PHYS 210: Introduction to Computational Physics Fall 2013 Homework 2
Version 2: October 10, 2013
Due: Thursday, October 17, 11:59 PM

PLEASE report all bug reports, comments, gripes etc. to Matt: `choptuik@physics.ubc.ca`

Please make careful note of the following information and instructions:

1. Including this page, this handout has 10 pages. Again, do not be taken aback by its length—most of the verbosity is problem description/specification, and, with the exception of Problem 1, solutions for virtually all subproblems can be achieved with just a few lines of *Maple*. Moreover, 2 pages are devoted to a bonus question that you need not address/answer to complete the homework.
2. The following assignment requires
   (a) Using `xmaple` (the graphical version of *Maple*) to produce *Maple* worksheets (Problems 1 and 2).
   (b) Preparing source code for *Maple* procedures in plain-text files that can be input into `maple` or `xmaple` via the `read` command (Problems 3, 4, and should you choose to complete it, Problem 5).
3. Problem 5 is strictly *optional*, and is for bonus credit. In keeping with the intrinsic nature of a "bonus" activity, please try to *minimize* the amount of help/hints you request from the instructor and/or TAs for this problem.
4. In order to complete your homework—especially for Problems 1 and 2 (and the bonus, 5)—it is recommended that you use the computer lab. If you try to use `xmaple` remotely (using, e.g., `Xming/putty` under Windows, or a Mac) you are apt to find the performance unacceptably sluggish. However, for Problems 3 and 4, where you are to write procedures in text files, you should be able to work remotely using command line `maple` running on `hyper`.
5. **IMPORTANT!!** To complete the homework, all of the files specified below must exist and be in their proper locations within your `/phys210/$LOGNAME/hw2` directory. The TAs and I must be able to read all of the required worksheets and text files into *Maple* sessions of our own without encountering errors.
6. Whenever working with *any* worksheet in `xmaple`, be sure to save your work frequently, using, for example, `Ctrl-S`. This will minimize the amount of time and effort that you might lose should the interface crash (as it has been known to do occasionally)
7. **IMPORTANT!!** Although Problem 1 is straightforward, it will take some time to fully complete. You will be provided with some lab time to work on it (and the other problems), but you are advised not to leave its completion until the last minute.
8. Please follow all instructions for each problem carefully, again ensuring that all requested files are in their correct locations—i.e. within subdirectories of `/phys210/$LOGNAME/hw2`—and with the correct names. Also note that any reference to the directory `hw2` below is implicitly a reference to `/phys210/$LOGNAME/hw2`.
9. Do not do any of your work, or save any files, in your home directory or anywhere else that is accessible by your fellow students.
10. Finally, as always, let me know immediately if there is something that you do not understand, or if you encounter serious problems with any part of the assignment.

Problem 1: As an introduction to *Maple* we went through a worksheet that I created, and which was based on Chapter 2 of the *Maple Learning Guide*.

IMPORTANT!! I distributed two versions of the worksheet: the copy handed out on Sep. 20 contained a typo and a misordering of some material, the second, corrected version was given out on Sep. 24. The corrected version is also available online as a Postscript (not PDF) file via `Course Home Page -> Course Notes -> Maple -> Worksheet [PS] showing calculations ....` *PLEASE use the corrected worksheet to complete this problem.*

In your `hw2` directory create a subdirectory `a1`. In that subdirectory, make a facsimile of my worksheet, called `a1.mw`, by entering all of the Maple commands contained within it, and providing annotations for the various sections, commands, etc., as I have done. Recall that I also distributed a hardcopy of Chapter 2 of the *Learning Guide* in its entirety. You may wish to use this for supplemental information if you encounter, for example, difficulties getting commands to evaluate properly—it too is available online through the Course Notes web page.

We went through the procedure for inserting annotations (text comments) in a worksheet during a lab session. For completeness, that information is reproduced below.

