Computation Lab for Freshman Engineering Students

Bruce Char, Jeremy Johnson and David Augenblick *

Keywords: Introductory computing, computer-based training, technical computation, mathematical modeling,

Extended Abstract

1 Introduction

Computation Lab has been offered for the past five years to 800-1,000 Drexel freshmen including most freshmen engineering students. Course objectives include:

1. To introduce students to desktop computational tools that can handle numeric, symbolic, and visualization needs for technical problems.
2. To familiarize students with the basic ideas of mathematical modeling and simulation and their application to elementary problem-solving situations similar to what might occur in other undergraduate technical courses.
3. To introduce students to basic programming concepts such as: scripts, variables, loops and conditional statements, user-defined functions and procedures in the context of solving typical engineering.
4. To familiarize students with the application of computation to solve technical problems through the use of modeling and simulation.
5. To introduce students to the software development process in the context of solving such problems.

The current version of the course uses Maple[1], although it is clear that the approach would carry over to other technical computation systems such as Mathematica, Matlab, Octave, Sage, Scientific Python, etc.

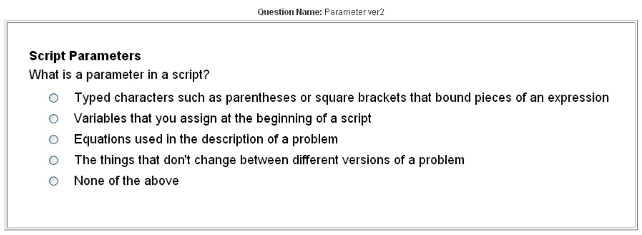
2 Course elements

Scheduling The class consists of 12 two hour lab sessions, held every two weeks during the three terms of the standard Drexel freshman year, along with pre-lab and post-lab on-line exercises. There are three proficiency exams, one given at the end of each term. Lab sessions are conducted in sections of 30-35 students, staffed by an instructor and two assistants (a mixture of regular faculty, graduate TAs, and undergraduate assistants). In Fall 2011, we had ten instructors and fifteen undergraduate assistants for the 33 sections. On-line work is graded automatically by Maple TA[2]. Lab work is graded by instructional staff during the lab session.

Pre-lab activity In lieu of lectures, pre-lab reading and demonstration videos are available on-line. A required pre-lab on-line quiz (see Figure 1 for an example problem) checks that a student is familiar with key concepts and knows enough to be able to provide answers to simple problem situations. Figure 2 shows an excerpt of a chapter pre-lab reading.

*Department of Computer Science, 3141 Chestnut Street, Drexel University, Philadelphia, PA 19104 bchar@cs.drexel.edu

Figure 1: A pre-lab multiple choice question. The correct and incorrect options are randomly selected from a larger list of responses, then permuted.



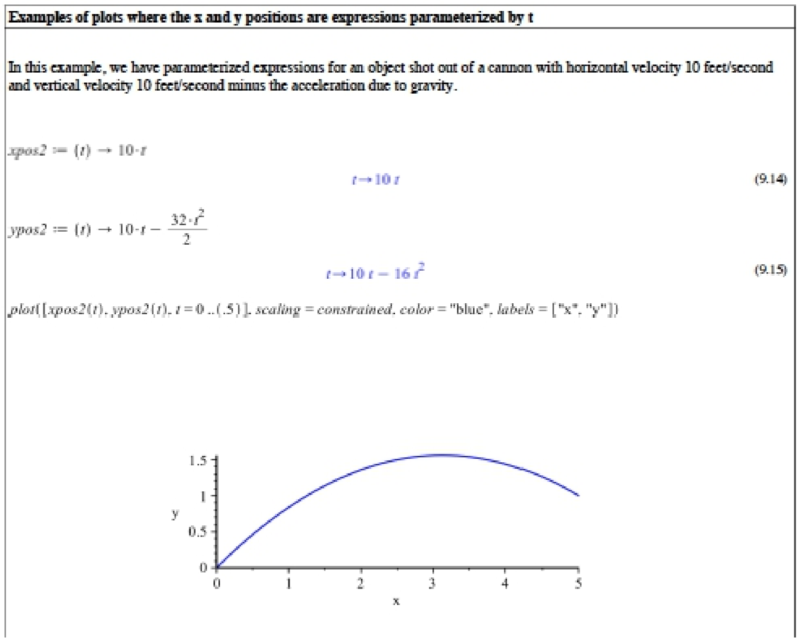
Question Name: Parameter ver2

Script Parameters

What is a parameter in a script?

