the center of mass. However, the simulation does not share this information with the driver. Instead, the driver must estimate the speed of the car in the same way that real cars do: by sensing the rotation rates of the wheels.

One way to estimate speed is to multiply the rotation rate of one of the wheels by that wheel’s tire radius. But which wheel? If they didn’t realize it before, students learn that the wheels rotate at different rates as the car executes a turn. The difference will depend on whether the car over-steers or under-steers. Also, the driven and non-driven wheels rotate differently. At times, the limited-slip differential locks up as well.

Generally, there is no single best way to estimate speed. The purpose of the exercise is to get students to think and explore. Furthermore, it gets them to write computer code. The speedometer is the students’ first C++ class that they write from scratch.

4.2.2 **Step #2: Creating an accelerometer.** As we shall describe soon, our technique for determining optimal points requires students to estimate the acceleration of the car. The acceleration, of course, is the time derivative of the car’s speed. Since the speed is calculated only at discrete times, the goal of determining optimum shift points provides students with an authentic reason to want to learn finite difference approximations.

In class and in assigned reading, students are exposed to the technicalities of forward difference, backward difference, and central difference techniques of various orders. These details take on added meaning in the minds of students who are invested in a project where the nuances matter. In this case, it is best to use a second order backward difference scheme. And since the driver routine is called over unequal time intervals, students need a difference scheme suitable for unequally spaced data.

To handle the unequally spaced data, students need to learn the basics of polynomial interpolation, and the Lagrange interpolation polynomial specifically. Students, therefore, learn numerical methods topics in an order that makes practical sense, rather than an order determined by the taxonomy of some textbook author.

4.2.3 **Step #3: Collecting acceleration data.** Next, students need to collect acceleration data for their car. They are given an oval track with long straights to drive upon. (See Figure 4.) At the beginning of one of the long straights, the computerized driver brings the car to a complete
stop. After a brief pause, the driver puts the car in first gear, releases the brake, and sends a full throttle gas pedal command to the car. As the car accelerates down the straight section of track, the student’s driver program calculates and records speed and acceleration data.

When the car reaches the end of the straight section of the track, it slows down, makes a gentle left turn, and then brings itself to a stop at the beginning of the next long straight. The whole process is repeated again, but this time the car is placed in second gear. The process repeats until all gears have been tested.

When testing is finished, students have acceleration versus speed data for all forward gears of the car. Figure 5a shows an example for the small sports car (a Bizzarrini). Of course, the discrete data are point-wise approximations to a continuous curve which gives the maximum possible acceleration of the car for a given speed and gear.

Here, the optimal shifting strategy comes into focus. If the goal is to achieve the fastest possible speed in the least amount of time, then students want acceleration always to be as large as possible. Therefore, one should shift from first gear to second gear when the car reaches the speed for which the first gear acceleration curve intersects the second gear acceleration curve. The shift points from second gear to third gear and from third gear to fourth gear can be found similarly using the other curves.

### 4.2.4 Determining Shift Points

Given the discussion above, one can, in principle, use the naked eye to read the shift points off the acceleration vs. speed curves shown in Figure 5. However,
Figure 5: Data depicting car acceleration versus speed in each of the car’s four gears. The data are used to determine optimal shift points.

the students are going to need to determine proper shift points for every different car they drive. In the final exam, they have to race a car they have never seen before. (See the video [link](http://www.ceet.niu.edu/faculty/coller/video.htm).) Therefore, the process needs to be automated. We make them write computer code to determine the intersection points.

Finding where the curves in Figure 5 intersect is a root-finding problem. Because the Newton-Raphson method is probably the best known root-finding algorithm, we feel obligated to teach it. The students, though, are engaged. They are learning root-finding with a purpose. Quickly they discover that Newton-Raphson is not appropriate for their problem. The method requires a derivative, and since we are trying to find the root of a function defined by noisy empirical data rather than an analytic expression, the derivative is likely to be corrupted.

Students are responsible for finding, within their textbook, a method that is appropriate. Most choose bisection. Some choose false position. As with everything in the course, students must implement their method of choice. In doing so, they have an automated means of calculating optimum shift points. They can accelerate from zero to 120 mph in the minimum possible time.

4.3 Selecting a New Transmission

The acceleration curves in Figure 5 strongly depend on the gear ratios in the transmission. Different gear ratios (keeping the engine constant) would yield different acceleration curves and hence different shift points. The time required to accelerate from zero to 100 mph, therefore, depends on the gear ratios that the driving program has available to it.