Finally, please observe the cautions made in the preamble concerning:

1. The time it may take to complete this problem.
2. The wisdom of frequent use of `Ctrl-S`, or some other save mechanism when working with *any Maple* worksheet.

To create annotations (comments) of the sort I made, use the three icons located roughly under the "Drawing" label on the top tool bar. From right to left these are

- An icon that resembles an hourglass—hovering the mouse over it displays the text "Enclose the current selection in a document block, or create a new one"
- An icon representing the *Maple* prompt: "Insert Maple Input after the current execution group"
- An upper case T: "Insert plain text after the current execution group"

To insert a comment, position the cursor to the immediate right of the prompt on the line *before* the location where you want to do the insertion.

Click on the hourglass icon, and then on the T.

You can then type in plain text to create the annotation. Once the note (section heading, comment etc.) has been inserted, you can alter its appearance (font, font size, style etc.) by sweeping the text and using the icons and pulldowns immediately above the main input/output area.

Since text blocks are always inserted *after* the line on which the cursor is positioned, it is slightly troublesome to begin a worksheet with an annotation.

One way to do so is as follows. First, insert an execution group before the initial command of the worksheet by positioning the cursor beside the corresponding prompt and typing `Ctrl-k`. Using the above prescription, insert the text that is to appear at the beginning of the worksheet after the new execution group. Now delete the execution group before the annotation by repositioning the cursor beside the corresponding prompt, pressing and holding the `Ctrl` key and pressing the `Delete key` two times.

Your annotations do *not* have to match mine *precisely* (i.e. you don't have to use the same font [I used Lucida Sans], font size, style etc.), but, again, you should try to ensure that everything that I have typed into my worksheet is included in yours.

Problem 2: Make the subdirectory `hw2/a2`. Within that subdirectory, use `xmaple` to create a worksheet called `a2.mw` in which the following computations and plotting have been carried out:

$$\frac{\partial^3}{\partial x\,\partial y^2}\left(\sin\left(xy\right)\exp\left(\tanh^{-1}\left(y/x\right)\right)\right)\Big|_{x=2,y=6} \tag{2.1}$$

$$\int \frac{x^5-6x^3+4}{x^2+6}\,dx \tag{2.2}$$

$$\int_{y=1}^{y=3}\int_{x=1}^{x=2} \frac{x^3+4xy+y^2}{x^2+y^2}\,dx\,dy \tag{2.3}$$

Taylor series about $x=0,$ up to and including the $O(x^{10})$ term, of $\sqrt{\cos(x)+\sin(3x)}$ (2.4)

A plot of the error in the above expansion for $0 \le x \le 0.01$ (2.5)

For (2.1), note that $\tanh^{-1}(...)$ denotes $1/\tanh(...)$, *not* an application of an inverse hyperbolic function, and that the result should be a single numerical value. For (2.4) and (2.5), "including the $O(x^{10})$ term" means that the explicit form of the term with x^{10} in it must appear in the expansion. As a concrete example, the following expansion for the exponential function includes the $O(x^3)$ term:

$$\exp(x) = 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + O(x^4)$$

You may find (2.5) a bit more challenging that the other parts of this problem. Define "error" as "exact value - approximate value", and be sure to set `Digits` to a value sufficiently large to produce an accurate plot. Finally, note that you can't plot a Taylor series directly (due to the $O(x^p)$ term that generically appears, and which has no specific value). However, as discussed in class (in the handout "Some Useful Maple Commands", also available via the Course Notes web page), you can readily convert a series to a polynomial using the `convert` command.

Problem 3: Make the subdirectory `hw2/a3` and, within that subdirectory, create two text files, `procs` and `fd2`. The file `procs` is to contain definitions (code) for two *Maple* procedures, `fcn5` and `prod_even_odd`; and **OPTIONALLY** for `plot5`. The file `fd2` will define the procedure `fd2`. The procedures have headers and functionalities as specified below.