- ○ Typed characters such as parentheses or square brackets that bound pieces of an expression
- ○ Variables that you assign at the beginning of a script
- ○ Equations used in the description of a problem
- ○ The things that don't change between different versions of a problem
- ○ None of the above

Figure 2: A paragraph from the chapter readings of the first term. It explains how to produce a plot to help understand a population management problem.



Examples of plots where the x and y positions are expressions parameterized by t

In this example, we have parameterized expressions for an object shot out of a cannon with horizontal velocity 10 feet/second and vertical velocity 10 feet/second minus the acceleration due to gravity.

$xpos2 := (t) \to 10 \cdot t$

$$t \to 10\,t \qquad (9.14)$$

$ypos2 := (t) \to 10 \cdot t - \frac{32 \cdot t^2}{2}$

$$t \to 10\,t - 16\,t^2 \qquad (9.15)$$

```
plot([xpos2(t), ypos2(t), t = 0 ..(.5)], scaling = constrained, color = "blue", labels = ["x", "y"])
```

Lab work Each two hour lab typically consists of a 20-30 minute overview and demonstration by the instructor, as well as work on "Part 0", where the staff and students work together in a synchronized fashion through basic concepts and actions. In the remainder of the two hour period students work on additional problems in small groups of two to three students.

Each lab room is equipped with individual laptops, with the students in small clusters to facilitate group interaction. Each group has a computer projector and whiteboard space that they or the instructional staff can use. Instructional staff (typically, in a 11:1 student/staff ratio) circulate among the groups during the period. At the end of the period, each group must save their work (to themselves, using a cloud-based resource or email). Lab work is broken up into objectives and graded by the staff on a "verification sheet" which provides a checklist of what the group was able to accomplish. Time is provided post-lab for students to come back and complete their work under supervision. We find that typically we have paced the material so that at most 10% of the groups need this extra time. Figures 3 and 4 show an example of lab work midway through the course.

Lab work often uses "starter" Maple worksheets that provide some of the code necessary to solve the problem. We have found that "complete the work" exercises are a way to get interesting things to happen in the time limits provided by the lab. Activities where the student is expected to create complete codes are described below in *Post-lab activities.*

One of the advantages of using Maple in the course is that Maple worksheets can be used both to publish the course readings as well as provide environments for the active lab work. This ability to write up calculations in the same environment where the calculations are being done is a feature of modern computational tools that we want the students to experience and use themselves.

Post-lab activity The week after a lab, students are required to do another on-line quiz. The quiz is typically four or five problems, most of which require computation similar to but different from the problem-solving work done in lab. Although it is called a quiz, it is really a homework assignment where the intermediate and final results are checked to verify that the work has been done correctly. Because of the limitations of our on-line grader, we check only for results rather than award partial credit for incorrect code that "looks plausible". However, we believe the lack of credit for incorrect but plausible answers reinforces the dictum that in computation results must be very close to completely correct in order to be useful. The harshness of this requirement is ameliorated by giving immediate feedback, and allowing unlimited retries. Problems are tuned to make exhaustive guessing unlikely to payoff in the time available.

Figures 5 through 7 show a heating problem. The question is programmed to choose the heating parameters so that each student is asked a different variant of the problem.

Figures 10- 13 shows a post-lab problem developed in Computation Lab II, after the students have already been introduced to how to fit data using linear least squares. Variant generation synthesizes a new set of data to be fit for each student. Most of the generation for this question occurs in Maple programming written by the question author invoked from the Maple TA generation script.

Figure 14 describes a problem in a simulated car controller in which the presentation facilities are used to create an animated GIF to be viewed in the student's web browser as part of the problem presentation. Students must write a control program in the Maple language using an API developed and used in a lab. They use it to successfully navigate the car through several variant scenarios. The quiz problem statement shows students the desired scenario outcomes as gif animations similar to what the students should see when they run their control programs through the simulator. Problem variants are randomly selected for a student's quiz.

Proficiency exam Each term ends in a two hour in-class proctored proficiency exam, where students take an exam that consists of a combination of pre-lab and post-lab questions, as well as additional questions that are similar to those used during the term. All questions are posted for on-line practice the week before the exam. Students are allowed access to most of the course written materials and Maple's on-line help during the exam. This allows us to give the students an exam where they are allowed to use the same computation system and information on the exam as they have been during the term. The proficiency exam allows us one major quantified observation of individual performance each term. The results are weighted heavily in the final grade.