In the next exercise, we give students acceleration vs. speed data for the sports car (Bizzarrini) with five different transmissions. The students are asked which transmission they would like in their Bizzarrini. The students are not allowed to test-drive the different transmissions. Instead, they must make their decisions based on acceleration versus speed curves that we give them. A well-calibrated eye might be able to detect that one or two of the transmissions are inferior. However, to determine which is the best requires analysis.
For their analysis we ask students to calculate how well each transmission would perform in a 500 meter drag race. Thus, students must write a small program to integrate the system of differential equations

\[
\frac{dx}{dt} = v, \quad \frac{dv}{dt} = a(v),
\]

subject to the boundary conditions

\[
x(t = 0) = 0; \quad v(0) = 0; \quad x(t_f) = 500 \text{ m}.
\]

In Equation (1), the quantity \( a(v) \) is acceleration as a function of speed \( v \). This is given to students in the form of acceleration data like that shown in Figure 5a. They get one set of data for each transmission.

The value of \( t_f \) which satisfies the last boundary condition in (2) is what student want to determine; it represents the time it takes the car to complete the race. Their task is to integrate (1,2) for each of the transmissions. The one with smallest value of \( t_f \) is the transmission with best acceleration performance.

In this assignment, students learn and apply the fourth order Runge-Kutta technique. In getting the correct answer, there is more at stake than just a homework grade. The transmission they choose will be the one they must use for the remainder of the semester.

### 4.4 Shifting Gears II

As students progress farther into the game and are given more cars to drive, they eventually get to the Humvee (Figure 6). It turns out that the automobile simulation within NIU-Torcs is on the verge of being stiff. The time scales associated with the tire slip mechanics are considerably smaller than the bulk time scales of the car’s dynamics. The difficulty is exacerbated for heavier vehicles.

**Figure 6:** The weight of the Humvee causes corruption in the acceleration data and thus requires a different approach to computing shift points.
The weight of the Humvee causes the wheels to chatter slightly in the simulation. The chatter is not noticeable in the animation of the Humvee’s dynamics. However, if we attempt to take a derivative of the wheel rotation rates, as in Section 4.2.2, the noise gets amplified dramatically and the acceleration signal is almost meaningless.

This is the type of thing that happens all the time in the real world, and it serves a good lesson for the students. To get around the noise issue, we formulate a one dimensional unconstrained optimization problem. Actually it is one optimization problem for each gear change. In these optimization problems, one can either maximize speed for a fixed distance traveled, or minimize the time it takes to travel that distance. Either way, we cover another computational technique in an authentic manner that has meaning for students.

### 4.5 Speed Control & Improved Steering

The next task for students is to write code for a speed controller. Essentially, we are asking them to create a cruise control. This is an important feature. For example, when going around a constant radius turn, the driver is going to want to maintain a constant speed just slightly below the speed at which the car begins sliding laterally off the track.

We start with a simple proportional controller of the form

$$u = -k_p(v - v_{des}).$$  

(3)

Here, $v$ is the current speed, $v_{des}$ is the desired speed, $k_p$ is the proportional gain. The control variable $u$ represents how much to step on the gas when $u > 0$ and how much to step on the brake when $u < 0$. Although most students in the class have not yet taken their controls course, the framework in (3) is familiar. It is similar to the steering control students implemented at the beginning of the semester (Section 4.1).

After encoding the controller and running the simulation, students realize that it doesn’t work quite as well as they expected. The car always reaches a speed a few miles per hour shy of the desired speed.

The steady state error, of course, can be eliminated through integral action. Furthermore, stability can be enhanced with derivative action. Thus the speed controller takes on the following structure:

$$u = -k \left[(v - v_{des}) + k_I \int (v - v_{des})dt + k_D \frac{dv}{dt}\right].$$  

(4)

To implement the so-called PID controller, students must write a routine to perform the integration. They already have an accelerometer (Section 4.2.2) to perform derivative action. After an hour or two of trial and error, they devise very effective speed controllers.

Upon closer inspection, students discover that there is a steady state error associated with their steering controller too. If they write the code for their PID controller (4) well, students can conveniently re-use it (without derivative action, $k_D = 0$) in their steering controllers.
4.6 Cornering Speed

Everyone who has driven a car, ridden a bicycle, or pedaled a tricycle knows that the tighter the turn that one tries to maneuver through, the slower one has to travel. In the next exercise, students must determine how fast their cars can travel through turns of different radii.