Note that `fd2` is to return a *list* of four values. As a hint for dealing with this aspect of its implementation, the description of the subproblem includes the definition of a procedure, `sumdiff`, which returns a 2-element list.

1. *Header*: `fcn5 := proc(x::numeric)`
   Functionality: `fcn5` returns a value defined as follows

$$\begin{array}{ccc} 0 & \text{for} & x \le -1 \\ 2-4\left(-x-\frac{1}{2}\right) & \text{for} & -1 < x \le -\frac{1}{2} \\ -4x & \text{for} & -\frac{1}{2} < x \le 0 \\ 4x & \text{for} & 0 < x \le \frac{1}{2} \\ 2-4\left(x-\frac{1}{2}\right) & \text{for} & \frac{1}{2} < x \le 1 \\ 0 & \text{for} & x > 1 \end{array}$$

2. **IMPORTANT!!:** This sub-problem is now **OPTIONAL**: if completed, it will be graded for bonus credit.
   Header: `plot5 := proc(xmin::numeric, xmax::numeric)`
   Functionality: `plot5` uses the *Maple* `plot` procedure to generate a *single* graph that plots both `fcn5` and `-fcn5`—where `fcn5` is the procedure defined above—on the domain `xmin` $\le x \le$ `xmax`.

3. *Header:* `prod_even_odd := proc(m::integer, n::integer)`
   Functionality: Given two integers, m and n, with $n > m$, `prod_even_odd` returns a value given by

$$\left(\sum_{i=m}^{n} i \mid i \text{ is even}\right)\left(\sum_{i=m}^{n} i \mid i \text{ is odd}\right)$$

   For example

$$\texttt{prod\_even\_odd(3, 12)} = (4+6+8+10+12)\,(3+5+7+9+11) = 1400$$

   If `prod_even_odd` is supplied with arguments m and n that do *not* satisfy $n > m$, it must invoke the statement `error "second argument, n, must be > first argument, m";`

   For example

```
> prod_even_odd(10,10);
Error, (in prod_even_odd) second argument, n, must be > first argument, m
```

   Here and below, recall that execution of the `error` command/statement causes a procedure to automatically and immediately exit, without returning a value.

4. *Header:* `fd2 := proc(f::procedure, x::float, h::float)`
   Functionality: In future lectures we will be studying the topic of *finite differencing*, where derivatives of functions are approximated by algebraic expressions. Specifically, for the case of the first derivative, $f'(x)$, of a function, $f(x)$, of one variable, x, two such approximations that we will consider are

$$f'(x) \approx \frac{f(x+h)-f(x)}{h} \equiv D_f \tag{1}$$

$$f'(x) \approx \frac{f(x)-f(x-h)}{h} \equiv D_b \tag{2}$$

   where h is a strictly positive real quantity, and the symbol $\equiv$ is to be interpreted as "is defined to be". For both (1) and (2) we expect to recover the exact value of the derivative in the limit $h \to 0$.

   The procedure `fd2` has arguments defined as follows:

   (a) `f`: A *Maple* procedure that defines a function, f, of one variable, i.e. the procedure takes a single argument, which you can assume will always be of type `float`. This procedure can be, for example, one of *Maple*'s built-ins, such as $\sin(x)$, or a procedure that is user-defined.

(b) `x`: The specific value of the independent variable at which the approximations (1) and (2) are to be computed.

(c) `h`: The value of h, with $h > 0$, to be used in calculating the approximations.

Return value: `fd2` is to return a length-4 list with elements given by

$$[\ h,\ D_f,\ D_b,\ (D_f - D_b)/h\ ]$$

Also, if $h \leq 0$, the procedure must call `error "h must be > 0";`

Examples:

```
> fd2(cos, 0.0, 0.1)
      [0.1, -0.04995834700, 0.04995834700, -0.9991669400]

> fd2(cos, 0.0, 0.01)
      [0.01, -0.004999960000, 0.004999960000, -0.9999920000]

> fd2(cos, evalf(Pi/4), 0.01)
      [0.01, -0.7106305000, -0.7035594900, -0.7071010000]

> fd2(x -> x^3, 1.0, 0.01)
      [0.01, 3.030100000, 2.970100000, 6.000000000]

> fd2(cos, 0.0, -0.1)
Error, (in fd2) h must be > 0
```

Note that the fourth example uses the $x \rightarrow f(x)$ "arrow notation" for the definition of a function: one might think that simply supplying x^3 as the first argument would produce the same result, but, as you will be able to verify, it does not (and you should try to understand *why* it doesn't).