The exams are constructed by categorizing the pool of questions as to type and degree of difficulty (which can be corroborated by quiz results from prior use of a question when it has been used before) and then selecting some at random from various categories. Multiple exams are constructed and scheduled into exam time slots at random. This allows us to give an exam across multiple time periods, because there is no extra information available about the questions to the later exam takers.

Overall performance on the proficiency exam is not typically as good (8-15 percentage points less) as for the pre- and post-lab on-line quizzes, which are not proctored, untimed, and can be taken anywhere including during staff consultation hours (often highly popular just before a quiz is due). We believe our on-line quiz grades are an indicator of time spent practicing with feedback, but when unproctored and untimed are not a proof of proficiency in a realistic performance. We would make the same comments about estimating proficiency in a conventional course with manually graded submitted homework.

3 More on Programmable On-line grading

Having an on-line grading system to provide practice and immediate feedback has become an important feature of the course. Deliberate practice is an crucial ingredient in learning [3, p. 236], particularly subjects that require procedural knowledge. Giving students feedback is crucial to the development of their skill as technical problem-solvers, since they may not be efficient at telling how well their efforts are working. Recording evaluation results is useful in large courses where there are many details.

On-line quizzes and exercises, as well as the proficiency exam, are graded by programming an on-line automatic grading system, Maple TA[2]. We have provided the programming for all problems we use, which includes the problem generation, the production of the html and graphics to present the problem to the student's web page, and answer checking.

If the questions are static then students may find it easier to replace learning of facts or processes by learning by rote or Internet look up. Random generation of numerical or structural parameters to the problem allows each student to be given their unique version of a problem without increasing the staff effort in grading. Over 120,000 student responses are automatically graded annually. This allows most staff time to be spent in assisting students in labs and tutoring sessions, and in materials and presentation development.

An important characteristic of the technology is its anytime-availability with immediate feedback. Students can work on pre- and post-lab exercises 24/7. Having immediate feedback and allowing retries typically results in a successful completion rate of over 90% for pre- and post-lab exercises. Immediate feedback with retries also seems to encourage practice by students. In Fall 2011, our class of approximately 900 students took a total of 2867 non-credit, optional practice tests in preparation for the exam. Use of the on-line system has made it easier to use "best practices" such as interleaving worked example solutions and problem-solving exercises, and use of quizzing to re-exposing students to information, as well as use of pre-questions to introduce a new topic. [4]. We have also found it easier to schedule exams and quizzes for large classes over multiple time periods, and allows unproctored student practice outside of class or lab.

Because we make up the questions and their grading ourselves, we are not tied to what a publisher or textbook author has developed. The downside of this is the time it takes to develop a good question, typically five hours or more. We have found that the development cost can be amortized not only over the grading savings from our large class, but also because the variability allows reusability of our questions over several course offerings. To enhance variability, we sometimes use the symbolic computation facilities of Maple to generate structural variations in the problem statement or solution algorithm that should be used.

Figure 1 shows a simple multiple choice problem. The variation there consists of choosing one from several possible correct answers, and four of several possible incorrect responses. The variation comes from random selection rather than varying parameter values.

4 Conclusions

Computation Lab introduces students to the use of mathematical models and computation by having them work and receive feedback on problems from instructional staff by lab work, and through graded on-line work. We use autograding technology to increase the amount of time the instructional staff has helping students with problems they face when problem solving. It allows students to develop proficiency by providing a large supply of practice problems with immediate feedback. On-line grading technology also gives us convenient means to run the course consistently across many time periods and offerings over several terms.

Writing autograded questions with variation incurs a higher cost in software engineering (analysis, testing, and design) than a typical one-paper/single-use assignment or exam question. However, the cost may be amortized by grading savings in several offerings over time of large courses.

References

[1] Maplesoft, *Maple 15 User Guide.* Maplesoft, 2011.

[2] ——, *Maple T.A. 7 User Guide.* Maplesoft, 2011.

[3] Committee on Developments in the Science of Learning for the Commission on Behavioral and Social Sciences and Education, *How People Learn*, expanded edition ed., J. Bransford, A. Brown, and R. Cocking, Eds. National Research Council, 2000.