We take an empirical approach. Students are given the wide circular track shown in Figure 7. Students write a program to drive the car around a circular path of a given radius, and then slowly increase the speed until the car begins sliding laterally. The experiment is repeated for several different turning radii. Experimental data points are shown with circular symbols on the right half of Figure 7.

![Figure 7](image.png)

**Figure 7:** Students perform computational experiments in order to determine the handling behavior of the car.

Suppose the car is driving on a race track, approaching a turn with a radius of curvature of 200 m. How fast should the car go through the turn? The radius of 200 m lies outside the range of the experimental data. Therefore, the driving program is going to have to extrapolate the data to determine the maximum speed around the large radius turn.

Students are able to do this by fitting the data, in a least squares sense, to a polynomial. We do not tell the students what order the polynomial should be. Instead, they learn it by trial and error. When they try a linear fit the agreement is rather poor. In fact, for the data shown in Figure 7, the straight-line curve fit intersects the $v = 0$ at a radius of -56 m. Of course, this is nonsense.

Students discover that it is best to use a quadratic polynomial to fit the data:

$$ r = a_0 + a_1 v + a_2 v^2. \tag{5} $$

This structure is not only justified by the fact that the fit (dashed curve in Figure 7) looks excellent. The quadratic relationship of the variables derives from the fact that centripetal acceleration is proportional to $v^2$. 
In this exercise, students learn how to formulate nonlinear regression problems. Furthermore, they get to apply their linear algebraic equation solver to solve for the coefficients $a_0$, $a_1$, and $a_2$ in (5). Again, learning is authentic in that the problems students are asked to solve a real engineering problems. There are consequences (beyond getting a few points deducted) if the calculations are not performed correctly. Choice of technique and other implementation details have significance.

As in the case of calculating shift points (Sections 4.2 and 4.4), the speed curve calculation must be embedded in the driver code. In the final exam/project, students get new cars that they have never driven before. Their driver programs must calculate all the car’s relevant performance characteristics within minutes before the car is raced.

### 4.7 Braking

Generally speaking, the best racing strategy is to drive as fast as possible when the car is in a straight section of track. While in turns, the car should drive as fast as the turn will allow as determined in Section 4.6. In transitioning from a straight to a turn the car typically has to slow down. So as not to bleed speed unnecessarily, students need to calculate the last possible moment to begin applying the brakes.

To accomplish this task, students write programs to collect speed versus distance data while issuing a range of different brake commands. The brake data is fitted in a least-squares approach much like the cornering data in Section 4.6. This time, however, students must select a functional relationship between speed and distance so that the curve passes directly through the origin.

After writing the computer code to implement their braking and cornering strategy, students have a car that can drive much more quickly and smartly through the circuit that their drivers at the beginning of the semester.

### 4.8 Maneuvering

In the final formal assignment before students are completely unleashed to work on their final project, students are given the task of maneuvering quickly around obstacles. (See Figure 8.) The steering controller will keep the car close to any physically realizable path that we give it. The task is to choose a smooth path with relatively little curvature so that the car can drive through quickly.

For this problem, students learn how to construct splines. While it is common to use cubic order splines, this choice makes particular sense in the maneuvering application. The cubic order guarantees that path curvature (related to centripetal acceleration) is continuous. Furthermore, the so-called “natural” end conditions are convenient since cars enter the maneuver from a straight path.
4.9 Final Project

Finally, students in MEE 381 complete a project in which they have to put their numerical methods and programming skills to creative use. Both times that we have offered the course, the project was a friendly competition. In the first year, it was a standard race. In the second year, it was a more elaborate team race/pursuit game. The reader is encouraged to watch the video at www.ceet.niu.edu/faculty/coller, which contains footage of the final projects.

In the final event of the course, students are required to drive cars with configurations that they have never experienced before, and drive on tracks that they had never seen before. Therefore, one of the first things students do, in preparation for the final event, is develop crash recovery strategies. In general, the crash recovery routines generally need to get the car back on track after it slides off the road and is pinned against the wall, in the slippery grass. Additionally, students often encode routines for finding improved driving paths. Students devise strategies that incorporate combined braking and cornering. In team-based events, a whole host of other issues arise that students must solve or resolve, often using the numerical methods techniques and programming skills that they have developed.