Once you are confident that your implementation of `fd2` is correct, execute the following at the `bash` command line—*not* in a *Maple* session—*ensuring that the working directory is the directory for this problem.*

```
% pwd
/phys210/<your-login>/hw2/a3

% maple < /home/phys210/hw2/prob3/runfd2 > outfd2
```

Note that the second `bash` command causes `maple` to start, read the *Maple* commands (code) from the file

```
/home/phys210/hw2/prob3/runfd2
```

and redirect the resulting *Maple* output to the file `outfd2`. As you can check, one of the statements in the `runfd2` file is `read fd2;` so *ensure that you have defined the* procedure `fd2` *in the* file `fd2` *before you complete this part of the question.* Also observe that if the file `outfd2` exists you will need to explicitly remove it before issuing the second `bash` command.

Assuming that you *have* coded `fd2` correctly, the file `outfd2` should contain

```
                                    .
                                    .
                                    .
Invoking fd2(x -> cos(x^2), evalf(Pi/4), h) with a sequence of values of h ...

h := .5
fd2 returns [.5, -1.79413318804, -.361960185842, -2.86434600440]

h := .250000000000
fd2 returns [.250000000000, -1.34951596862, -.573966327592, -3.10219856411]
                                    .
                                    .
                                    .
Output from fd2 ends ...
```

where the "vertical ellipsis" (dots) denotes portions of the file contents which have not been reproduced here.

What do you notice about the output from `fd2`? More specifically, considering the sequences of values of D_f, D_b and $(D_f - D_b)/h$ that are returned, do you detect any trends, either within any given sequence, or among two or more of them? If so, what are they? Answer in `README` (as usual you will need to create this file).

If you can (i.e. this part is optional, and you will not lose marks for not answering it), provide explanations/hypotheses in `README` for any of the observations that you have made.

Again, should you have difficulty constructing `fd2` so that it returns a *list* of values, here is a simple procedure that illustrates the basic mechanism (there are more concise implementations, but this one also demonstrates 1) the use of local variables—which you may find convenient in your coding of `fd2`—and 2) the fact that such variables *should* be declared explicitly to be local).

```
#####################################################################
# sumdiff returns a 2-element list: the first element is the sum
# its two arguments, the second is their difference.
#####################################################################

sumdiff := proc(x::numeric, y::numeric)

   local xysum, xydiff;

   xysum := x + y;
   xydiff := x - y;

   # Return the two element list
   [xysum, xydiff];

end proc;
```

Example invocation:

```
> sumdiff(3,4);
      [7, -1]
```

Be sure to test all four procedures using input of your own choosing. The TAs and I must be able to read `procs` and `fd2` into a *Maple* session using the `read` command without encountering errors. We will test your implementations with our own input.

Summary: File inventory for this question:

1. Text file `procs`: Contains the definitions of `fcn5`, `prod_even_odd` and, optionally, `plot5`
2. Text file `fd2`: Contains the definition of `fd2`.
3. Text file `README`: Contains answers to the specific questions posed in part 4, and where some answers/explanations are optional.

Problem 4: Make the subdirectory `hw2/a4` and, within that subdirectory, create a text file `procs` that contains definitions for the following three *Maple* procedures. The procedures must have headers and functionalities as specified.