[4] H. Pashler, P. Bain, B. Bottge, A. Graesser, K. Koedinger, M. McDaniel, and J. Metcalfe, "Organizing instruction and study to improve student learning: A practice guide (ncer 2007-2004)." U.S. Department

of Education, Institute of Education Sciences, National Center for Education, Tech. Rep., 2007. [Online]. Available: http://ies.ed.gov.ezproxy2.library.drexel.edu/ncee/wwc/pdf/practiceguides/20072004.pdf;

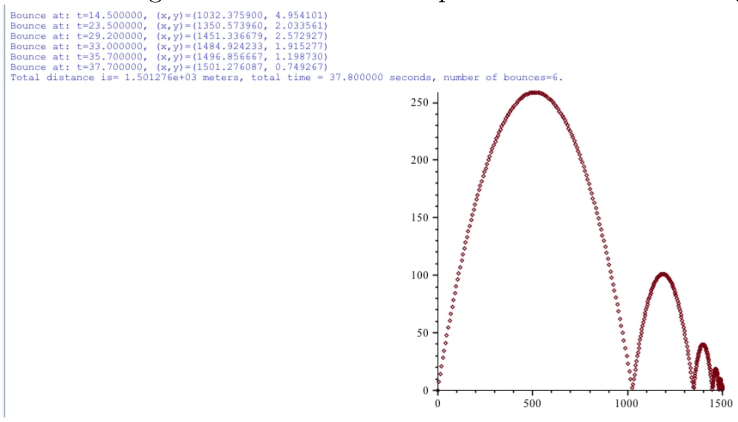
Biographical Information

Bruce Char is Professor of Computer Science in the College of Engineering at Drexel University. He is a past chair of ACM SIGSAM, the special interest group on symbolic computation. He has had a long-term interest in the application of IT tools to technical education. Jeremy Johnson is head of the Computer Science Department at Drexel. He is present chair of ACM SIGSAM, and was the originator of the course. David Augenblick is Auxiliary Professor of Computer Science at Drexel and has been course coordinator for Computation Lab for several years.

Acknowlegments

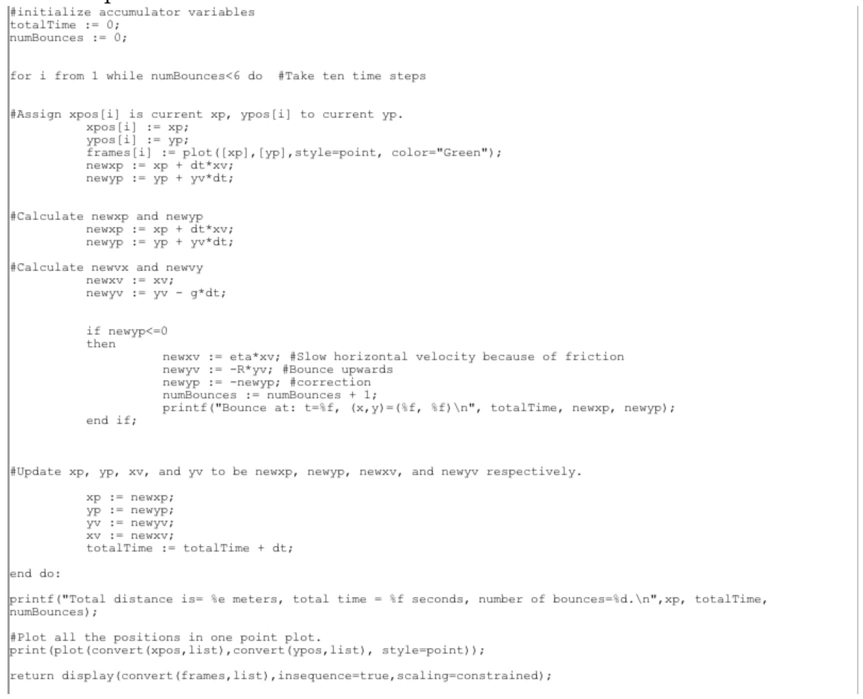
The support of the Drexel University College of Engineering is gratefully acknowledged. Mark Boady helped with question development and supplied some of the usage statistics.

Figure 3: A graph produced by student programming in the second term (Lab 8) to simulate motion of a bouncing ball. Students have already learned about lists and loops. In this exercise they are asked to apply this knowledge to flesh out a time-step simulation of a bouncing ball that loses velocity with each bounce.