4.10 Inductive Learning

It is not clear whether the video game has been incorporated into the numerical methods course or whether numerical methods have been incorporated into the game. What is clear is that the numerical methods course is much more inductive than it was before we included the video game. Before, we would teach a technique or theory first, and then apply it to a series of homework problems. Now, we start with the problem, and then learn the material necessary to solve the
problem. As Hamming argues in his classic text\textsuperscript{14}

... computing is, or at least should be, intimately bound up with both the source of the problem and the use that is going to be made of the answers – it is not a step to be taken in isolation from reality.

Because of our insistence on letting the embedded engineering problems dictate course content, and because we place an emphasis on programming, we do not cover as many techniques as a traditional numerical methods course. However, results outlined in Section 3.2 indicate that students taking the game-based course generally recall as many techniques as students taking a more traditional textbook-based course.

5 Comparison to Traditional Textbook Problems

For comparison purposes, we carefully examined the end of chapter exercises in two of the most popular textbooks that mechanical engineering faculty adopt for their numerical methods courses. We refer to these two texts as “Book 1”\textsuperscript{8} and “Book 2”.\textsuperscript{20}

Neglecting the chapters on partial differential equations, there are a total of 748 end of chapter exercises in Book 1 and 680 in Book 2. For each of the texts, we have classified the problems into four categories: (NCE) problems that have No obvious Connection to Engineering; (CPU) problems based on the theory and generalizable application of the course material; (AENG) Artificial ENGineering problems; and (ENG) ENGineering problems. In Section 5.1, we elaborate more on our definitions of these categories. Figure 10 shows how the four categories of problems are distributed in the two textbooks.

![Figure 10: Distribution of the four categories of problems are distributed in Book 1 and Book 2.](image)

5.1 Categories of Textbook Problems

Students typically decide to pursue mechanical engineering because they like to build things and to create things mechanical. We suspect that learning to perform mathematical calculations
on computer is usually not a motivating factor. With this in mind, we have created a scheme for classifying textbook problems.

5.1.1 Category NCE: Problems with No obvious Connection to Engineering. There is a large class of textbook problems which ask students to perform tasks that have no apparent connection to engineering. For example, problem 5.3 from Book 2 asks students to find the real root of the polynomial
\[ f(x) = -25 + 82x - 90x^2 + 44x^3 - 8x^4 + 0.7x^5. \]  
(6)
The ability to find roots to equations like (6) might be very important in engineering analysis. However, to the novice undergraduate engineering student, it looks like just a math problem with no connection to reality, and no connection to the things that got her interested in engineering. As seen from the charts in Figure 10, these types of problems appear more frequently than any other type of problem in both of the representative textbooks.

5.1.2 Category AENG: Artificial ENGineering problems. Upon reading the hundreds of problems in the two textbooks, it is clear that both authors made a concerted effort to place the problems in an engineering context. Consider the following problem from Book 1:

**Problem 2.6.** The normal stress induced at the inner fiber of a torsional helical spring is given by
\[ \sigma_i = \left\{ \frac{4C^2 - C - 1}{4C(C - 1)} \right\} \frac{Mc}{I}, \]  
(7)
where \( I = \frac{\pi d^4}{64}, \ c = \frac{d}{2}, \ C = \frac{D}{d}, \ M \) is the bending moment, \( D \) is the mean coil diameter, and \( d \) is the wire diameter. Find the value of \( C \) that corresponds to a stress of \( \sigma_i = 55 \times 10^3 \) psi, when \( M = 5 \) lb-in and \( d = 0.1 \) in.

In this problem, there is a concrete connection between the mathematical problem and something physical. However, we have two criticisms of the homework exercise.

First, the authors do not derive Equation (7); nor do they justify it in any way. Therefore, in order for students to make the connection to engineering, they have to take the author’s word that a connection exists.

Second, we ask why would a student care about the normal stress at the inner fiber of a torsional helical spring? Is it a lingering question that she has carried over from her mechanics of materials course? Does it explain why the cantilever brakes on her mountain bike failed? Not likely.