1. *Header:* `lweave := proc(l1::list, l2::list)`
   Functionality: Given two lists, `l1`, `l2`, of equal length, `N`, `lweave` returns the list with elements:

   ```
   [ l1[1], l2[1], l1[2], l2[2], ..., l1[N-1], l2[N-1], l1[N], l2[N] ]
   ```

   If `l1` and `l2` are not of equal length, `lweave` is to invoke `error "input lists are not of equal length";`

   Examples:

   ```
   > lweave( [2, 3, 6], [w, x, y] );
         [2, w, 3, x, 6, y]

   > lweave( [[a, b], c, d], [1, 2, [3, 4]] );
         [[a, b], 1, c, 2, d, [3, 4]]

   > lweave( [], [] );
         []

   > lweave( [a, b, c], [1, 2] );
   Error, (in lweave) input lists are not of equal length
   ```

2. *Header:* `lsignum := proc(l::list(float))`
   Functionality: Given a non-empty list, `l`, of floating point values, `lsignum` returns a 3-element list containing the number of 1) positive elements, 2) zero elements, and 3) negative elements in `l`, and in that order. Note the use of the "structured" type `list(float)` in the procedure header. If `l` is the empty list, `lsignum` is to invoke `error "argument is the empty list";`

   Examples:

   ```
   > lsignum( [0.0, 6.0, -1.0, 0.0, -2.0, -4.0] );
         [1, 2, 3]

   > lsignum( [-3.0, 1.6, 23.5] );
         [2, 0, 1]

   > lsignum( [] );
   Error, (in lsignum) argument is the empty list

   > lsignum([-3, 1.6, 23.5]);
   Error, invalid input: lsignum expects its 1st argument, l, \
      to be of type list(float), but received [-3, 1.6, 23.5]

   > lsignum([-3.0, a, 23.5]);
   Error, invalid input: lsignum expects its 1st argument, l, \
      to be of type list(float), but received [-3.0, a, 23.5]
   ```

   Note that for the last two examples, and provided that your have used a header for `lsignum` precisely as given above, the error messages are generated automatically via *Maple*'s type-checking mechanism (`3` is not a `float` in the first case [no decimal/radix point], nor is `a` in the second).

3. *Header:* `lminmax := proc(l::list(float))`
   Functionality: `lminmax` takes a non-empty list of floating point values and returns a 2-element list of 2-element lists with constitution:

   ```
   [ [lmin, lmin_index], [lmax, lmax_index] ]
   ```

   Here, `lmin` and `lmax` are the minimum and maximum values in the list, and `lmin_index` and `lmax_index` are the positions in the list of the corresponding values. That is, we have

   ```
   lmin = l[lmin_index]
   lmax = l[lmax_index]
   ```

   If `lmin` and/or `lmax` occur more than once in the list, your procedure is free to return any of the corresponding values of `lmin_index`. However, to be clear, even if `lmin_index` and/or `lmax_index` are *not* uniquely determined, the return value from `lminmax` must always be a 2-element list of 2-element lists, precisely as defined above.

   If `l` is the empty list, `lminmax` is to call `error "argument is the empty list";`

   Examples:

   ```
   > lminmax( [10.7, 2.3, -1.4, 12.6, 6.1] );
         [[-1.4, 3], [12.6, 4]]

   > lminmax( [3.14] );
         [[3.14, 1], [3.14, 1]]

   > lminmax( [1.5, -1.5, 1.5, -1.5, 1.5, -1.5] );
         [[-1.5, 2], [1.5, 1]]

   > lminmax( [] );
   Error, (in lminmax) argument is the empty list

   > lminmax([-3, 1.6, 23.5]);
   Error, invalid input: lminmax expects its 1st argument, l, \
      to be of type list(float), but received [-3, 1.6, 23.5]
   ```

   Note that—as in the `lsignum` case—the error message from the fourth example will be generated automatically, provided you have used a definition for the procedure header precisely as given above.

   Importantly, observe that for the third invocation, and per the comment made previously, the following would also be valid output

   ```
         [[-1.5, 6], [1.5, 5]]
   ```

   and there are, of course, other possibilities. (However, it is not recommended that you spend time trying to produce code that generates one of the other acceptable alternatives!)