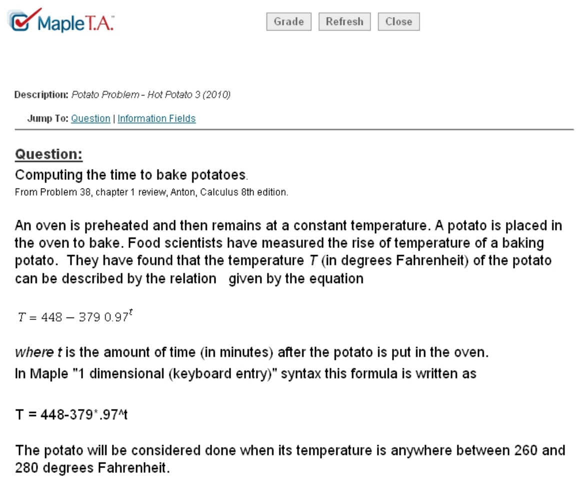
```
Bounce at: t=14.500000, (x,y)=(1032.375900, 4.954101)
Bounce at: t=23.500000, (x,y)=(1350.573960, 2.033561)
Bounce at: t=29.200000, (x,y)=(1451.336679, 2.572927)
Bounce at: t=33.000000, (x,y)=(1484.924233, 1.915277)
Bounce at: t=35.700000, (x,y)=(1496.856667, 1.198730)
Bounce at: t=37.700000, (x,y)=(1501.276087, 0.749267)
Total distance is= 1.501276e+03 meters, total time = 37.800000 seconds, number of bounces=6.
```

Figure 4: Code fragment producing the visualization of 3. Students are given a code framework and expected to fill in pieces of it.



```
#initialize accumulator variables
totalTime := 0;
numBounces := 0;


for i from 1 while numBounces<6 do  #Take ten time steps


#Assign xpos[i] is current xp, ypos[i] to current yp.
          xpos[i] := xp;
          ypos[i] := yp;
          frames[i] := plot([xp],[yp],style=point, color="Green");
          newxp := xp + dt*xv;
          newyp := yp + yv*dt;


#Calculate newxp and newyp
          newxp := xp + dt*xv;
          newyp := yp + yv*dt;

#Calculate newvx and newvy
          newxv := xv;
          newyv := yv - g*dt;


          if newyp<=0
          then
                    newxv := eta*xv; #Slow horizontal velocity because of friction
                    newyv := -R*yv; #Bounce upwards
                    newyp := -newyp; #correction
                    numBounces := numBounces + 1;
                    printf("Bounce at: t=%f, (x,y)=(%f, %f)\n", totalTime, newxp, newyp);
          end if;



#Update xp, yp, xv, and yv to be newxp, newyp, newxv, and newyv respectively.

          xp := newxp;
          yp := newyp;
          yv := newyv;
          xv := newxv;
          totalTime := totalTime + dt;

end do:

printf("Total distance is= %e meters, total time = %f seconds, number of bounces=%d.\n",xp, totalTime,
numBounces);

#Plot all the positions in one point plot.
print(plot(convert(xpos,list),convert(ypos,list), style=point));

return display(convert(frames,list),insequence=true,scaling=constrained);
```

Figure 5: A question with parametric variation, part 1.



Description: *Potato Problem - Hot Potato 3 (2010)*

Jump To: Question | Information Fields

Question:

Computing the time to bake potatoes.

From Problem 38, chapter 1 review, Anton, Calculus 8th edition.

An oven is preheated and then remains at a constant temperature. A potato is placed in the oven to bake. Food scientists have measured the rise of temperature of a baking potato. They have found that the temperature *T* (in degrees Fahrenheit) of the potato can be described by the relation given by the equation

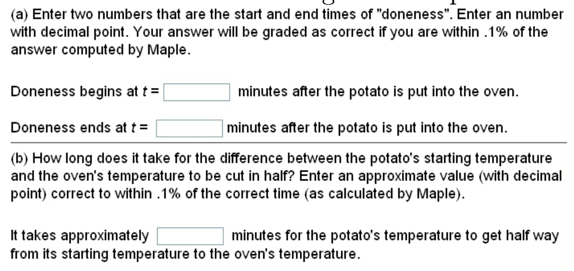
$T = 448 - 379\ 0.97^{t}$

where t is the amount of time (in minutes) after the potato is put in the oven.
In Maple "1 dimensional (keyboard entry)" syntax this formula is written as

T = 448-379*.97^t

The potato will be considered done when its temperature is anywhere between 260 and 280 degrees Fahrenheit.

Figure 6: A question with parametric variation, part 2



(a) Enter two numbers that are the start and end times of "doneness". Enter an number with decimal point. Your answer will be graded as correct if you are within .1% of the answer computed by Maple.