Because the problem is placed in an engineering context, students might realize better that there may eventually be a need for the computational methods that they are learning. “Potential future benefit”, however, is not known to be a factor that motivates students to learn.24

For these reasons, we categorize the example problem in this section as an artificial engineering exercise (AENG). Although it is couched in engineering language, the context is mostly window dressing for a mathematical problem that provides little more meaning to students than problems in the NCE category.
5.1.3 Category ENG: Engineering problems. Unfortunately, one of the smallest categories of problems in the two representative books (Figure 10) is that of true engineering problems. We define such engineering problems simply as those that are presented in an engineering context, but do not fall into the trappings of the AENG category.

As indicated in Figure 10, Book 1 has about twice as many engineering problems (as a percentage) than Book 2. The difference derives mostly from the fact that Book 1 has several chapters devoted only to engineering case studies. In these chapters, the authors work out examples from chemical, biomedical, electrical, civil, and mechanical engineering. They develop the problems fully, demonstrating how numerical methods concepts can be put to use in practical problems. Because many of the subsequent homework exercises relate directly to the well-developed examples and narrative within the chapter, we have labeled them as engineering problems. Unfortunately, however, many of these problems merely ask students to perform the same calculation that was given in the chapter, except with different parameter values. We feel that it is unfortunate that the authors of Book 1 fail to take advantage of the framework they create.

Another criticism we have of some of the so-called engineering problems, in both books, is that the numerical methods techniques that students are asked to apply have no clear benefit. The issue is most clearly seen in electric circuit and truss problems. These are the types of problems that students have seen before in previous classes. They are well-recognized engineering problems. However, the circuit and truss problems are so small that they can be easily solved by hand. From the perspective of the student, we are teaching him a complicated technique that allows him to perform computations that are just as easy, if not easier, to perform by hand. One of the most important reasons to use a computer to solve truss problems or circuit problems is that one can explore the behavior of really big trusses or complicated circuits. Computational methods in this context allow one to perform parametric studies involving hundreds or thousands of design iterations at the click of a mouse. The student does not see this benefit in many of the engineering problems. In Book 1 and Book 2, 45% and 37% of the engineering problems, respectively, are ones for which there are good alternative approaches, and there is no clear need to use computation.

5.1.4 Category CPU: Computational problems. The last category is for a valuable set of problems that typically serve one of two purposes. First, there are problems which ask students to write programs that solve large classes of computational problems rather than a specific problem. The second type of problem one finds in this category are ones which probe the theoretical aspects of the techniques, in a general sense. Both types of problems are valuable and, from our perspective, under emphasized.

5.2 Common Features of Textbook Problems

In general, regardless of which category the problem falls in, it is clear that almost all the carefully crafted homework problems in the two textbooks share common features:

- Almost all problems are self contained and narrowly focused. The problems are designed to probe specific aspects of specific concepts covered earlier in the chapter. All the information necessary to solve the problem is contained in that part of the chapter. There is no need to
look elsewhere for additional information. The problems have specific correct answers that can be expressed as a number or series of numbers.

- The problems are small. Like the example problems mottled throughout the textbook, the homework problems can be solved with less than a single page of computer code, Matlab script, or Excel operations.

- Almost all problem-based or application-based homework exercises can be solved by canned routines in Matlab, Maple, Mathematica, or other software package. Since this is the case, it is natural for students to question the purpose of the course. Why bother learning Gauss elimination when a modern hand-held calculator will solve a system of linear algebraic equations with a few key-strokes?

6 Closing Remarks

One of the best established theories of learning is that of constructivism. It is based on the premise that our understanding of the world is constructed by reflecting on our experiences. We create our own mental models. Within this framework, learning is a search for meaning. Thus learning starts with issues around which students are actively trying to construct meaning.\(^2\)

From our contrasting descriptions of our game-based numerical methods course in Section 4 and the types of active experiences that typical textbooks offer (Section 5), it is clear that the game-based activities are much better aligned with the constructivist theory. We postulate that this may explain the dramatic differences we see in students who take the game-based course, compared to those who take the traditional textbook-based course. What is unclear is the role of the video game itself. Is it the video game, itself, providing a key motivating ingredient? Perhaps the game based course is more effective because it forces us to structure the learning experiences around activities that are more effective, regardless of the presence of a video game. It is an area of research that we are actively pursuing now that we have a video game with which we can test these and other ideas.

Acknowledgment

The author gratefully acknowledges support from the National Science Foundation under grants 0354557 and 0633162. Any opinions, findings, and conclusions are those of the author and do not necessarily reflect the views of the National Science Foundation. Also, most of the screen-shots in the paper come from the work of MEE 381 students, Spring 2005 and Spring 2006.

References