Test your procedures thoroughly with various input—invalid as well as valid—including empty lists. Again, empty-list input is valid for `lweave`, but not for `lsignum` or `lminmax`. All three procedure definitions should be commented: the documentation need not be extensive, but should include, at a minimum, a description of what the procedure does. Again, the code for the procedures must be prepared in a *single Maple* source file (plain text file) called `procs`. The TAs and I must be able to read `procs` into a `maple` or `xmaple` session using the `read` command without encountering errors. We will test your procedures with our own input.

Problem 5 (Optional, for bonus credit): Recall from your studies of the kinematics of one-dimensional particle motion, that given the particle's acceleration, $a(t)$, where t is time, the particle's velocity, $v(t)$ and displacement, $s(t)$ are given by

$$v(t) = v_0 + \int_{t_0}^{t} a(t')\,dt' \tag{3}$$

$$s(t) = s_0 + \int_{t_0}^{t} v(t')\,dt' \tag{4}$$

respectively, where t_0 is the initial time for the motion, and $v_0 = v(t_0)$ and $s_0 = s(t_0)$ are the initial velocity and displacement of the particle, respectively. Note that one might often have $t_0 = 0$, but in the context of this problem, we want to retain the freedom to set t_0 to some arbitrary value.

Within the subdirectory `hw2/a5`, create a text file named `kinematics` that contains a definition for a *Maple* procedure having the following header

```
kinematics := proc(a::procedure, t::name, s0::numeric, v0::numeric,
                   t0::numeric, t1:: numeric)
```

Here

1. `a` is a *Maple* function such as

   ```
   t -> cos(t)
   ```

   that is defined using the *Maple* "arrow notation" and that gives the acceleration of the particle as a function of time (the independent variable).

2. `t` is a *Maple* name, identifying the independent variable, and, in invocations of `kinematics` will typically be literally `t`.

3. `s0` and `v0` are numeric values specifying the initial position, $s(t_0)$, and initial velocity, $v(t_0)$, of the particle, respectively.

4. `t0` and `t1` are numeric values giving the initial ($t = t_0$) and final ($t = t_1$) times of the motion, respectively.

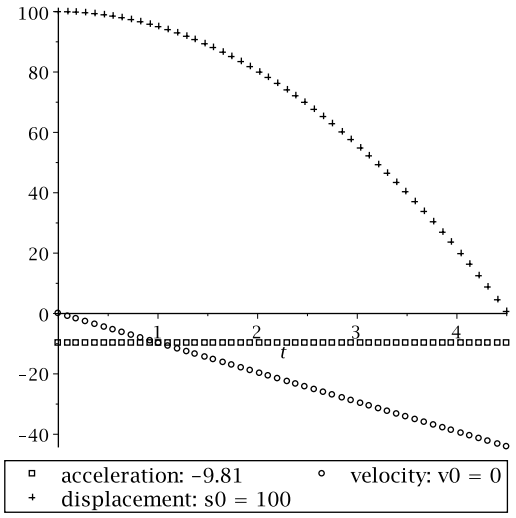
Functionality: Using (3) and (4), your implementation of `kinematics` is to use the *Maple* `plot` procedure to produce a *single* plot showing the acceleration, $a(t)$, the velocity $v(t)$, and the position, $s(t)$, for $t_0 \le t \le t_1$, and where t is understood to be whatever independent variable is actually passed to the procedure. (To get general help on `plot` use `?plot`: you may find the `options` subtopic (use the help menu, or `?plot[options]`) especially useful for this problem).