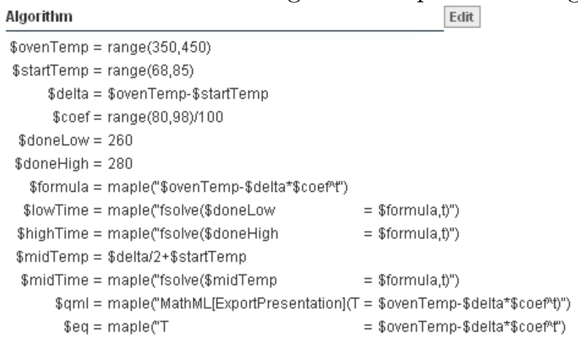
Doneness begins at *t* = ______ minutes after the potato is put into the oven.

Doneness ends at *t* = ______ minutes after the potato is put into the oven.

(b) How long does it take for the difference between the potato's starting temperature and the oven's temperature to be cut in half? Enter an approximate value (with decimal point) correct to within .1% of the correct time (as calculated by Maple).

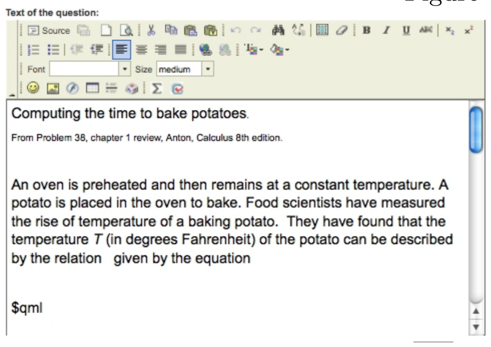
It takes approximately ______ minutes for the potato's temperature to get half way from its starting temperature to the oven's temperature.

Figure 7: Maple TA coding of parametric variation problem



```
 $ovenTemp = range(350,450)
 $startTemp = range(68,85)
       $delta = $ovenTemp-$startTemp
        $coef = range(80,98)/100
  $doneLow = 260
  $doneHigh = 280
    $formula = maple("$ovenTemp-$delta*$coef^t")
   $lowTime = maple("fsolve($doneLow              = $formula,t)")
  $highTime = maple("fsolve($doneHigh             = $formula,t)")
  $midTemp = $delta/2+$startTemp
   $midTime = maple("fsolve($midTemp              = $formula,t)")
        $qml = maple("MathML[ExportPresentation](T = $ovenTemp-$delta*$coef^t)")
          $eq = maple("T                          = $ovenTemp-$delta*$coef^t")
```

Figure 8: Uninstantiated question text



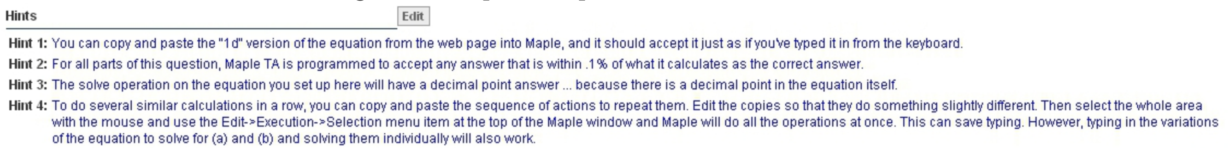
Computing the time to bake potatoes.

From Problem 38, chapter 1 review, Anton, Calculus 8th edition.

An oven is preheated and then remains at a constant temperature. A potato is placed in the oven to bake. Food scientists have measured the rise of temperature of a baking potato. They have found that the temperature *T* (in degrees Fahrenheit) of the potato can be described by the relation given by the equation

$qml

Figure 9: Maple TA question – hints to students



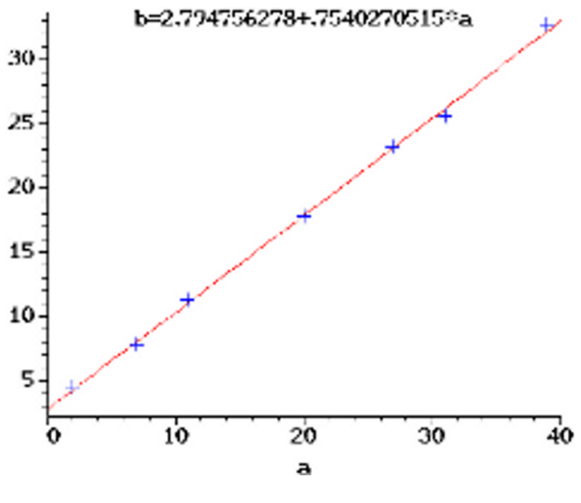
Hint 1: You can copy and paste the "1d" version of the equation from the web page into Maple, and it should accept it just as if you've typed it in from the keyboard.

Hint 2: For all parts of this question, Maple TA is programmed to accept any answer that is within .1% of what it calculates as the correct answer.

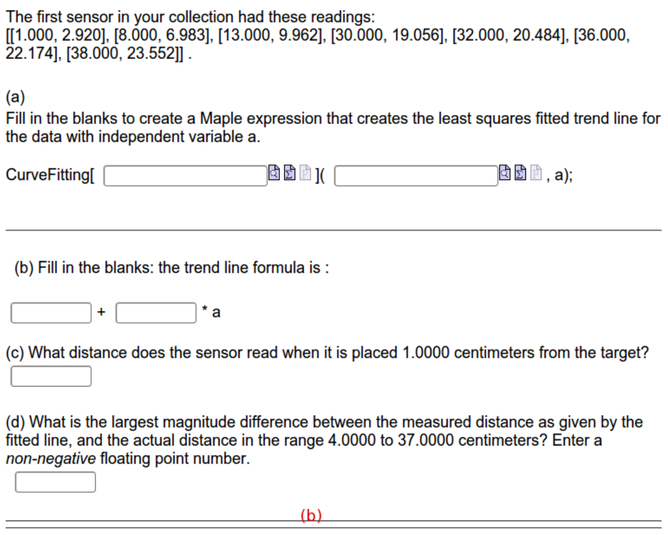
Hint 3: The solve operation on the equation you set up here will have a decimal point answer ... because there is a decimal point in the equation itself.

Hint 4: To do several similar calculations in a row, you can copy and paste the sequence of actions to repeat them. Edit the copies so that they do something slightly different. Then select the whole area with the mouse and use the Edit->Execution->Selection menu item at the top of the Maple window and Maple will do all the operations at once. This can save typing. However, typing in the variations of the equation to solve for (a) and (b) and solving them individually will also work.

Figure 10: (a) Sensor problem plot (b) Sensor question



(a)



The first sensor in your collection had these readings:
[[1.000, 2.920], [8.000, 6.983], [13.000, 9.962], [30.000, 19.056], [32.000, 20.484], [36.000, 22.174], [38.000, 23.552]] .

(a)
Fill in the blanks to create a Maple expression that creates the least squares fitted trend line for the data with independent variable a.

CurveFitting[ ](, a);

(b) Fill in the blanks: the trend line formula is :

+ * a

(c) What distance does the sensor read when it is placed 1.0000 centimeters from the target?

(d) What is the largest magnitude difference between the measured distance as given by the fitted line, and the actual distance in the range 4.0000 to 37.0000 centimeters? Enter a *non-negative* floating point number.

(b)

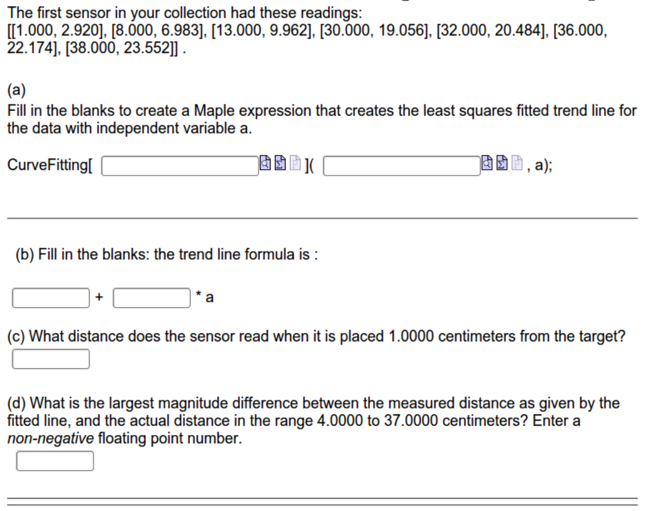


Figure 11: Sensor problem, continued

The first sensor in your collection had these readings:
[[1.000, 2.920], [8.000, 6.983], [13.000, 9.962], [30.000, 19.056], [32.000, 20.484], [36.000, 22.174], [38.000, 23.552]] .

(a)
Fill in the blanks to create a Maple expression that creates the least squares fitted trend line for the data with independent variable a.

CurveFitting[ ](, a);

(b) Fill in the blanks: the trend line formula is :

+ * a

(c) What distance does the sensor read when it is placed 1.0000 centimeters from the target?

(d) What is the largest magnitude difference between the measured distance as given by the fitted line, and the actual distance in the range 4.0000 to 37.0000 centimeters? Enter a *non-negative* floating point number.