The plots that `kinematics` generates should include legends that duplicate, as closely as possible, those produced by my implementation, per the examples shown on the next page. To that end you may wish to consider getting help on the topics of

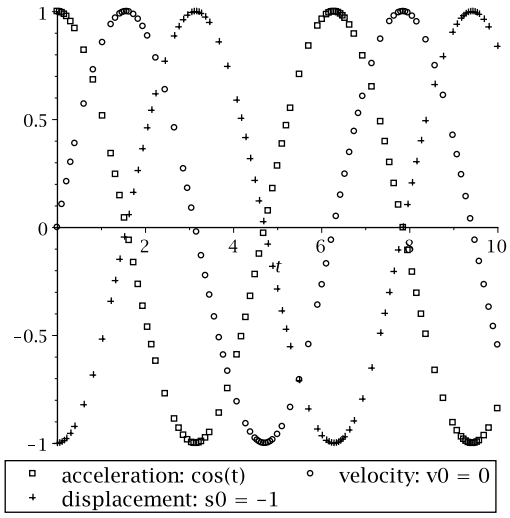
- Legends for plots: see the `legend` section of `?plot[options]`
- The `sprintf` command: use `?sprintf`

Here are some sample invocations along with the corresponding output (which, again, in all cases is a single plot that will appear within the worksheet where it is executed—no other output is required). Observe the use of `T`, rather than `t` in the bottom-right example. In addition to mimicking the legends my version of `kinematics` produces, also try to use the same plotting style and number of symbols per curve (approximately) that my implementation employs (again, refer to appropriate sections of `?plot[options]`, where you will need to determine which sections those are!)

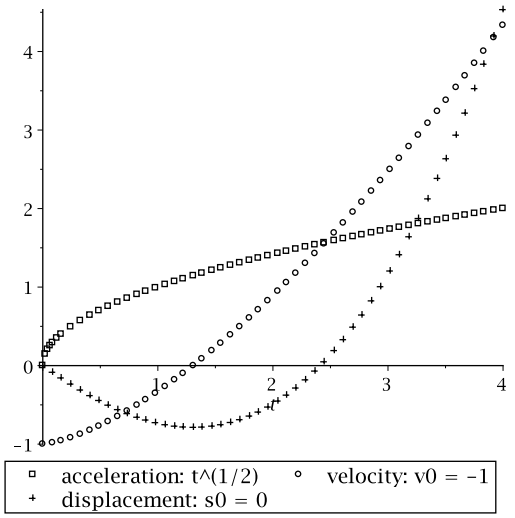
```
> kinematics(t->-9.81, t, 100, 0, 0, 4.5);
```



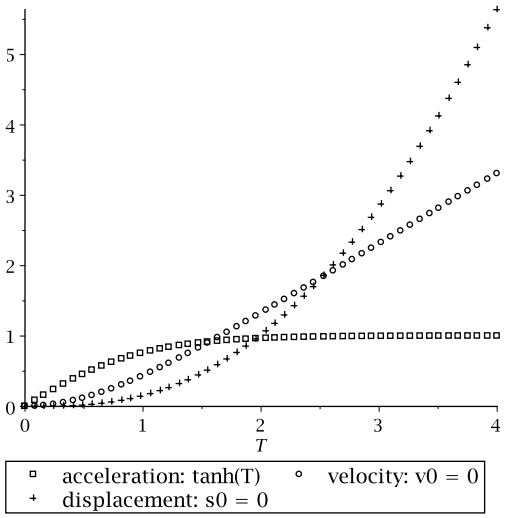

```
> kinematics(t->cos(t), t, -1, 0, 0, 10);
```




```
> kinematics(t->sqrt(t), t, 0, -1, 0, 4);
```




```
> kinematics(Time->tanh(Time), T, 0, 0, 0, 4);
```



Even more bonus (one time offer!): Using an acceleration function, $a(t)$, of your own design (try to be imaginative, but ensure that it's something for which the requisite anti-derivatives exist!), produce a `jpeg` image of the resulting plot produced by your implementation of `kinematics` and proudly include it in your course web page. If you do this, leave a comment in a `README` file in the solution subdirectory that proclaims your heroic effort. Otherwise, there is no need to create `README`.