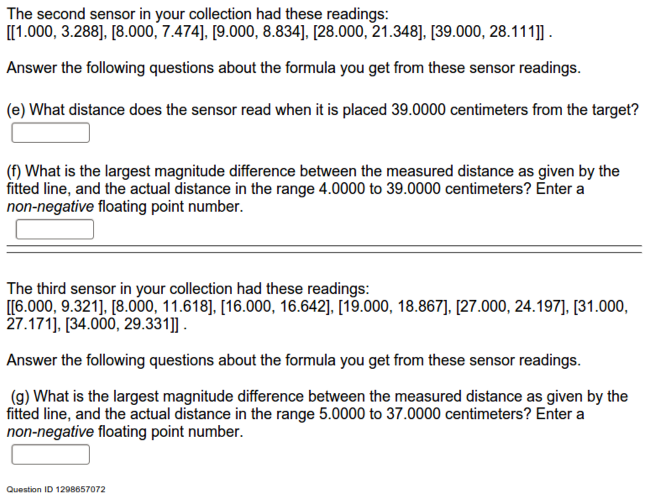


Figure 12: Sensor problem, continued.

The second sensor in your collection had these readings:
[[1.000, 3.288], [8.000, 7.474], [9.000, 8.834], [28.000, 21.348], [39.000, 28.111]] .

Answer the following questions about the formula you get from these sensor readings.

(e) What distance does the sensor read when it is placed 39.0000 centimeters from the target?

(f) What is the largest magnitude difference between the measured distance as given by the fitted line, and the actual distance in the range 4.0000 to 39.0000 centimeters? Enter a *non-negative* floating point number.

The third sensor in your collection had these readings:
[[6.000, 9.321], [8.000, 11.618], [16.000, 16.642], [19.000, 18.867], [27.000, 24.197], [31.000, 27.171], [34.000, 29.331]] .

Answer the following questions about the formula you get from these sensor readings.

(g) What is the largest magnitude difference between the measured distance as given by the fitted line, and the actual distance in the range 5.0000 to 37.0000 centimeters? Enter a *non-negative* floating point number.

Question ID 1298657072

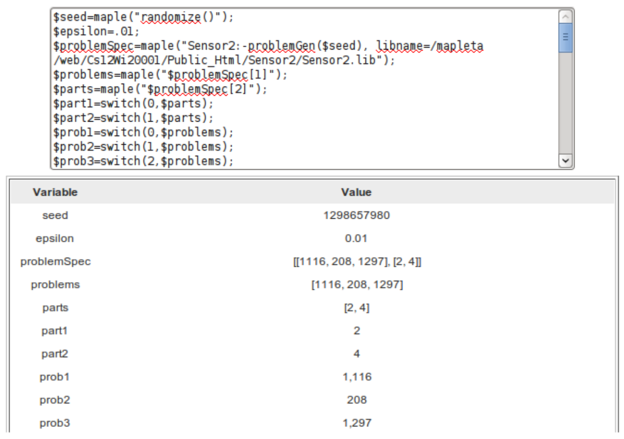
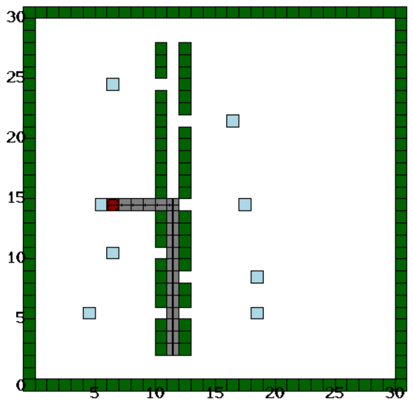


Figure 13: Sensor problem, continued

```
$seed=maple("randomize()");
$epsilon=.01;
$problemSpec=maple("Sensor2:-problemGen($seed), libname=/mapleta
/web/Cs12Wi20001/Public_Html/Sensor2/Sensor2.lib");
$problems=maple("$problemSpec[1]");
$parts=maple("$problemSpec[2]");
$part1=switch(0,$parts);
$part2=switch(1,$parts);
$prob1=switch(0,$problems);
$prob2=switch(1,$problems);
$prob3=switch(2,$problems);
```

Variable	Value
seed	1298657980
epsilon	0.01
problemSpec	[[1116, 208, 1297], [2, 4]]
problems	[1116, 208, 1297]
parts	[2, 4]
part1	2
part2	4
prob1	1,116
prob2	208
prob3	1,297

Figure 14: A frame of an animation for a car control simulator problem .


30
25
20
15
10
5
0
5
10
15
20
25
30